UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FERRAMENTA COMPUTACIONAL PARA SÍNTESE DE FILTROS ANALÓGICOS E DIGITAIS

TRABALHO DE CONCLUSÃO DE CURSO

Renan Birck Pinheiro

Santa Maria, RS, Brasil

2015

FERRAMENTA COMPUTACIONAL PARA SÍNTESE DE FILTROS ANALÓGICOS E DIGITAIS

Renan Birck Pinheiro

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Elétrica da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Engenheiro Eletricista

Orientador: Prof. Dr. Cesar Ramos Rodrigues

Santa Maria, RS, Brasil

2015

Universidade Federal de Santa Maria
Centro de Tecnologia
Graduação em Engenharia Elétrica

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

FERRAMENTA COMPUTACIONAL PARA SÍNTESE DE FILTROS ANALÓGICOS E DIGITAIS

elaborado por
Renan Birck Pinheiro

como requisito parcial para obtenção do grau de
Engenheiro Eletricista

COMISSÃO EXAMINADORA:

Cesar Ramos Rodrigues, Dr.
(Orientador)

Alexandre Campos, Dr. (UFSM)

Cesar Prior, Dr. (UFSM)

Santa Maria, 08 de julho de 2015.

AGRADECIMENTOS

À minha família e amigos pelo apoio e incentivo durante minha trajetória no curso.

Ao professor Cesar Ramos Rodrigues, por ter me orientado na execução deste trabalho.

Aos meus colegas de trabalho na Chip Inside Tecnologia, que colaboraram na revisão deste trabalho e sugeriram correções e modificações.

“Em algum lugar, alguma coisa incrível está esperando para ser conhecida.”
— CARL SAGAN

RESUMO

Trabalho de Conclusão de Curso
Graduação em Engenharia Elétrica
Universidade Federal de Santa Maria

FERRAMENTA COMPUTACIONAL PARA SÍNTESE DE FILTROS ANALÓGICOS E DIGITAIS

AUTOR: RENAN BIRCK PINHEIRO
ORIENTADOR: CESAR RAMOS RODRIGUES
Local da Defesa e Data: Santa Maria, 08 de julho de 2015.

Uma tarefa comum na eletrônica e nas diversas áreas na qual o processamento de sinais é empregado é o projeto de filtros analógicos e digitais, com os mais diversos objetivos. Porém, para filtros de maior ordem essa tarefa torna-se trabalhosa devido ao grande volume de cálculos envolvidos.

Embora existam *softwares* para a realização de tais projetos, frequentemente ele é proprietário ou apresenta complexidade de uso, o que acaba por restringir sua aplicação. Visando fornecer uma alternativa, propõe-se neste trabalho um *software* livre e de código aberto, desenvolvido na linguagem *Python*, para o projeto de filtros analógicos e digitais; dessa forma, não apenas ele pode ser usado gratuitamente, como pode ser usado como base para outros trabalhos e aplicações.

Suas funcionalidades principais serão a síntese de filtros analógicos e digitais utilizando-se os métodos já descritos na literatura. Inicialmente será feita uma revisão teórica sobre os diversos tipos de filtros, seguindo-se uma discussão sobre ferramentas de desenvolvimento e metodologias de desenvolvimento; por fim, o *software* será demonstrado e serão feitas considerações sobre futuras melhorias.

Palavras-chave: Filtros Eletrônicos. Ferramenta Computacional. Processamento de Sinais.

ABSTRACT

Undergraduate Final Work
Electrical Engineer
Federal University of Santa Maria

SOFTWARE TOOL FOR ANALOG AND DIGITAL FILTER DESIGN

AUTHOR: RENAN BIRCK PINHEIRO
ADVISOR: CESAR RAMOS RODRIGUES
Defense Place and Date: Santa Maria, July 08th, 2015.

A common task in electronics and in the many areas where signal processing is used is the design of analog and digital filters, which find many different uses. However, for higher-order filter this task becomes cumbersome due to the large number of calculations required.

While there is software to design filters, in many cases it is proprietary or presents a high complexity, which then restricts its application. To provide an alternative, this work proposes an open-source software, developed in the Python programming language, for the design of analog and digital filters; thus, it can be used freely either on its own or as a base for other works and applications.

Its main functions will be the synthesis of analog and digital filters using the methods described on the literature. Initially a brief review on the diverse types of filters and their characteristics will be done, followed by a discussion on development tools and methodologies; after, the software will be demonstrated and considerations about future enhancements will be done.

Keywords: Electronic Filters, Software Tools, Signal Processing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog to Digital Converter*
BIBO	*Bounded-Input, Bounded-Output*
DAC	*Digital to Analog Converter*
DSP	*Digital Signal Processing*
FFT	*Fast Fourier Transform*
FIR	*Finite Impulse Response*
FPGA	*Field-Programmable Gate Array*
IIR	*Infinite Impulse Response*
LTI	*Linear Time-Invariant*
TDD	*Test-Driven Development*

LISTA DE SÍMBOLOS

ω	frequência angular (rad/s)
ω_c	frequência de -3 dB (rad/s)
$\mathcal{F}$	transformada de Fourier
G_p	ganho na banda de passagem (dB)
G_s	ganho na banda de parada (dB)
H	função de transferência
v_i	tensão de entrada
v_{out}	tensão de saída
f_{bw}	largura de banda do sinal
f_s	frequência de amostragem

SUMÁRIO

1 INTRODUÇÃO

Uma importante tarefa em processamento de sinais analógicos e digitais é o projeto de filtros. Ainda que nos últimos anos o aumento na capacidade de processamento dos computadores e sistemas embarcados tenha possibilitado o desenvolvimento de filtros digitais cada vez mais sofisticados e eficientes, filtros analógicos continuam tendo enorme importância; de fato, alguns autores (por exemplo, [27]) afirmam que *o mundo moderno... não existiria sem os filtros analógicos*.

Todavia, o projeto de ambos tipos de filtros é uma tarefa complexa: embora filtros de pequena ordem possam ser projetados sem dificuldade por meio de funções de transferência, este projeto torna-se mais trabalhoso à medida em que a ordem desejada aumenta. Assim, tornam-se necessárias ferramentas computacionais que tirem do projetista a necessidade de cálculos manuais e suscetíveis a erros.

Neste trabalho propõe-se uma ferramenta computacional interativa para o projeto de filtros analógicos e digitais, desenvolvida empregando-se *software* livre e aplicando-se algoritmos já descritos na literatura.

1.1 Motivação

As seguintes razões motivaram a escolha do tema e a escrita deste trabalho:

- Interesse no desenvolvimento de ferramentas em *software* livre, para redução da dependência em soluções proprietárias - muitas vezes de custo e complexidade altos, ou limitadas e sem possibilidade de melhoria;
- Interesse em aproximar a teoria (análises e cálculos teóricos vistos em sala de aula ou em livros) da prática, reduzir ou eliminar os cálculos e permitir que o usuário dedique-se ao entendimento de conceitos.

1.2 Objetivos

O objetivo desse trabalho é o desenvolvimento de uma ferramenta computacional gratuita e de código aberto para a síntese de filtros analógicos e digitais, capaz de realizar o projeto destes conforme descrito na literatura, tanto para fins didáticos quanto para aplicações profis-

sionais, sem estar atrelada a um fabricante específico. Para isso, serão seguidos os passos:

- Fazer uma breve revisão teórica de conhecimentos sobre filtros;
- Descrever a ferramenta desenvolvida, junto com as metodologias de desenvolvimento empregadas;
- Avaliar os resultados obtidos;
- Propor ideias para a continuidade do trabalho.

1.3 Estrutura do trabalho

O capítulo 2 irá realizar uma revisão teórica dos conhecimentos empregados nesse trabalho, sendo feitas considerações iniciais aplicáveis a ambos filtros analógicos e digitais e, após, sendo discutidos assuntos referentes a cada tipo de filtro.

No capítulo 3 será abordado o processo de desenvolvimento, descrevendo-se as ferramentas e metodologias usadas, seguindo-se um capítulo no qual o *software* desenvolvido será demonstrado e discutido.

Na finalização deste trabalho, serão apresentadas conclusões sobre os resultados obtidos e sugestões para futuras melhorias e continuidade deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Considerações Iniciais

Podem-se estabelecer algumas premissas para guiar o projeto de filtros:

- **Linearidade e Invariância no Tempo**: Para filtros analógicos, essa premissa é bastante clara visto que componentes eletrônicos reais não podem ter seus valores e características modificados facilmente. Dessa forma, o sistema inteiro é tratado como LTI (*Linear Time-Invariant*), facilitando a implementação do circuito e permitindo o emprego da transformada de Laplace para análise no domínio da frequência.

  Similarmente, para filtros digitais, uma das principais ferramentas de análise é a transformada Z, a qual também tem um sistema LTI como premissa.[1]

  Desta forma, a linearidade permite o uso de ferramentas matemáticas convenientes e poderosas (transformadas Z e Laplace, convolução, superposição etc...) as quais resultam na simplificação do processo de projeto [25].

- **Causalidade**: para aplicação em tempo real, um filtro deverá ser causal, isto é, depender apenas da entrada atual e das entradas anteriores. Isso requer que, para uma função de transferência qualquer dada por $Y(s) = \frac{N(s)}{D(s)}$, o grau de $N(s)$ seja menor que o grau de $D(s)$, e que ambos graus sejam finitos; isto equivale a dizer que, para qualquer $t < 0$, $y(t) = 0$ (ou, para sistemas discretos, $y_n = 0$ para $n < 0$) [25].

- **Estabilidade**: visto que os circuitos empregados em filtros ativos possuem *feedback*, o critério BIBO (*bounded in, bounded out*), o qual garante que para uma entrada finita, teremos uma saída finita (isto é, existe um valor $B > 0$ tal que $|y(t)| \leq B$ para todo t em sistemas analógicos ou $|y_n| \leq B$ para todo n em sistemas digitais), torna-se um critério importante [25]; caso contrário, podem ocorrer problemas de estabilidade com o sinal filtrado. Em uma implementação prática, teremos saturação ou oscilação na saída se esse critério for violado.

  Em filtros digitais existem considerações sobre o tamanho máximo das variáveis (uma variável de n bits[2] pode armazenar um valor máximo de $2^n - 1$ sem sinal ou uma faixa

[1] Existem filtros digitais cujo comportamento é não-linear - por exemplo, o filtro de Kalman - porém, eles não serão abordados neste trabalho.

[2] Em geral, n = 8, 16, 32 ou 64 bits, os tamanhos mais comuns de variáveis na linguagem C

de valores de -2^{n-1} a $2^{n-1} - 1$ com sinal): é necessário implementar lógica para tratar *overflows* e *underflows* [36].

- **Implementação**: para filtros analógicos, torna-se necessário definir valores de componentes disponíveis no mercado, suas características (por exemplo, a largura de banda e a *slew rate* do amplificador operacional) e suas respectivas tolerâncias de forma a obter um circuito que na prática forneça a função de transferência especificada. [18] Em altas frequências ou quando sinais de amplitude muito baixa estão envolvidos, também surgem questões ligadas ao ruído dos dispositivos empregados.

  Já para filtros digitais é necessária a escolha de um dispositivo computacional (microprocessador, microcontrolador, DSP, FPGA)[3] e de conversores analógico-digital e digital-analógico que atendam às especificações necessárias e que sejam capazes de executar a quantidade necessária de operações no intervalo entre uma amostra e outra.

  Para um sinal cuja largura de banda seja f_{bw}, o critério de Nyquist faz com que seja necessária uma frequência de amostragem f_s de no mínimo $2f_{bw}$ [15]: na prática ela será muito maior, para capturar o conteúdo do sinal até a harmônica desejada. Assim, o período que a CPU terá para realizar todas as operações (leitura do conversor A/D, cálculo, acesso à memória RAM e saída para o conversor D/A) deverá ser de no máximo $1/f_s$.

  A maioria das CPUs modernas possui conversores analógico-digital (A/D) e digital-analógico (D/A) integrados; se estes forem usados, é necessário considerar as limitações inerentes (por exemplo, número de bits e velocidade de amostragem) a esses; caso demonstrem-se insuficientes, é necessário interfaceamento de conversores externos - o que aumenta o tempo de conversão, pois se faz necessária a comunicação com os dispositivos para a leitura e a escrita de dados.

  Considerado que uma das principais aplicações de CPUs é no processamento de sinais digitais, muitos fabricantes incluem instruções de multiplicação e acumulação (*multiply and accumulate*), realizadas por *hardware* dedicado dentro da CPU, que realizam a operação $y \leftarrow y + (a * b)$, na qual y é um acumulador e a e b são variáveis, em um número mínimo de ciclos de *clock* [10].

  Surgem, também, problemas de precisão numérica que podem afetar os resultados, princi-

[3] A partir daqui, quando for mencionado *CPU*, entenda-se qualquer um desses dispositivos.

palmente para altas frequências de amostragem e na operação em ponto flutuante. Assim, torna-se necessário estruturar o algoritmo para tentar minimizar esses impactos ou, se possível - considerado que isso incorre em uma perda de precisão - utilizar ponto fixo. [36]

2.2 Tipos de filtros

Podem ser descritos cinco principais tipos de filtro, cujas respostas em frequência estão visualizadas na tabela 2.1:

- **Filtro passa-baixa** (*low-pass*): um filtro deste tipo irá rejeitar todas as frequências $f > f_c$, onde f_c é a frequência de corte estabelecida.
- **Filtro passa-alta** (*high-pass*): este filtro irá se comportar de forma oposta ao filtro passa-baixa, rejeitando todas as frequências $f < f_c$.
- **Filtro passa-faixa** (*band-pass*): é a junção dos dois tipos de filtro anteriores; para uma frequência de centro f_c e uma largura de banda (*bandwidth*) B, teremos que o filtro deixará sinais cuja frequência esteja no intervalo dado por $f_c - B$ e $f_C + B$.
- **Filtro rejeita-faixa** (*band-stop*): apresenta comportamento oposto ao filtro passa-faixa, rejeitando as frequências que estejam entre $f_L = f_c - B$ e $f_H = f_C + B$. Para B muito pequeno (ou seja, um filtro que rejeita uma faixa muito específica de frequências), é chamado de filtro *notch*.
- **Filtro passa-todas** (*all-pass*): este filtro não modifica a amplitude do sinal (isto é, a resposta em frequência dele é plana), apenas a fase dele.

2.3 Filtros Analógicos

Os filtros acima descritos são comumente chamados de *brick wall* ou filtro sinc, pois a sua representação no domínio do tempo é dada por funções da família $sinc(x) = \frac{\sin x}{x}$ [15]; são ideais e impossíveis de serem implementados na prática, por violarem a premissa de linearidade e invariância no tempo. Dessa forma, torna-se necessária uma função de transferência que consiga aproximar a resposta desejada. Comumente são empregadas cinco diferentes implementações, as quais serão descritas a seguir.

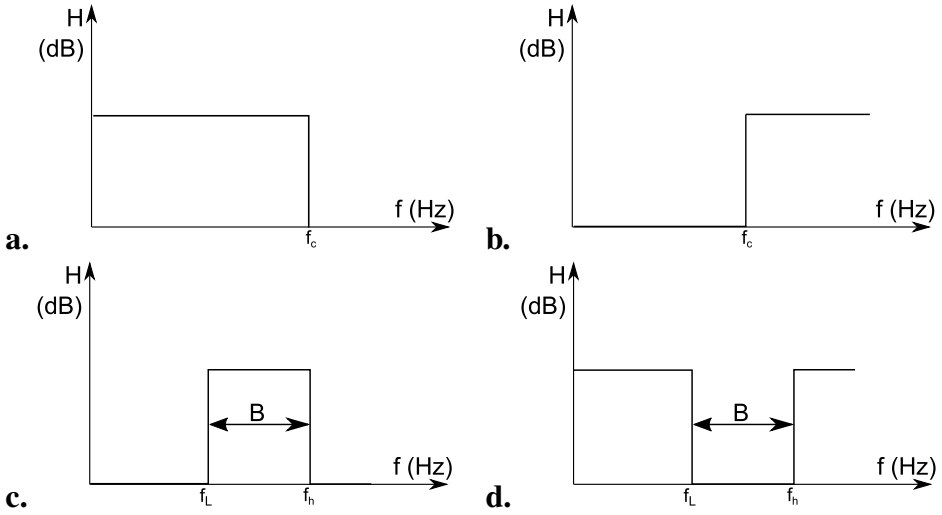

Tabela 2.1: Respostas em frequência genéricas dos filtros descritos.

2.3.1 Funções de Transferência de Filtros Analógicos

2.3.1.1 Filtro de Butterworth

A resposta em frequência de um filtro Butterworth passa-baixa, de ordem n e frequência[4] de -3 dB $\omega_c = 2\pi f_c$, para uma frequência ω é dada por

$$H(j\omega) = \sqrt{\frac{G_0}{1 + (\frac{\omega}{\omega_c})^{2n}}} \tag{2.1}$$

onde G_0 é o ganho em $\omega = 0$. Alternativamente, pode-se afirmar que os polos do filtro de Butterworth de ordem N, para $\omega = 1$, estão posicionados no plano complexo conforme a relação

$$p_k = \omega_0 \frac{\sin{(2k-1)\pi}}{2n} + j\frac{\cos{(2k-1)\pi}}{2n} \tag{2.2}$$

para $k = 1 \ldots n$.

Filtros de Butterworth, como descritos em [5], são marcados por uma resposta em frequência sem ondulações (*ripple*) na banda de passagem - assim, eles são empregados quando a característica de planicidade é desejável; todavia, sua transição da frequência de passagem (*passband*) para a frequência de corte (*stopband*) é bastante lenta em comparação aos outros filtros.

[4] As funções de transferência serão expressadas em termos de $\omega = 2\pi f_0$ pois isso simplifica sua representação

2.3.1.2 Filtro Chebyshev, tipo 1

A resposta em frequência de um filtro Chebyshev tipo 1 passa-baixa com *ripple* ϵ na banda passante é [34]:

$$H_n(\omega) = \frac{1}{\sqrt{1 + \epsilon^2 T_n^2(\frac{\omega}{\omega_c})}} \tag{2.3}$$

Onde T_n é o polinômio de Chebyshev de ordem n, definido pela relação de recorrência $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$ para $T_0(x) = 1$ e $T_1(x) = x$.

O valor de ϵ para um determinado *ripple* em dB é dado pela relação

$$\epsilon_{dB} = 20 \log \sqrt{1 + \epsilon^2} \tag{2.4}$$

Similarmente, seus polos estão distribuídos no plano complexo em uma elipse, dada pela relação

$$p_k = -\omega_0 \sin \frac{(2k-1)\pi}{2n} \sinh(\frac{1}{n} \sinh^{-1} \frac{1}{\epsilon}) + j\omega_0 \cos \frac{(2k-1)\pi}{2n} \cosh(\frac{1}{n} \sinh^{-1} \frac{1}{\epsilon}) \tag{2.5}$$

para $k = 1 \ldots n$.

Este filtro é um meio-termo entre o filtro de Butterworth e o filtro elíptico: na figura 2.1 é fácil ver que ele apresenta inclinação intermediária entre a banda de passagem e a banda de parada, não apresentando *ripple* nesta última.

2.3.1.3 Filtro Chebyshev, tipo 2

Filtros Chebyshev tipo 2 - também chamados de *Chebyshev inverso* por alguns autores [41] - apresentam *ripple* na banda de parada, em contraste ao filtro tipo 1 que apresenta este fenômeno na banda de passagem. A resposta em frequência de um filtro passa-baixa, de ordem n e frequência de -3 dB ω_c, para uma frequência ω é dado por [34]:

$$H_n(\omega) = \frac{1}{\sqrt{1 + \frac{1}{\epsilon^2 T_n^2(\frac{\omega_c}{\omega})}}} \tag{2.6}$$

sendo o parâmetro ϵ referente à atenuação na banda de parada; para uma atenuação γ em dB, ele é dado pela expressão

$$\epsilon = \frac{1}{\sqrt{10^{\gamma/10} - 1}} \tag{2.7}$$

.

Seus polos são o inverso dos polos determinados na equação 2.5, para o filtro Chebyshev tipo 1. Para este filtro, excepcionalmente, define-se ω_c como sendo a frequência onde a atenuação especificada é atingida.

2.3.1.4 Filtro de Bessel

A resposta em frequência de um filtro Bessel passa-baixa, de ordem n e frequência de -3 dB ω_c, para uma frequência ω é dada por

$$H(\omega) = \frac{\theta_n(0)}{\theta_n(\frac{\omega}{\omega_0})} \tag{2.8}$$

onde θ_n é o polinômio de Bessel, determinado pela recorrência $\theta_n(x) = (2n-1)\theta_{n-1}(x) + x^2\theta_{n-2}(x)$, de ordem n.

Os polos de um filtro Bessel de ordem n estão distribuídos no plano complexo em um círculo, espaçados em $\frac{2}{n}$, exceto nos dois polos mais próximos do eixo y, que estarão espaçados em $\frac{1}{n}$. Uma das características mais importantes desse tipo de filtro é sua resposta de fase linear.

2.3.1.5 Filtro Elíptico

A resposta em frequência de um filtro elíptico[5] passa-baixa, de ordem n e frequência de -3 dB ω_c, para uma frequência ω é dada por

$$H(\omega) = \frac{1}{1 + \epsilon^2 R_n^2(\frac{\omega}{\omega_c})} \tag{2.9}$$

onde R_n é uma função elíptica [6]. Em geral, devido ao seu cálculo mais complexo - e que não será abordado aqui devido à exigência das funções especiais, estando disponível em [27] - são usados valores tabelados.

A principal vantagem deste filtro é ter a transição da *passband* para a *stopband* mais rápida possível, ao custo de introduzir um *ripple* em ambas as bandas.

[5] ou filtro de Cauer, em homenagem a Wilhelm Cauer

[6] Funções elípticas pertencem a uma classe de funções especiais, que surgiram a partir de problemas de cálculo envolvendo elipses.

As respostas dos filtros são exemplificadas na figura 2.1. O gráfico dos polos das funções de transferência é apresentado na figura 2.2. Do gráfico podemos concluir que, quanto mais próximos os polos estiverem da origem, maior o *ripple* e mais rápida é a transição.

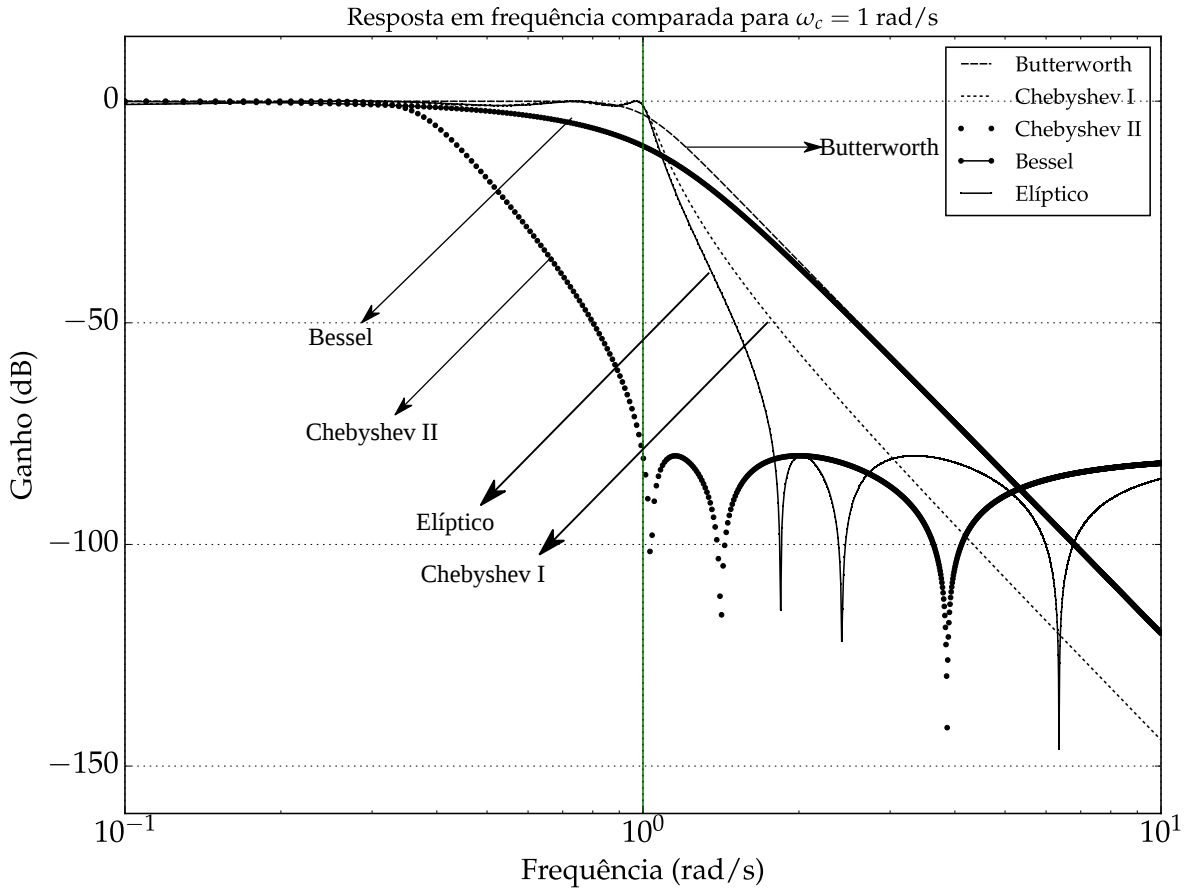

Figura 2.1: Comparação das respostas em frequência para filtros *low-pass* com ω_c = 1 rad/s. Todos os filtros têm ordem $N = 6$.

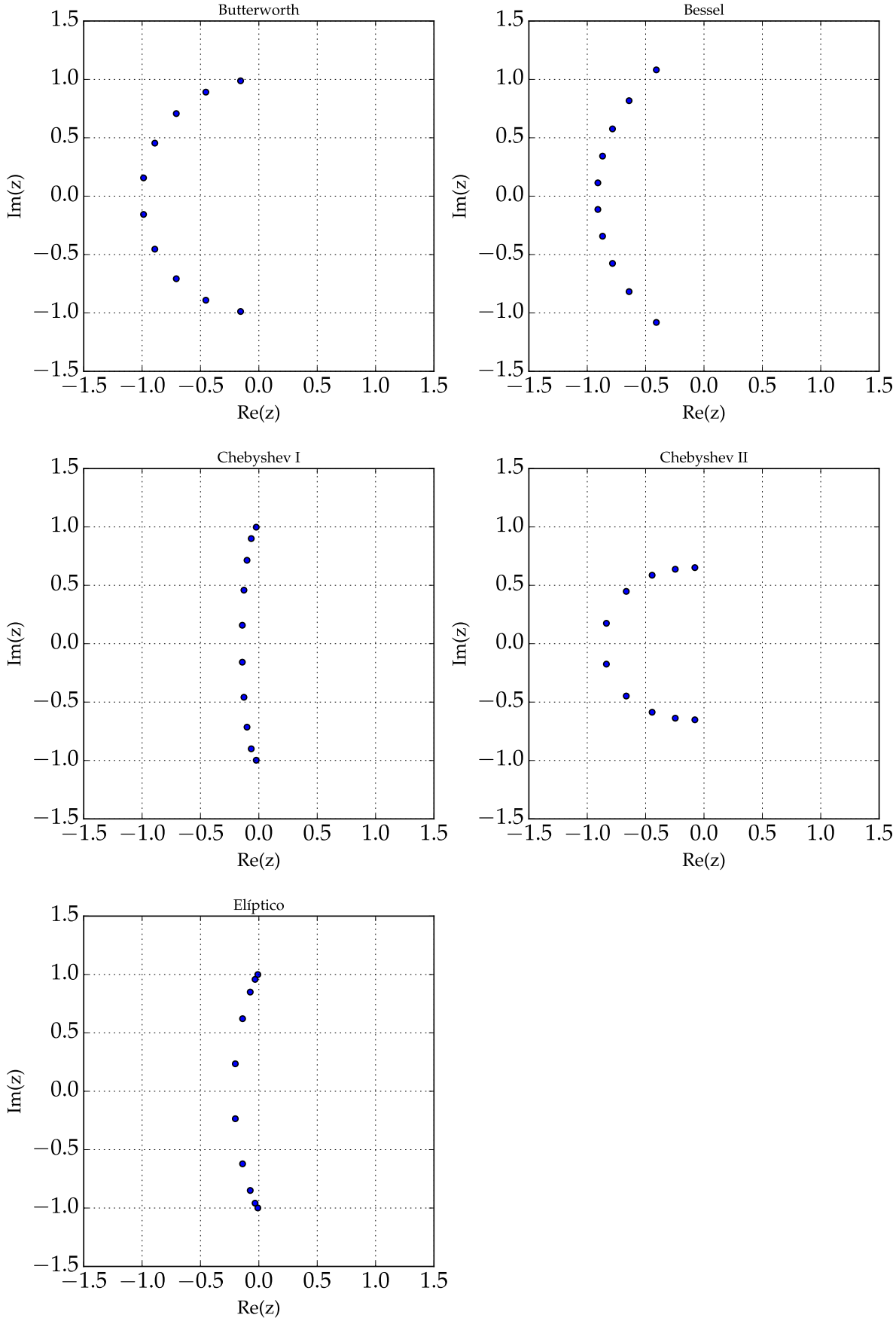

Figura 2.2: Diagramas de polos dos tipos de filtros descritos no capitulo. Todos os filtros têm ordem $N = 10$ e ω_c = 1 rad/s.

2.3.2 Transformação de Filtros

As funções de transferência acima descritas referem-se ao filtro passa-baixa. Para a obtenção das funções de transferência referentes a outros filtros a partir desta, podem-se realizar as transformações a seguir [41]:

- **Passa-baixa para passa-alta**: Multiplicar o numerador por s^n, onde n é a ordem do filtro.

- **Passa-baixa para passa-faixa**: O numerador torna-se $H_0\omega_0^2 s^n$, onde n é a ordem do filtro e H_0 é o ganho do circuito, definido pela expressão $\frac{H}{Q}$ para um valor de H especificado.

  Q, por sua vez, é a seletividade do filtro e é calculada por $\frac{F_0}{F_H - F_L}$, na qual F_0 é a frequência de ressonância do filtro e F_H e F_L são as frequências onde as respostas são de 3 e -3 dB, em relação ao pico F_0, respectivamente. Pode-se demonstrar que $F_0 = \sqrt{F_H F_L}$.

- **Passa-baixa para rejeita-faixa**: Troca-se s por $\frac{s(\omega_H - \omega_L)}{s^2 + \omega_H \omega_L}$ onde ω_H e ω_L são as frequências onde as respostas são de 3 e -3 dB respectivamente.

2.3.3 Implementação de Filtros

Obtida a função de transferência, é necessário implementá-la na forma de um circuito eletrônico ativo ou passivo[7].

Várias topologias são comumente utilizadas, implementando blocos de primeira ou segunda ordem que podem ser cascateados - respeitando-se as impedâncias de entrada e saída e adicionando *buffers* - conforme necessário.

Além de circuitos compostos por resistores e capacitores e que não serão abordados aqui pois sua análise é bastante simples, as topologias mais comuns para filtros ativos são:

[7] Filtros passivos não serão descritos neste trabalho, porém, a implementação deles pode ser feita de forma similar: determina-se um circuito que satisfaça a função de transferência desejada e então decidem-se os valores dos componentes.

2.3.3.1 Topologia *Sallen-Key*

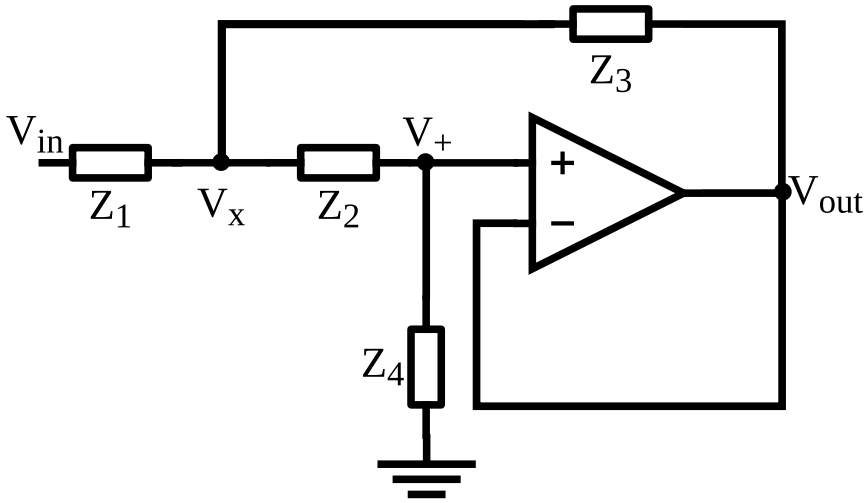

Figura 2.3: Circuito Sallen-Key genérico.

A partir da figura 2.3 podemos realizar a dedução da função de transferência da topologia Sallen-Key para elementos Z_n genéricos e assumido um *op-amp* ideal. Como o amplificador operacional está configurado com realimentação negativa,

$$v_+ = v_- = v_{out} \tag{2.10}$$

Aplicando-se a lei dos nós de Kirchoff no nó V_x tem-se que

$$\frac{V_{in} - V_x}{Z_1} = \frac{V_x - V_{out}}{Z_3} = \frac{V_x - V_+}{Z_2} \tag{2.11}$$

que pode ser reescrita, usando-se a relação descrita em 2.10, como

$$\frac{V_{in} - V_x}{Z_1} = \frac{V_x - V_{out}}{Z_3} - \frac{V_x - V_{out}}{Z_2} \tag{2.12}$$

Similarmente, fazendo-se a lei dos nós de Kirchoff na entrada não-inversora V_+ tem-se que

$$\frac{V_x - V_{out}}{Z_2} = \frac{V_{out}}{Z_4} \rightarrow V_x = V_{out}(\frac{Z_2}{Z_4} + 1) \tag{2.13}$$

Combinando-se as expressões para V_{in} e V_{out}, pode-se demonstrar que elas representam uma função de transferência da forma

$$\frac{v_{out}}{v_{in}} = \frac{Z_3 Z_4}{Z_1 Z_2 + Z_3(Z_1 + Z_2) + Z_3 Z_4} \tag{2.14}$$

que com os valores adequados dos componentes, representa uma função típica de um sistema de segunda ordem. Por exemplo, definindo-se Z_1 e Z_2 como resistores de valor R_1 e R_2 respectivamente, e Z_3 e Z_4 como capacitores de valores C_1 e C_2, a função de transferência terá a forma daquela de um filtro passa-baixa, ou seja,

$$H(s) = \frac{1}{R_1 R_2 C_1 C_2 s^2 + C_2(R_1 + R_2)s + 1} \tag{2.15}$$

O mesmo raciocínio pode ser aplicado para a obtenção das outras funções de transferência a partir da função genérica apresentada em 2.14 (por exemplo, trocando-se a posição dos resistores e capacitores, será obtido um filtro passa-alta). Uma análise detalhada desta topologia é feita pelos seus criadores em [32].

2.3.3.2 Topologia *Multiple Feedback*

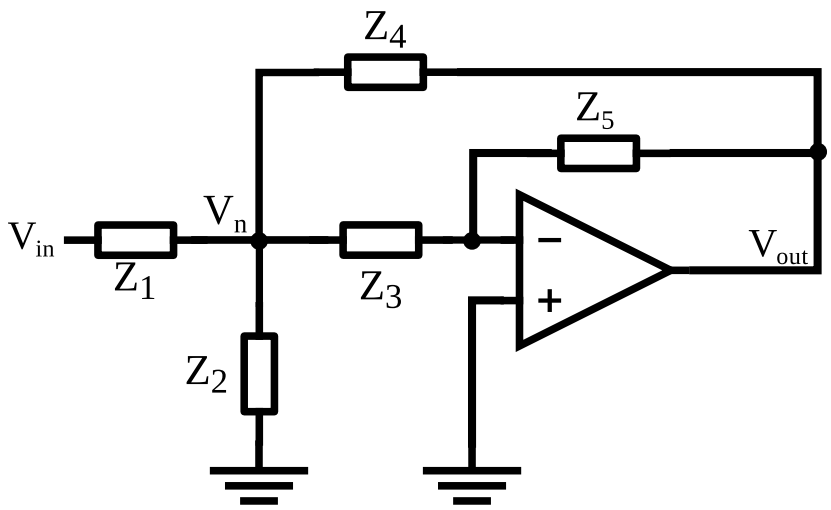

Figura 2.4: Circuito *multiple feedback* genérico.

Para simplificar a análise do circuito apresentado na figura 2.4, serão usadas as admitâncias dos componentes, de forma que $Y_n = \frac{1}{Z_n}$. Dessa forma, pela lei dos nós de Kirchoff, teremos que, onde I_n refere-se à corrente que passa pelo componente n:

$$I_1 = Y_1(V_{in} - V_n) = I_2 + I_3 + I_4 \rightarrow Y_1 V_{in} = V_n(Y_1 + Y_2 + Y_3 + Y_4) - Y_4 V_{out} \tag{2.16}$$

e também que

$$Y_3 V_n = -Y_5 V_{out} \rightarrow V_n = -\frac{Y_5 V_o}{Y_3} \tag{2.17}$$

$$Y_1 Y_3 V_{in} = V_{out}[-Y_5(Y_1 + Y_2 + Y_3 + Y_4) - Y_3 Y_4] \tag{2.18}$$

e assim pode-se demonstrar que

$$\frac{H(s)}{=} \frac{-Y_1 Y_3}{Y_5(Y_1 + Y_2 + Y_3 + Y_4) + Y_3 Y_4} \tag{2.19}$$

.

na qual fazendo-se Y_1, Y_3 e Y_4 resistores e Y_2 e Y_5 capacitores obtém-se uma função de transferência de um filtro passa-baixa, por exemplo.

2.3.3.3 Topologia *KHN (Kerwin-Huelsman-Newcomb)*

A topologia KHN (Kerwin-Huelsman-Newcomb), ou de variáveis de estado, é dada pelo circuito:

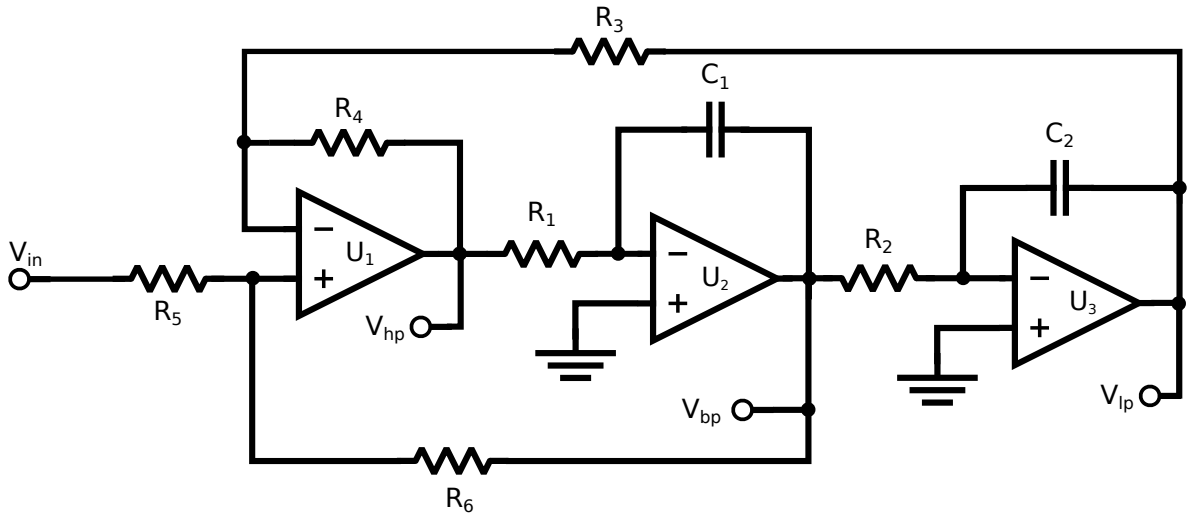

Figura 2.5: Circuito KHN

Pode-se demonstrar que a função de transferência é, para a saída passa-baixa V_{lp},

$$H_{lp}(s) = \frac{K_{lp} a_0}{s^2 + a_1 s + a_0} \tag{2.20}$$

onde $a_0 = \frac{R_4}{R_3}\frac{1}{R_1 C_1}\frac{1}{R_2 C_2}$, $a_1 = \frac{1+\frac{R_4}{R_3}}{1+\frac{R_6}{R_5}}\frac{1}{R_1 C_1}$ e $K_{lp} = \frac{1+R_3/R_4}{1+R_5/R_6}$ [31].

Similarmente, tomando-se a saída passa-alta V_{hp} na saída do somador U_1, determina-se que a função de transferência dela é

$$H_{hp}(s) = \frac{K_{hp} s^2}{s^2 + a_1 s + a_0} \tag{2.21}$$

com $K_{hp} = \frac{1+R_4/R_3}{1+R_5/R_6}$. a_0 e a_1 são os mesmos enumerados acima.

E para o filtro passa-faixa, tomado a partir da saída do integrador U_2 temos a expressão,

$$H_{bp}(s) = \frac{K_{bp} a_1 s}{s^2 + a_1 s + a_0} \tag{2.22}$$

onde $K_{bp} = \frac{R_6}{R_5}$.

Somando-se as saídas do filtro passa-baixa e passa-alta, pode-se construir o filtro rejeita-faixa.

2.3.3.4 Vantagens e Desvantagens de Cada Topologia

As topologias *Sallen-Key* e *Multiple Feedback* são bastante simples e exigem poucos componentes, o que reduz o seu consumo de energia e proporciona uma implementação mais fácil; todavia, elas são mais sensíveis às variações dos dispositivos empregados [2] [3].

Já a topologia KHN exige três op-amps que deverão ter uma largura de banda maior devido a estarem configurados como integradores (alguns autores, ex. [18], recomendam uma largura de banda 10 vezes maior que a frequência de corte do filtro); todavia, eles apresentam menor sensitividade, pois as variações dos componentes (assumindo que eles sejam relativamente similares) tenderão a se cancelar; além disso, esta topologia é insensível a variações no ganho dos amplificadores operacionais (uma análise de sensibilidade é feita detalhadamente em [19]).

Além disso, ela permite obter os três tipos de filtro mais comuns (passa-baixa, passa-alta e passa-faixa) a partir do mesmo circuito - e, com uma pequena modificação, obtém-se o filtro rejeita-faixa.

Visto que a topologia KHN é bastante comum, alguns fabricantes fornecem circuitos integrados (por exemplo, UAF42 da Texas Instruments [39]) com três *op-amps* no mesmo componente, bastando então adicionar os resistores e capacitores para as frequências desejadas.

Dessa forma, a seleção da topologia a ser empregada envolve um *trade-off* entre custo, consumo de energia, sensibilidade às variações dos componentes e complexidade de projeto.

2.4 Filtros Digitais

2.4.1 Famílias de Filtros Digitais

Existem duas famílias de filtros digitais: os filtros FIR (*Finite Impulse Response*) e IIR (*Infinite Impulse Response*). Como seus próprios nomes dizem, o fator que diferencia essas duas categorias é a resposta ao impulso (isto é, a um sinal x no qual x_0 = 1 e $x_n = 0$ para todo $n \geq 1$).

2.4.1.1 Filtros FIR (*Finite Impulse Response*)

Esta família de filtros, para um filtro de ordem N, pode ser descrita por uma equação da forma

$$y_n = b_0 x_n + b_1 x_{n-1} + \cdots + b_N x_{n-N} \equiv \sum_{i=0}^{N} b_i x_{n-i} \tag{2.23}$$

onde:

- x é o sinal de entrada;
- y é o sinal de saída;
- N é a ordem do filtro;
- b_i são os coeficientes do filtro.

Sendo constituída apenas por zeros e não tendo polos, essa família de filtros não têm problemas de estabilidade.

2.4.1.2 Filtros IIR (*Infinite Impulse Response*)

Esta família de filtros é projetada, em geral, a partir das funções de transferência dos filtros analógicos, realizando-se a transformação para o tempo discreto empregando-se, por exemplo, a transformada de Tustin que será descrita a seguir.

Ela em geral é implementável com menos cálculos, porém, requer cuidados com a sua estabilidade por possuir *feedback*.

2.4.2 Algoritmos para Projeto de Filtros Digitais

2.4.2.1 Projeto de filtros FIR: método da janela

Inicialmente, determina-se um filtro ideal que atenda à função de transferência desejada. Isso pode ser feito realizando-se a transformada discreta inversa de Fourier: em geral, o filtro obtido (chamado de h_{ideal}) será não-causal e, conforme já explicado na seção 2.3, escrito em termos da função sinc [15].

Para tornar esse filtro causal, poderíamos simplesmente truncá-lo na ordem desejada, porém, isso resulta no fenômeno de Gibbs visível na figura 2.6, comportamento fortemente oscilatório nas descontinuidades [15]:

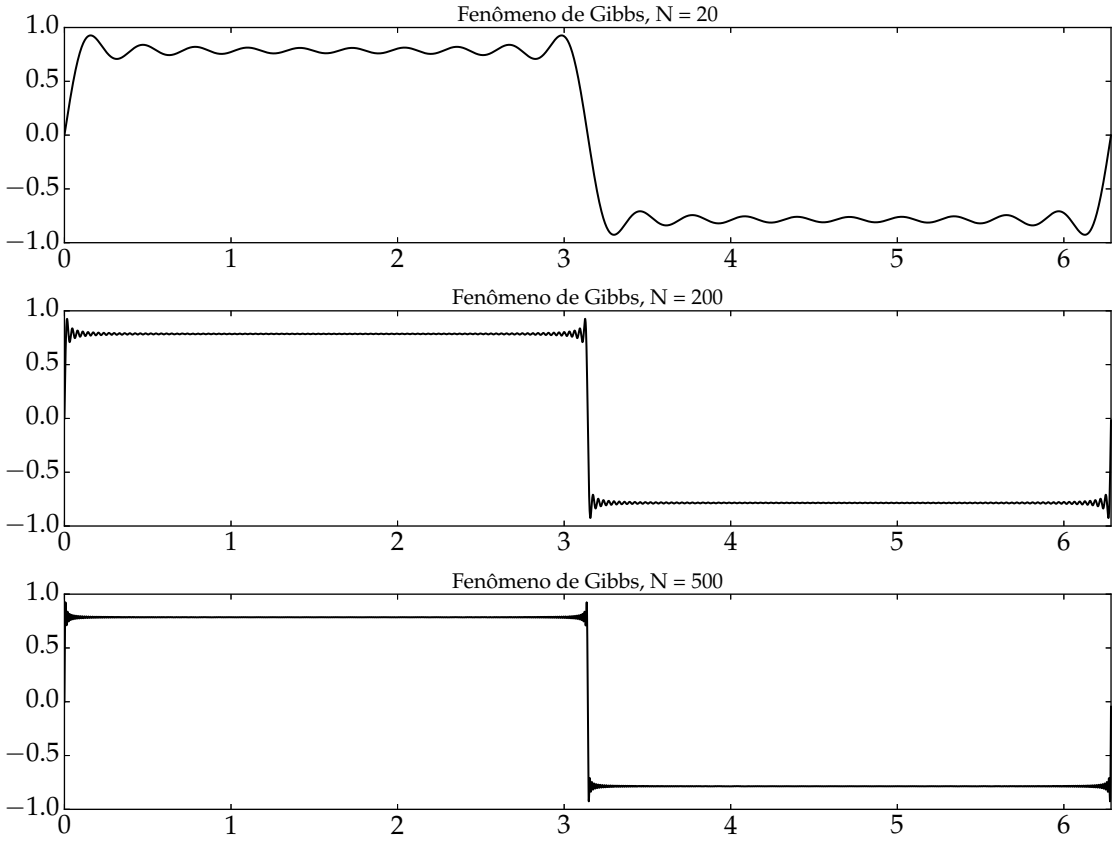

Figura 2.6: O fenômeno de Gibbs para uma soma de senoides.

Dessa forma, o filtro é transformado em causal empregando-se uma função janela (*window function*). Funções janela são funções cuja definição, para um filtro de ordem N, é dada por

$$w[n] = \begin{cases} f[n, N] & 0 \leq n \leq N \\ 0 & \text{outros casos} \end{cases}$$

Existem diversas funções janela descritas na literatura (por exemplo, [9] descreve cerca de 10 delas e [14] descreve cerca de 20 mais utilizadas), sendo que as mais comuns são:

- retangular (isto é, $w[n] = 1$ para todo $n \leq N$);
- de Hamming, dada por

$$w[n] = \begin{cases} 0.54 - 0.46 \cos \frac{2\pi n}{N} & 0 \leq n \leq N \\ 0 & \text{outros casos} \end{cases}$$

- triangular, dada por

$$w[n] = 1 - \left| \frac{n - \frac{N-1}{2}}{\frac{N}{2}} \right| \tag{2.24}$$

Feito isso, obtém-se o novo filtro $h[n]$ por meio da convolução $h_{ideal}[n] \times w[n]$. Esse filtro será causal, linear e invariante no tempo.

Não existe um critério para a seleção da função janela a ser empregada no projeto; em [14], são propostas algumas figuras de mérito para comparar estas funções (por exemplo, a amplitude dos lóbulos laterais (*side lobes*) e a largura do lóbulo principal, como visto na figura 2.7), porém a decisão será do projetista que deverá considerar todos os fatores envolvidos para o projeto do filtro (*ripple* aceitável, atenuação desejada, velocidade da transição da banda de passagem para a banda de parada etc...).

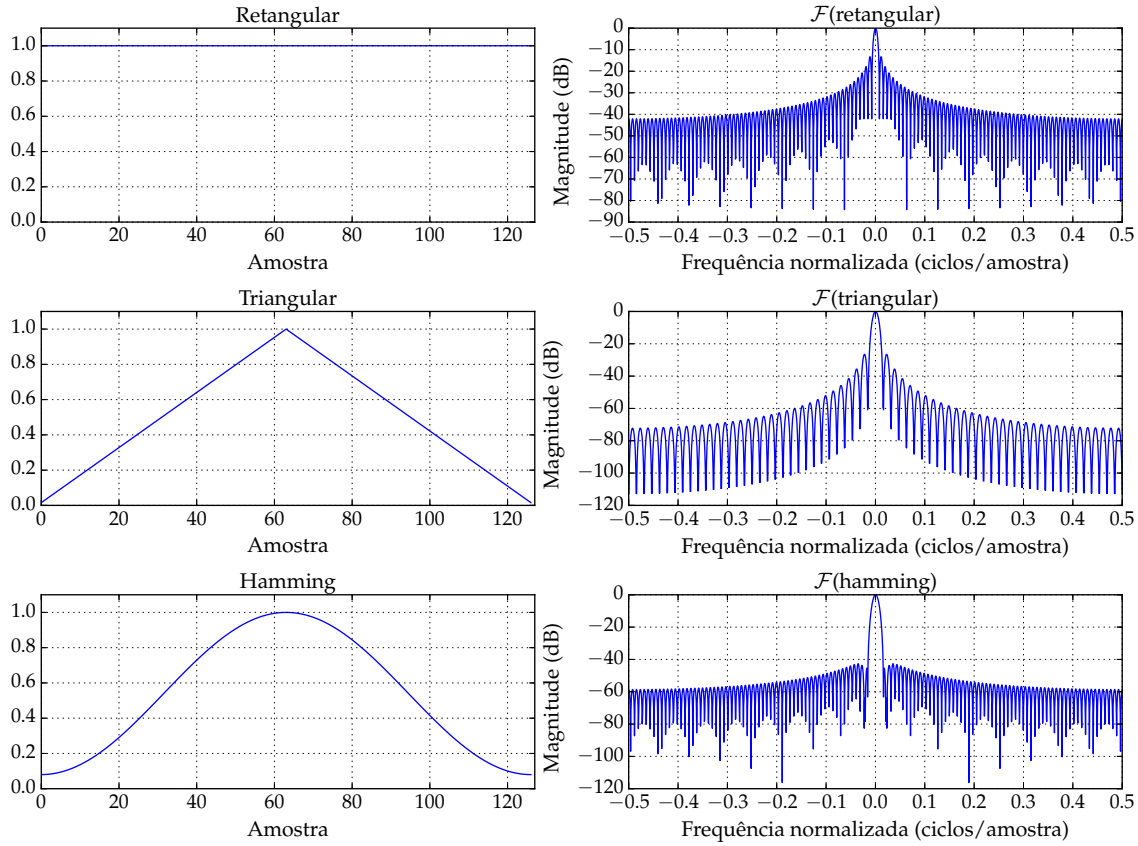

Figura 2.7: Funções janela mais comuns e sua transformada de Fourier.

2.4.2.2 Projeto de filtros FIR: algoritmo de Parks-McClellan

O método da janela, embora simples para filtros de pequena ordem e capaz de fornecer resultados *bons o suficiente* para muitas aplicações, torna-se pouco prático e já não fornece resultados ótimos à medida que a ordem cresce ou para filtros mais complexos (por exemplo, filtros que permitam a passagem de múltiplas faixas de frequências).

A estratégia adotada por [22], então, é tratar o projeto de filtros como um problema de otimização, isto é, encontrar coeficientes que aproximem a forma desejada para a resposta em frequência, ao mesmo tempo em que se minimiza o *ripple* desta aproximação. Esse algoritmo será explicado mais detalhadamente no capítulo relativo ao desenvolvimento da ferramenta.

2.4.2.3 Projeto de filtros IIR: Conversão de um Filtro Analógico em Digital

Para o projeto de filtros IIR, é comum projetar um filtro analógico equivalente e depois convertê-lo para um filtro digital, escrevendo-se sua função de transferência $H(z)$ a partir da função de transferência $H(s)$ do filtro analógico projetado.

Existem diversos métodos para tal; o método mais comum e implementado nesse trabalho é o método de Tustin ou transformada bilinear [15]. Para este método, seja $z = e^{sT}$. Reescrevendo-se essa função em termos de s, ter-se-á que $s = \frac{\ln(z)}{t}$.

A série de Taylor em função de z, truncada para o primeiro termo (suficiente para nossa aplicação) é

$$s \approx \frac{2}{t}\frac{z-1}{z+1} \tag{2.25}$$

onde t é o tempo de amostragem ($t = 1/f_s$). Um fenômeno que acontece nesta de conversão é o *warping* das frequências, o qual encontra-se descrito mais detalhadamente, por exemplo, em [15].

Para que o filtro projetado tenha a devida frequência de corte, realiza-se o procedimento conhecido como *prewarp*: sendo Ω a frequência de -3 dB desejada no filtro digital, determina-se o ω para projeto de um filtro analógico equivalente empregando-se a expressão $\omega = \tan\frac{\Omega}{2}$ [15].

2.4.3 Comparação: Filtros FIR e IIR

A vantagem imediata dos filtros FIR é sua estabilidade incondicional, por não possuírem *feedback*. Entretanto, eles requerem um maior número de cálculos (e, portanto, mais tempo de processamento e mais memória, além de causarem um *delay* na saída) para chegar no mesmo resultado que pode ser obtido com um filtro IIR.

Porém, o projeto dos filtros IIR requer menos passos e a sua implementação, em geral, possui melhor desempenho por exigir menos armazenamento em memória e menos cálculos que os filtros FIR. Torna-se mais fácil obter um filtro IIR que atinja uma especificação desejada, pois pode-se partir do conhecimento de filtros analógicos e, após, discretizar o filtro projetado.

2.5 Comparação: Filtros Digitais e Analógicos

Ainda que o custo das CPUs esteja caindo ao mesmo tempo em que seu desempenho aumenta, os filtros analógicos têm como principal vantagem o seu baixo custo para pequenas ordens: são necessários resistores, capacitores e amplificadores operacionais, ao passo que filtros digitais demandam uma CPU com o desempenho necessário para a aplicação e o desenvolvimento de algoritmos - uma atividade que, por si só, pode ser mais demorada e dispendiosa que

o projeto de um circuito.

Filtros analógicos também são obrigatórios para os circuitos de *antialiasing* de conversores A/D e D/A, ou para operação em frequências muito altas ou baixas. Porém, eles sofrem com a variabilidade dos componentes empregados para sua construção - componentes mais precisos também são mais caros; este problema não ocorre em um filtro digital no qual coeficientes e valores são armazenados na memória da CPU e se mantém estáveis.

Nos filtros digitais, a ordem máxima é limitada principalmente pelo desempenho da CPU utilizada e pelas características dos conversores de dados empregados: conforme afirmado anteriormente, a CPU tem um intervalo de tempo inversamente proporcional à frequência de amostragem para realizar os cálculos. Assim, a aplicação dos filtros digitais em altas frequências torna-se complexa e limitada - embora, com o tempo e a evolução do *hardware*, isso tenda a se tornar um problema cada vez menor.

Conclui-se, então, que os filtros analógicos são desejáveis em aplicações onde o custo e simplicidade são críticos e/ou frequências altas (a partir de de algumas centenas de KHz) ou muito baixas (na faixa de alguns poucos Hz) estão envolvidas; já os filtros digitais apresentam como importante aspecto a sua flexibilidade e a possibilidade da implementação de filtros com ordens maiores.

3 DESENVOLVIMENTO

Esse capítulo irá discorrer sobre as metodologias e procedimentos utilizados no desenvolvimento da ferramenta.

3.1 Ferramentas de desenvolvimento

3.1.1 Python

Python [28] é uma linguagem de programação de alto nível, de propósito geral e com suporte a múltiplos paradigmas de programação (orientado a objetos, imperativo, funcional), cuja sintaxe permite ao programador expressar ideias complexas em poucas linhas de código sem que ele necessite, para isso, sacrificar a clareza ou a elegância do programa. A linguagem foi escolhida devido a sua facilidade de uso em comparação com outras linguagens, nas quais a criação de interfaces gráficas ou de algoritmos complexos exige maior quantidade de códigos; partes do código que exigem maior desempenho podem chamar bibliotecas externas escritas em outra linguagem.

Sendo ela uma linguagem de código aberto, surgiu uma ampla comunidade de usuários, a qual desenvolveu diversas ferramentas para ela. Para este trabalho, as bibliotecas mais relevantes, as quais serão descritas a seguir e são todas de código aberto, são as destinadas à computação científica:

3.1.2 NumPy

NumPy é uma biblioteca que fornece funções básicas necessárias às aplicações científicas em Python. Sua principal funcionalidade é adicionar um tipo de dados versátil, juntamente com funções matemáticas, para a representação de estruturas necessárias a aplicações científicas, como matrizes e vetores [24].

Sua sintaxe foi projetada de forma a ser similar àquela do MATLAB; dessa forma, pode-se migrar código entre as linguagens com pouca ou nenhuma dificuldade.

3.1.3 SciPy

SciPy é uma biblioteca que fornece funções para integração numérica, estatística, interpolação, otimização e processamento de sinais, entre outras empregadas na resolução de problemas matemáticos.

3.1.4 SymPy

SymPy [37] é uma biblioteca cujo objetivo é a manipulação de expressões algébricas, implementando funcionalidades de um sistema algébrico computacional (*computer algebra system*) na linguagem Python.

3.1.5 PyQt

PyQt é uma biblioteca responsável pelo interfaceamento da linguagem Python com o *framework* Qt, uma biblioteca desenvolvida na linguagem C++ para a construção de interfaces gráficas. Uma das suas principais vantagens, é a quase completa abstração no que diz respeito ao sistema operacional empregado: pode-se afirmar que uma interface gráfica desenvolvida em Qt irá ter aparência bastante similar em qualquer plataforma na qual ela seja executada.

As interfaces gráficas são desenhadas de forma visual utilizando-se a ferramenta *Qt Designer*, a qual gera um arquivo que será convertido para código Python executável. Posteriormente, outro código define as ações (por exemplo, o que irá acontecer quando botões são clicados) e as conecta com os elementos visuais desenhados na interface.

3.1.6 matplotlib

matplotlib é uma biblioteca de plotagem de gráficos 2D, capaz de produzir diversos tipos de gráficos de alta qualidade, também podendo ser integrada à maioria das bibliotecas de interface gráfica. Seu uso pode ser feito em dois modos: em um deles, ela se comporta de forma similar às funções gráficas do MATLAB; no outro, empregado no presente trabalho, ela fornece uma interface orientada a objetos na qual gráficos são construídos.

3.2 Estrutura do programa

O programa foi estruturado conforme a seguinte estrutura de diretórios:

```
pyfilter ...  Diretório raiz do programa
├── engine ...  Diretório com os algoritmos
│   ├── filter.py ...  Estruturas de dados e funções comuns a ambos
│   │   tipos de filtro
│   ├── analog.py ...  Funções para projeto analógico
│   ├── digital.py ...  Funções para projeto digital
│   └── utils.py ...  Funções diversas usadas no decorrer do programa
├── gui
│   ├── canvas.py ...  Biblioteca usada para criar gráficos na interface
│   │   gráfica
│   ├── gui_analog.py ...  Interface gráfica para projeto analógico
│   └── gui_digital.py ...  Interface gráfica para projeto digital
└── tests ...  Testes (um para cada arquivo no diretório engine)
    ├── test_analog.py
    ├── test_digital.py
    └── test_utils.py
```

Dessa forma, separa-se a interface gráfica do algoritmo em si (característica desejável para facilitar o desenvolvimento: modularizando-se o código, pode-se trabalhar em uma parte dele sem que isso afete as outras).

3.3 Metodologias de desenvolvimento

3.3.1 TDD (Test Driven Development)

TDD (*Test Driven Development*, na literatura em português encontrado como *Desenvolvimento orientado para testes*) é uma metodologia de desenvolvimento de software que consiste em escrever testes de funcionalidade do código e, após, desenvolver código que satisfaça esses testes [6].

Para adicionar-se uma nova funcionalidade ou corrigir um *bug*, inicialmente escreve-se e executa-se um teste, o qual será tão específico quanto possível. Se ele executar sem erros, considera-se o desenvolvimento desta funcionalidade completo; caso contrário, o código é desenvolvido e modificado em pequenos passos (o chamado *refactoring*) até que todos os testes passem com sucesso.

Dessa forma, o desenvolvedor tem maior confiança para realizar alterações em um código: se a mudança provocar falha em algum dos testes, sabe-se exatamente qual a mudança que causou o defeito; outro aspecto que torna TDD desejável é o fato dos testes documentarem a funcionalidade do código, podendo inclusive guiar o desenvolvimento deste [1].

Na figura 3.1 é apresentado um fluxograma com os passos empregados.

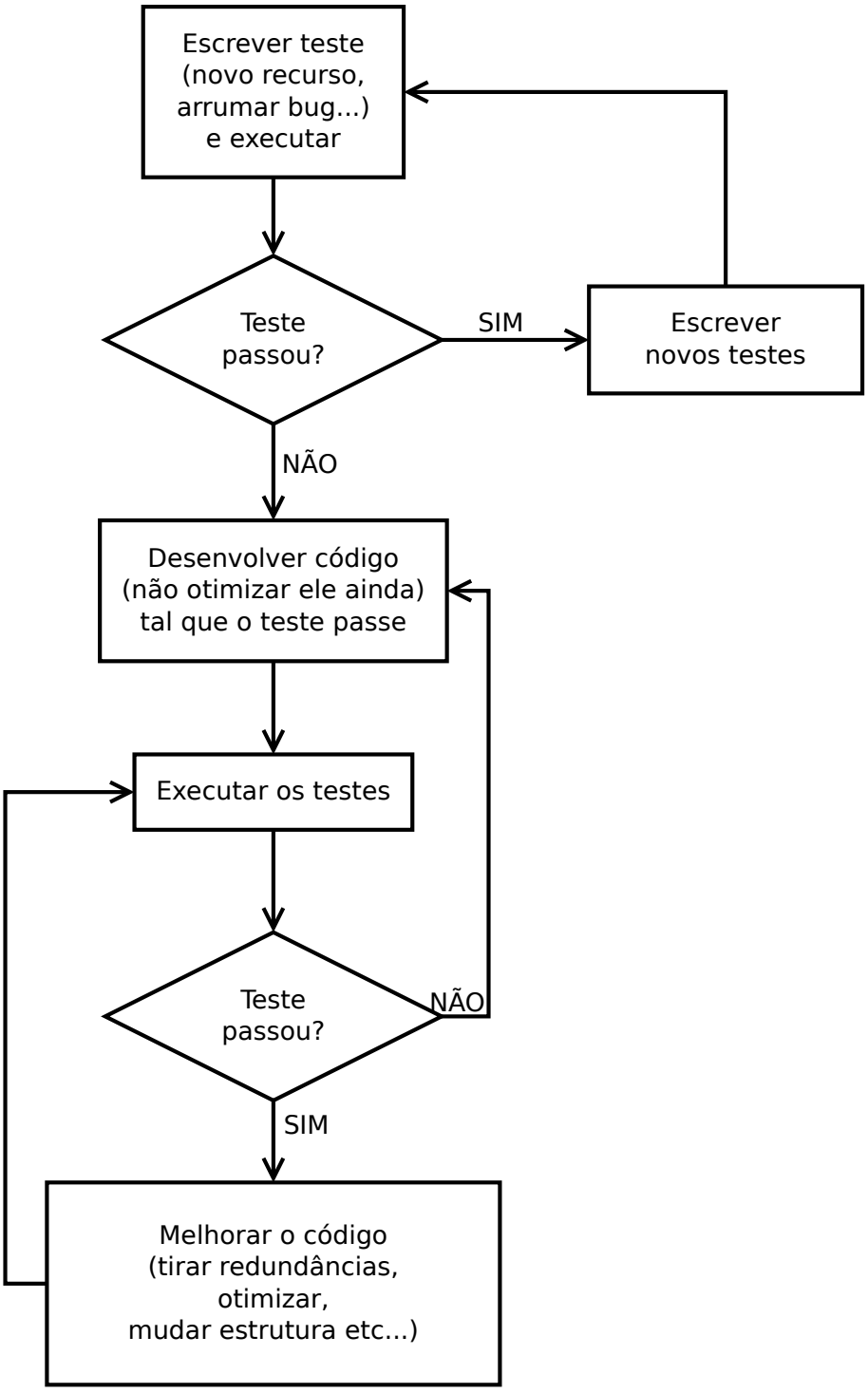

Figura 3.1: Fluxograma do desenvolvimento empregando-se TDD.

Considerou-se essa metodologia adequada para o presente trabalho pois ela fornece uma maneira rápida e eficiente de testar os algoritmos e estruturas de dados, além de obrigar o desenvolvedor a pensar em termos de atender requisitos e especificações.

Neste trabalho, testes foram desenvolvidos a partir dos resultados determinados em ferramentas similares; também foram escritos testes para validar a lógica do código (por exemplo,

para verificar se a detecção e tratamento de erros funcionavam adequadamente).

3.3.2 Controle de versão

Para gerenciar o desenvolvimento do *software*, foi empregada a ferramenta *git* para controle de versão. Dessa forma, cada modificação realizada e cada funcionalidade implementada são registradas no histórico de alterações.

Isso possibilita, entre outras conveniências, que *bugs* sejam corrigidos com maior facilidade e que se tenha um histórico do desenvolvimento.

Embora não seja o caso deste trabalho, o controle de versão torna-se uma ferramenta importante e imprescindível no trabalho em equipe: cada desenvolvedor pode ter acesso ao histórico de todas as modificações feitas no código, podendo apontar exatamente o responsável por cada linha de código.

3.4 Algoritmos

No desenvolvimento do software, constatou-se que a representação dos filtros por meio de numerador e denominador poderia causar erros numéricos, especificamente para filtros de maior ordem.

Para contornar este problema, os cálculos são realizados utilizando-se a representação ZPK (zeros/polos/ganho), isto é, são determinados vetores Z e P e um valor K tal que $G(s) = k\frac{(s-z_1)(s-z_2)...(s-z_n)}{(s-p_1)(s-p_2)...(s-p_n)}$; após, quando necessário, essa representação é convertida para uma função de transferência da forma numerador e denominador.

Para a determinação das funções de transferência dos filtros, são empregados algoritmos descritos a seguir:

- **Filtros Analógicos:** Inicialmente, é criado um filtro protótipo (isto é, um filtro com $\omega_c = 1$ rad/s) com a ordem e função de transferência desejada.

  Para determinação da ordem, são utilizadas as fórmulas clássicas descritas na fundamentação deste trabalho. Uma exceção é feita no *bandstop*: emprega-se um algoritmo de zeros de função para encontrar, a partir das especificações dadas, a menor ordem que satisfaça ambas as especificações.

  Posteriormente, o filtro é convertido para a frequência e tipo (passa-baixa, passa-alta, passa-faixa, rejeita-faixa) desejados utilizando-se as expressões descritas na fundamen-

tação teórica deste trabalho.

Os filtros de Butterworth e Chebyshev são projetados por meio do posicionamento dos seus polos no plano complexo; já o filtro de Bessel é implementado diretamente, por meio de valores tabelados. Isso limita sua ordem a $N = 25$; para ordens mais altas, seria necessário expandir tais tabelas.

O filtro elíptico não possui uma representação matemática simples como os outros filtros; ele é determinado por um algoritmo iterativo de cálculo de zeros de funções, que encontra os zeros da função elíptica que atendem às especificações desejadas. Este procedimento de cálculo está descrito, juntamente com sua derivação, em [26].

- **Filtros Digitais FIR:** são implementados os dois métodos vistos na revisão teórica, janelamento e Parks-McClellan.

  Para o método de janelamento, as rotinas de convolução são implementadas na linguagem C para melhor desempenho - uma otimização adequada, considerando que esta basicamente envolve um grande número de somas e multiplicações, e conforme citado na introdução os processadores atuais têm instruções dedicadas para isso - e o código em Python chama essas funções quando necessário.

  Constrói-se o vetor com os valores da função janela escolhida pelo usuário (uma lista completa está disponível na documentação da biblioteca SciPy); então, é feita a convolução deste com a função de transferência projetada.

  O algoritmo de Parks-McClellan, por sua vez, é uma implementação do algoritmo de Remez para aproximação de funções por meio da teoria de aproximações de Chebyshev. Podemos descrever a função erro E entre o filtro ideal D e o filtro real H, a qual será nosso objetivo minimizar, por

$$E(\omega_i) = W(\omega_i)[D(e^{j\omega_i}) - H(e^{j\omega_i})] = (-1)^{i+1}\delta \tag{3.1}$$

  onde a função H, que corresponde ao filtro de ordem N que queremos implementar, é dada pela forma

$$H(e^{j\omega}) = \sum_{k=0}^{\frac{N-1}{2}} a_k(\cos\omega)^k \tag{3.2}$$

Então, precisamos encontrar coeficientes a_k para H que minimizem o erro. A partir das frequências especificadas e de outras estabelecidas (aleatoriamente ou calculadas nos intervalos das especificadas), executa-se um método iterativo que tenta minimizar o maior erro encontrado (o chamado método minimax).

Neste caso, o erro aparecerá na forma de *ripple*; pode-se afirmar que os filtros obtidos são *equirriple*, isto é, apresentarão uma oscilação entre $-\delta$ e δ nas bandas passante e de parada.

Dessa forma, pode-se afirmar que o algoritmo de Parks-McClellan gera um filtro ótimo no sentido de minimização do *ripple*, ao custo das outras características (por exemplo, nem sempre se atingem os ganhos especificados).

A dedução deste algoritmo, junto com sua análise, encontra-se disponível na literatura, por exemplo, em [23]; uma discussão mais extensa encontra-se em [35].

- **Filtros Digitais IIR:** inicialmente é construído um filtro analógico (já com o *prewarp* adequado aplicado) da topologia desejada e, após, este é discretizado empregando-se Tustin. As frequências são normalizadas na escala $0 \ldots 1$, de tal forma que sua faixa vá de 0 até $f_s/2$.

4 RESULTADOS

Nesse capítulo a ferramenta será demonstrada, realizando-se projetos de exemplo a fim de demonstrar seu funcionamento e permitir a realização de comparações com as ferramentas já existentes.

4.1 A Ferramenta

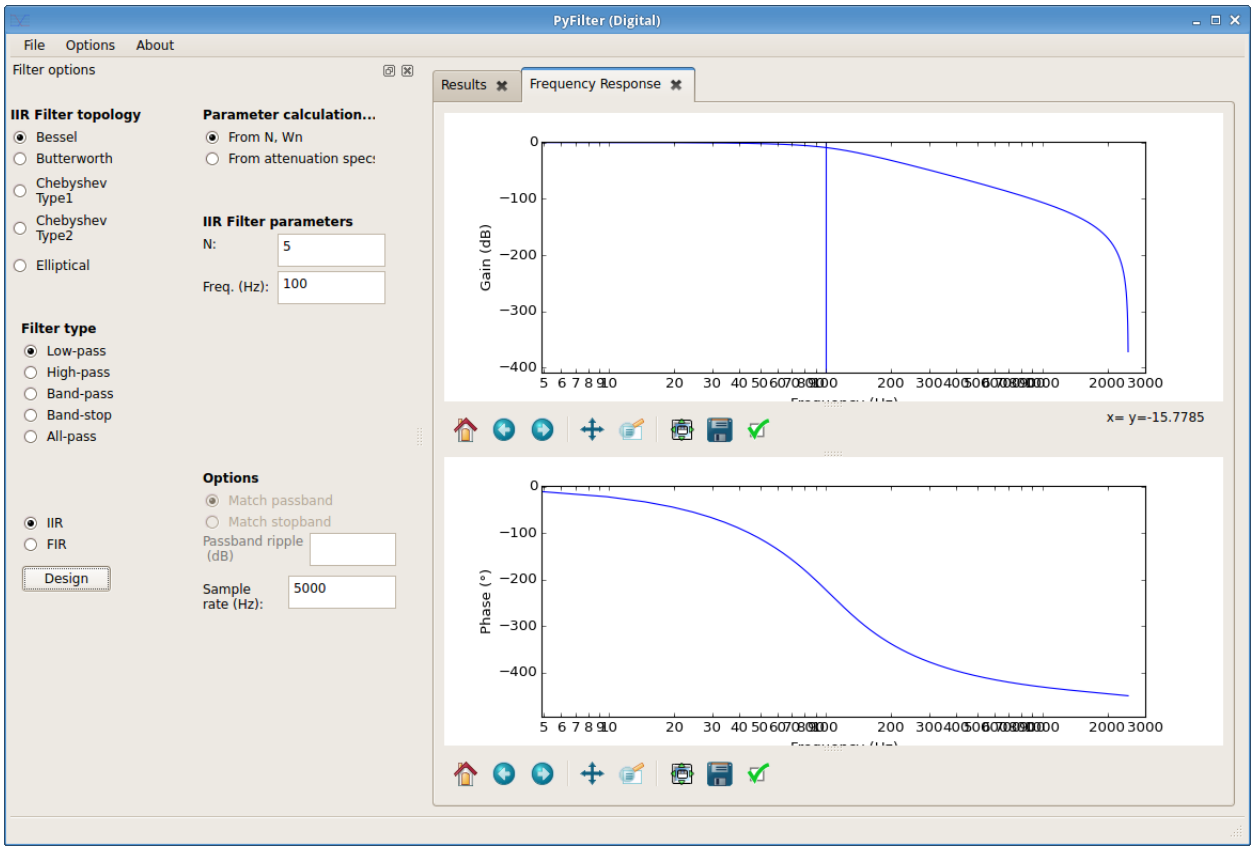

Figura 4.1: Interface gráfica da ferramenta, para projeto de filtros digitais.

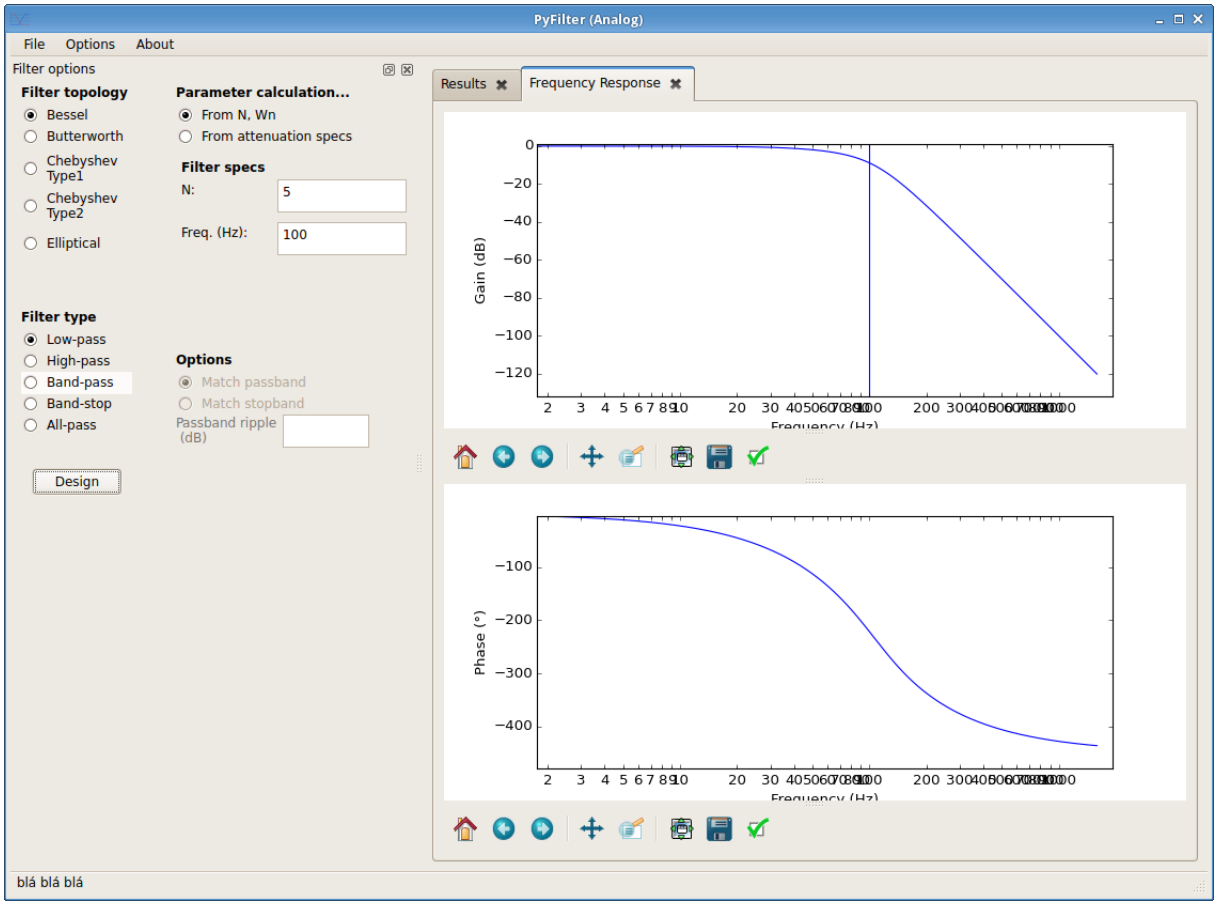

Figura 4.2: Interface gráfica da ferramenta, para projeto de filtros analógicos.

Nas figuras 4.1 e 4.2, apresenta-se a interface gráfica da ferramenta desenvolvida. Seu uso é bastante direto: o usuário irá selecionar e entrar com os parâmetros do filtro desejado, e após selecionada a opção de projetar o filtro, será apresentado um relatório com a função de transferência e os coeficientes, além das respostas de magnitude e fase para o filtro projetado.

4.2 Exemplo de projeto: filtro passa-baixa

Em [33] temos o exemplo de projeto de um filtro passa-baixa para aplicação biomédica, na qual é desejada uma frequência de corte muito baixa (8 mHz). A realização deste projeto demonstra-se uma tarefa desafiadora devido aos valores dos componentes envolvidos; no mesmo artigo encontram-se propostas de topologias de circuitos para solução desses problemas, porém que já fogem ao objetivo do presente trabalho.

4.2.1 Projeto Analógico

No artigo original, é implementado um filtro Butterworth de segunda ordem devido à característica de planicidade desejável à aplicação deste; aqui, foram implementados filtros de quarta ordem em todas as topologias expostas na introdução. Dessa forma, obtém-se as seguintes curvas para os filtros projetados:

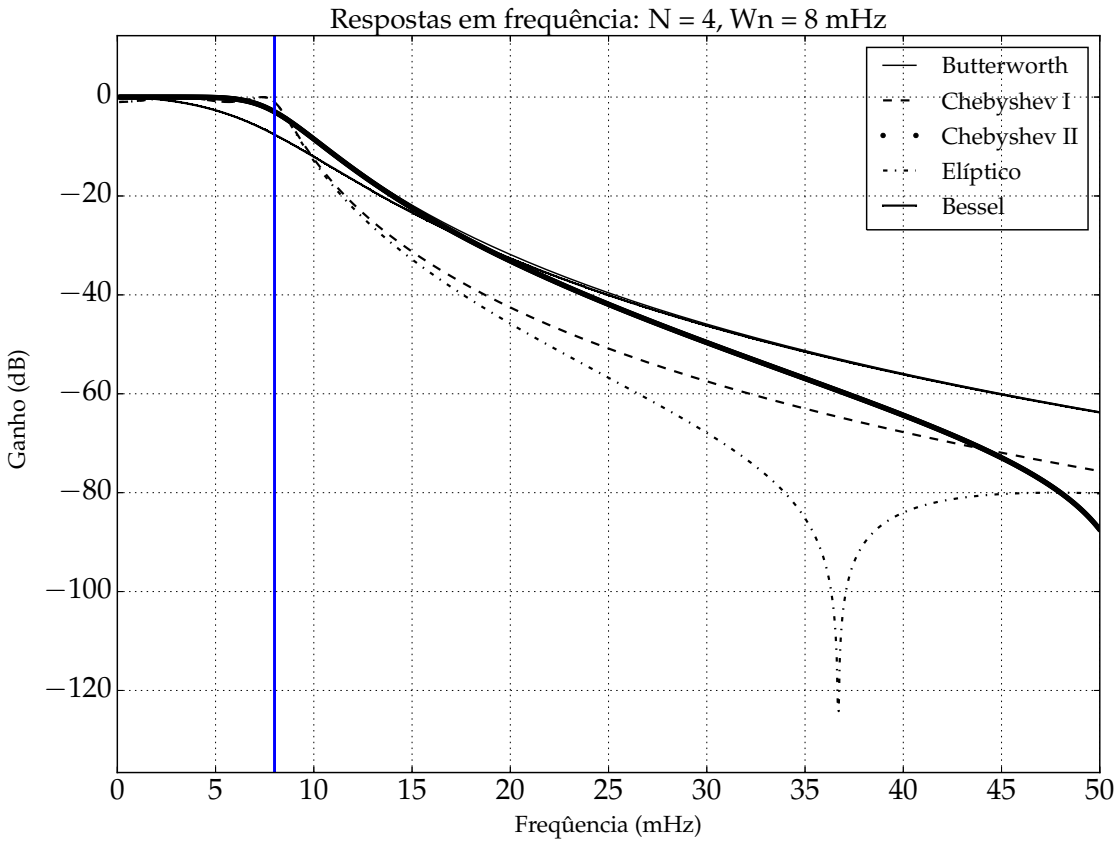

Figura 4.3: Resposta em frequência para filtro low-pass analógico.

A partir do resultado obtido, é possível verificar aquilo que foi afirmado na fundamentação teórica deste trabalho: filtros Butterworth e Bessel tem uma maior planicidade na banda de passagem, ao passo que os filtros Chebyshev apresentam uma transição mais rápida ao custo de terem maior *ripple*. Constata-se também que, para uma frequência $f \gg f_c$, o filtro de Bessel aproxima o filtro de Butterworth.

4.2.2 Projeto Digital: IIR

Inicialmente, foi tentada uma taxa de amostragem de 1000 Hz, todavia constatou-se que esta era inadequada para a aplicação, pois os coeficientes dos filtros efetivamente tornavam-se zero. Então, estabeleceu-se para este projeto que a taxa de amostragem será de 5 Hz.

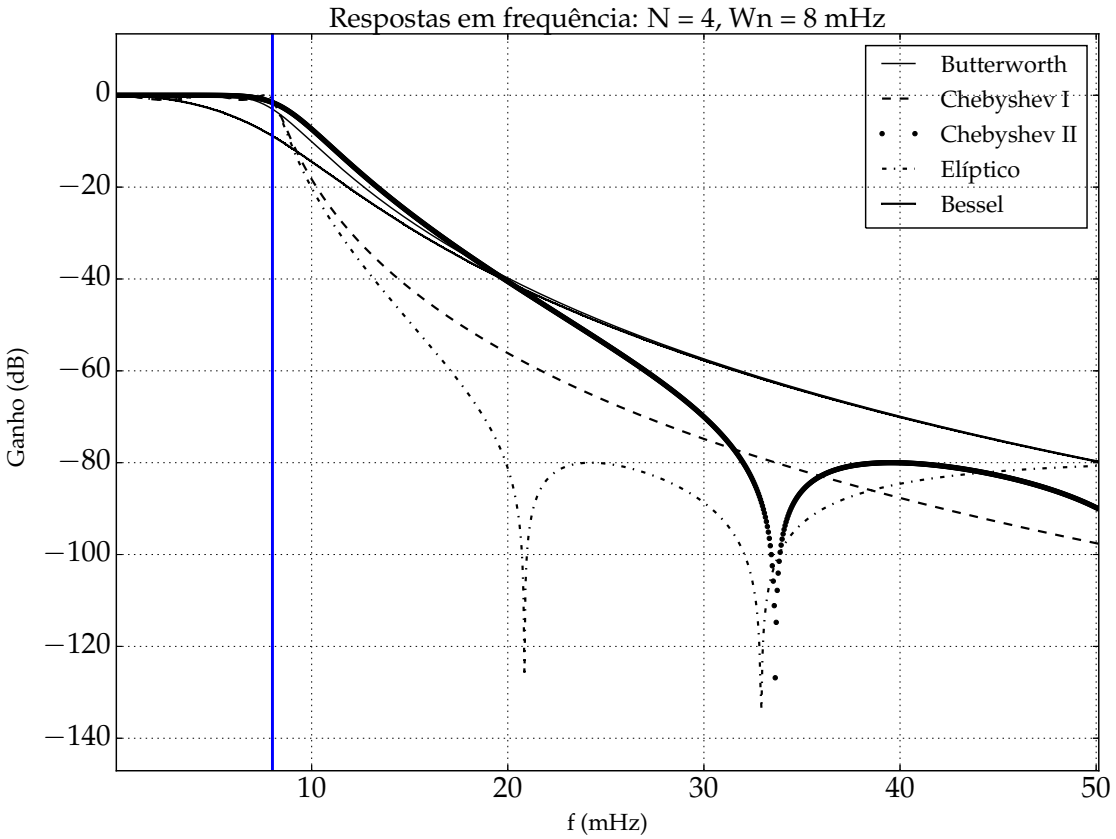

Figura 4.4: Resposta em frequência para filtro low-pass IIR.

Constata-se que os resultados obtidos são muito próximos dos resultados do filtro analógico.

4.2.3 Projeto Digital: FIR

Como os filtros FIR são projetados de forma a aproximarem uma resposta em frequência ideal, especificou-se um filtro ideal passa-baixa com frequência de transição de 8 mHz; na banda de parada, definiu-se uma atenuação de 80 dB. Obteve-se:

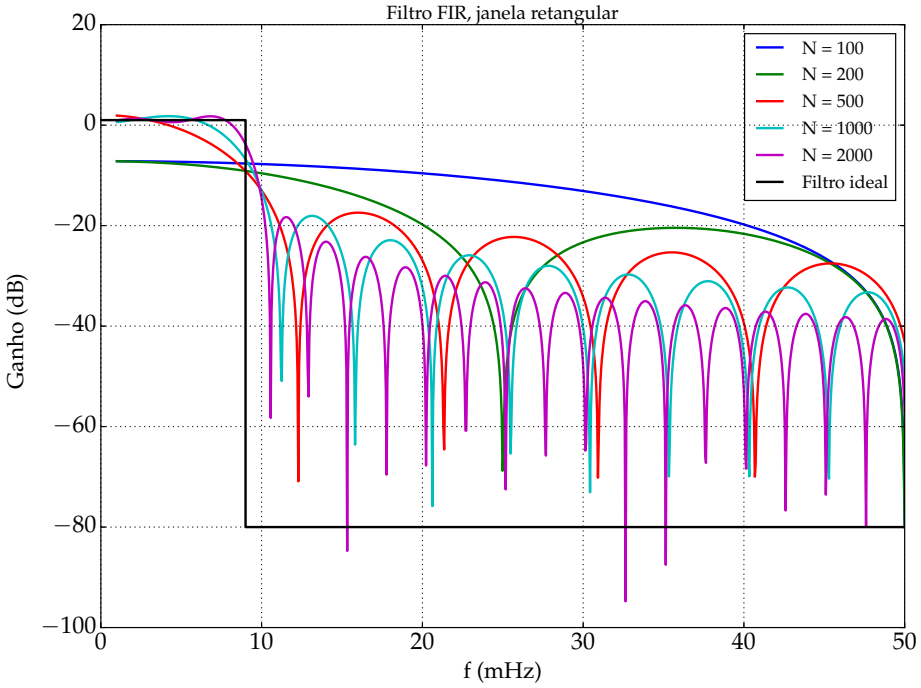

Figura 4.5: Resposta em frequência para filtro low-pass FIR com janela retangular.

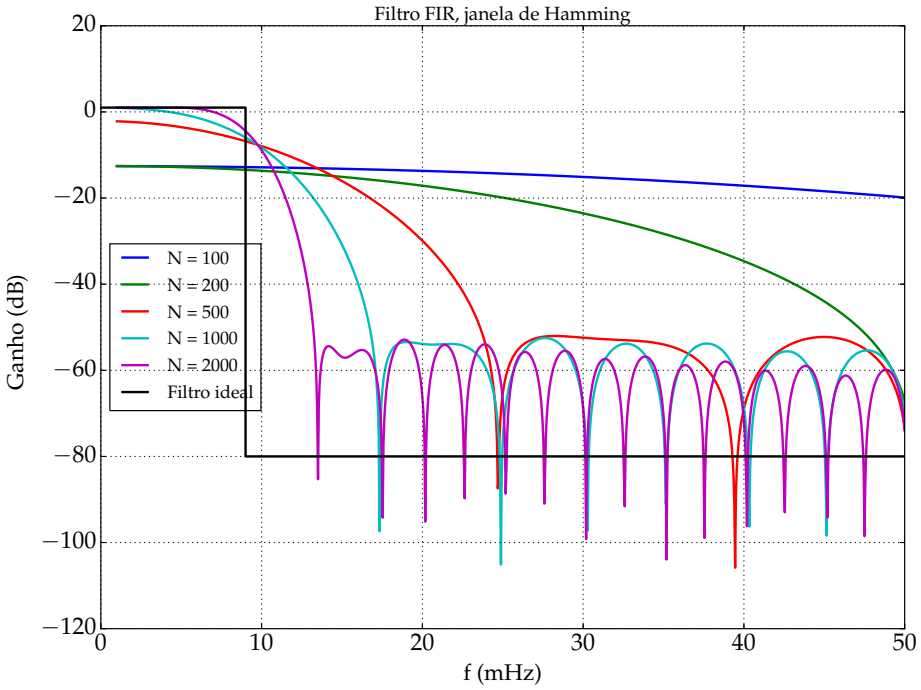

Figura 4.6: Resposta em frequência para filtro low-pass FIR com janela de Hamming.

Verifica-se que, para ordens muito baixas, o filtro é incapaz de atender os requisitos

especificados; já se esta for aumentada, os filtros apresentam maior proximidade daquele especificado, ao custo de maior *ripple* na banda de parada.

A partir dos resultados apresentados nesta seção, pode-se afirmar que filtros digitais IIR são uma ferramenta possível para a filtragem de sinais de baixa frequência, devendo porém a taxa de amostragem ser compatível com a frequência dos sinais a serem processados.

4.3 Exemplo de projeto: filtro *notch*

Uma aplicação bastante comum do projeto de filtros é o projeto de filtros *notch* para remoção do ruído de 60 Hz da rede elétrica de um sinal. O filtro a ser projetado será um *band-stop* cujas frequências de -3 dB são 59 e 61 Hz.

Para o projeto dos filtros analógico e IIR definiu-se que o *ripple* máximo aceitável é de 1 dB e que a atenuação na banda de parada será de 80 dB.

4.3.1 Projeto Analógico

Na figura 4.7 apresentam-se as respostas em frequência de um filtro *band-stop* de 5ª ordem, nas frequências de corte especificadas anteriormente, para diferentes funções de transferência:

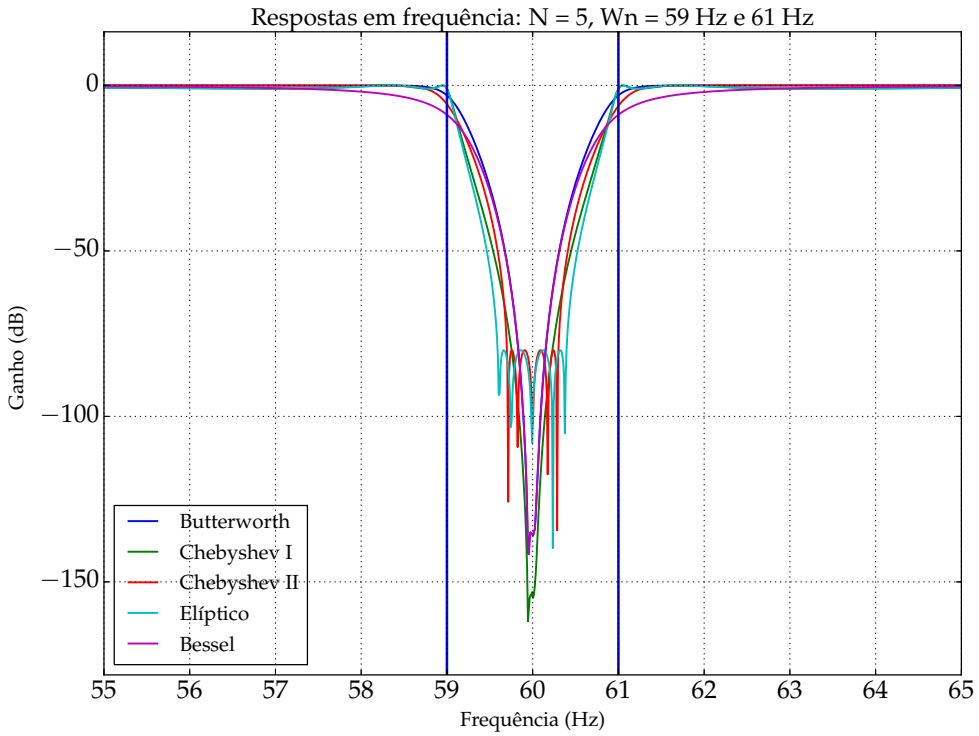

Figura 4.7: Resposta em frequência para filtro band-stop analógico.

Verifica-se o comportamento descrito na revisão teórica do presente trabalho, sendo que no filtro Chebyshev tipo II, a(s) frequência(s) ω_n representam os pontos nos quais é atingida a atenuação na banda de parada (ao contrário dos outros filtros, onde elas representam as frequências de -3 dB).

4.3.2 Projeto Digital: IIR

Projetou-se um filtro com especificações similares ao do analógico apresentado anteriormente, sendo a frequência de amostragem definida em 1000 Hz. Obteve-se a seguinte resposta em frequência:

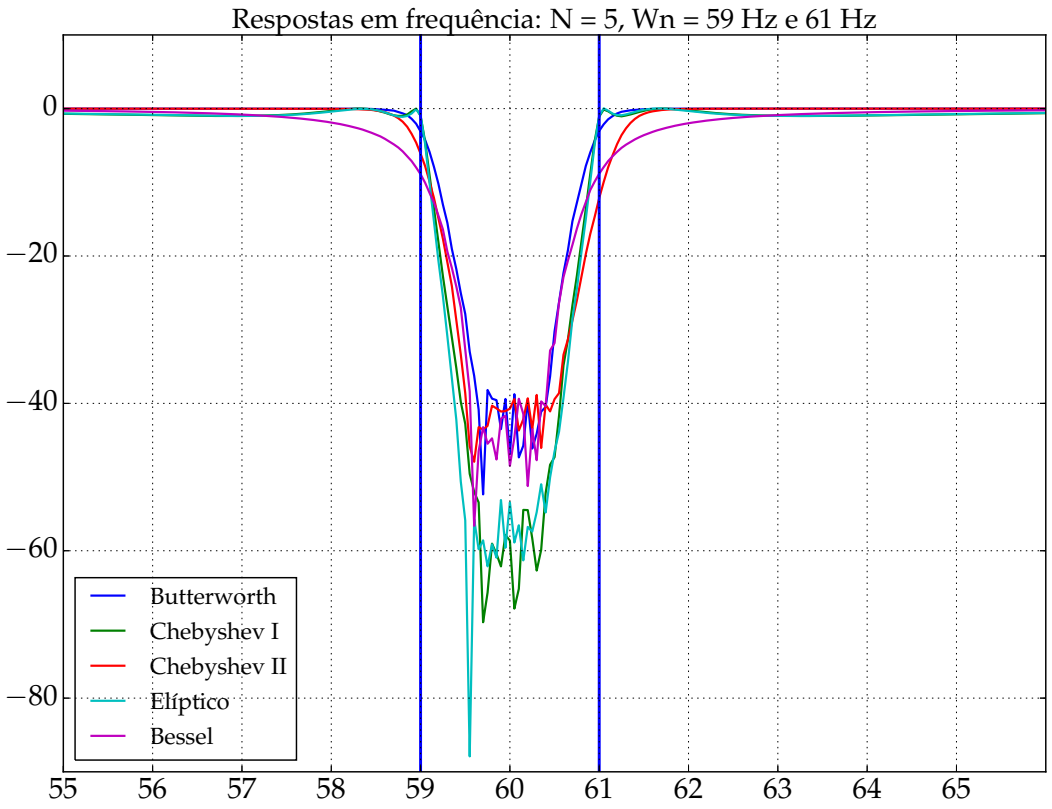

Figura 4.8: Resposta em frequência para filtro band-stop IIR.

As mesmas conclusões feitas no item anterior para os filtros analógicos podem ser feitas aqui; todavia, constata-se a presença de pequenas e praticamente desprezíveis oscilações devido ao processo de discretização.

4.3.3 Projeto Digital: FIR

O projeto deste filtro é especificado em função de ganhos e bandas; dessa forma, define-se:

- **Banda de passagem**: 0-58 Hz e 61-500 Hz, com ganho de 0 dB;
- **Banda de parada**: 59-61 Hz, com atenuação de 80 dB.

Foram obtidos os seguintes resultados para diferentes números N de *taps*[8], quando empregada a função janela retangular:

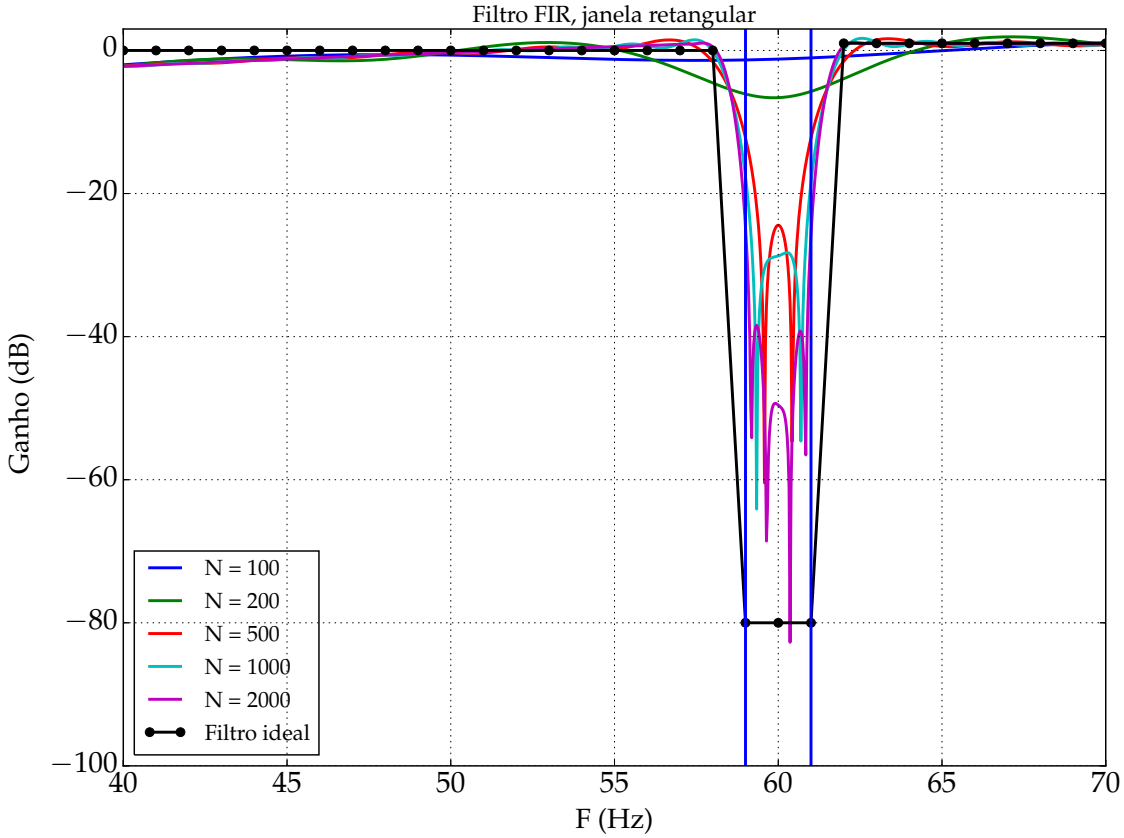

Figura 4.9: Resposta em frequência para filtro band-stop FIR com a função janela retangular.

[8] Não há uma regra geral para estimar a ordem necessária; desta forma, o processo muitas vezes envolve a simulação com diferentes ordens até atingida a resposta desejada

Quando é empregada a função janela de Hamming, tem-se o seguinte resultado:

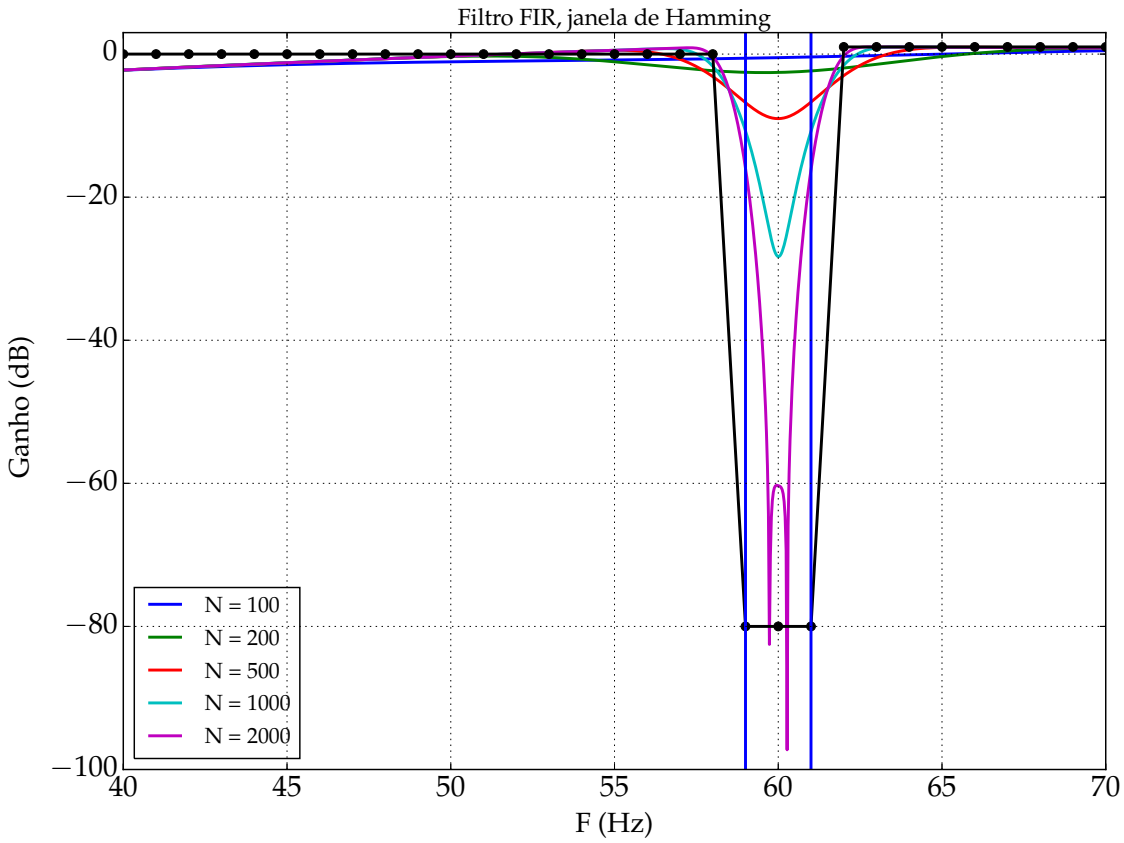

Figura 4.10: Resposta em frequência para filtro band-stop FIR com a função janela de Hamming.

Já para a janela triangular, tem-se:

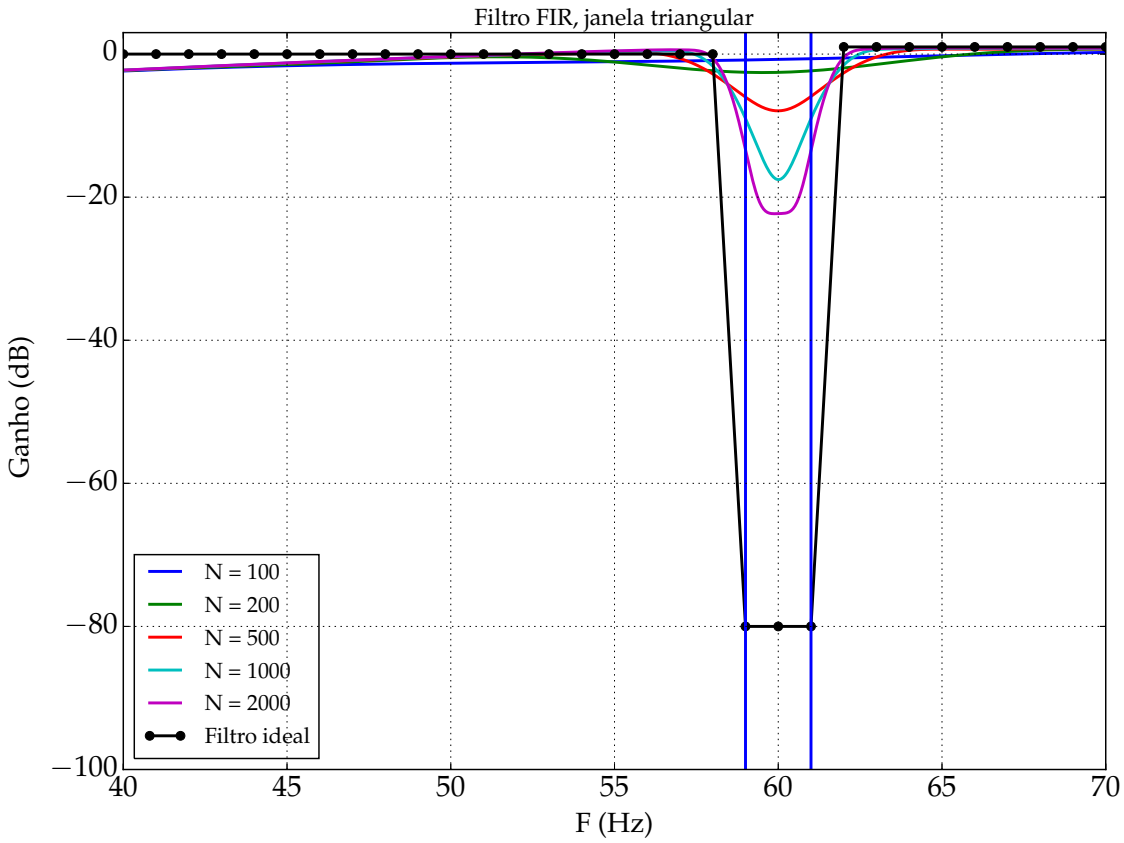

Figura 4.11: Resposta em frequência para filtro band-stop FIR com a função janela triangular.

De imediato constata-se que, para atingir o mesmo desempenho dos filtros IIR, é necessário um número maior de iterações. Verifica-se que as funções janela de Hamming e triangular não tem o *ripple* na banda de parada, ao custo de apresentarem uma menor atenuação - e também pode ser visto que a função janela triangular é inadequada para a aplicação proposta. A função janela retangular, também, atinge um resultado mais próximo dos filtros IIR elíptico e Chebyshev tipo II.

Para o filtro *equirripple* sintetizado pelo algoritmo de Parks-McClellan, é obtido:

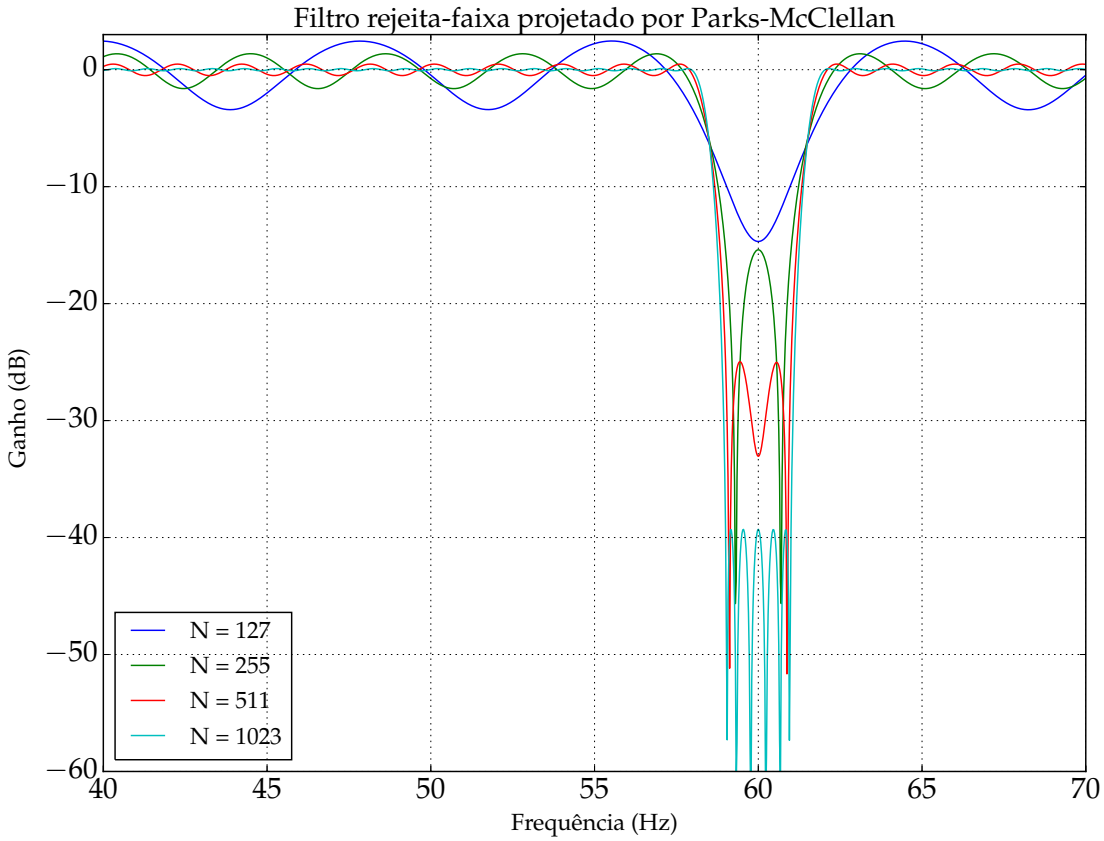

Figura 4.12: Resposta em frequência para filtro band-stop FIR projetado pelo algoritmo de Parks-McClellan.

Para o filtro implementado por esse algoritmo, tem-se uma imposição do algoritmo que requer que filtros *bandstop* tenham ordem ímpar, caso contrário os objetivos não são atingidos.

Pode-se verificar que esta família de filtros apresenta o *ripple* igual para todas as frequências na banda de passagem, sendo que este é inversamente proporcional à ordem do filtro ao custo de oscilações na banda de parada.

4.4 Exemplo de projeto: filtro passa-banda

Em aplicações de áudio e de telefonia é bastante comum a necessidade de filtros para permitir a passagem da voz humana, na faixa de 300 a 4000 Hz. Para este projeto tem-se as seguintes especificações:

- **Bandas de parada**: 100-300 Hz e 4000-6000 Hz.
- **Banda de passagem**: 300 a 4000 Hz
- **Ripple na banda de passagem**: 3 dB
- **Atenuação na banda de parada**: 80 dB

e devem-se determinar ordem e frequências críticas que as satisfaçam.

4.4.1 Projeto Analógico

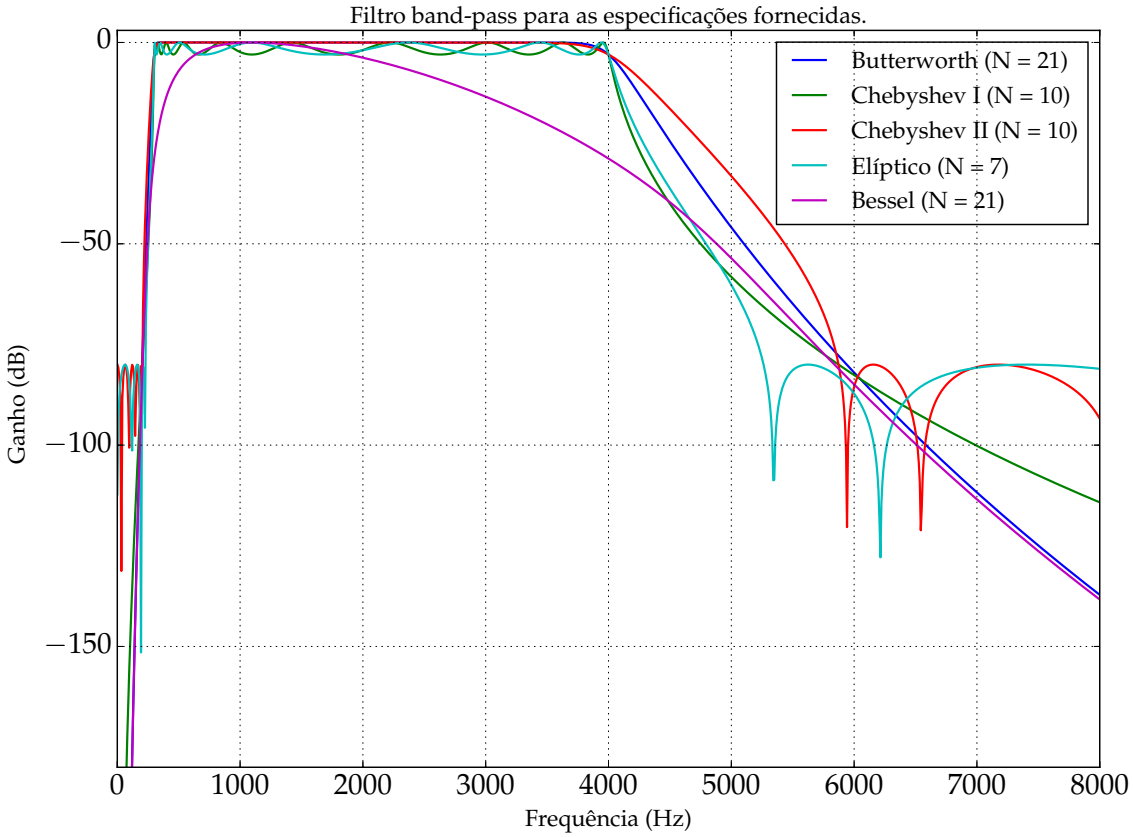

Figura 4.13: Resposta em frequência para filtro band-pass analógico.

A partir desses resultados, os *trade-offs* entre os diferentes tipos de filtros ficam claros. Verifica-se que o filtro de Butterworth, embora não apresente *ripple*, precisa de maior ordem para atingir resultados similares aos outros filtros; já o filtro de Bessel demonstrou-se insuficiente para atingir a especificação solicitada. O filtro elíptico consegue fornecer, para uma menor ordem, resultados bastante similares às outras topologias.

Um dos problemas já citados anteriormente e que foi constatado nesse exemplo de projeto foram instabilidades numéricas: inicialmente foi tentado o projeto com uma especificação de banda de parada de 4000 a 5000 Hz, porém os algoritmos falhavam ao calcular a resposta em frequência. O MATLAB tenta contornar esse problema calculando o filtro em seções de segunda ordem (*second order sections*), as quais não foram implementadas em Python, ficando em aberto a ideia de sua implementação.

4.4.2 Projeto Digital: IIR

Para este projeto, foi estabelecida uma taxa de amostragem de 20 KHz.

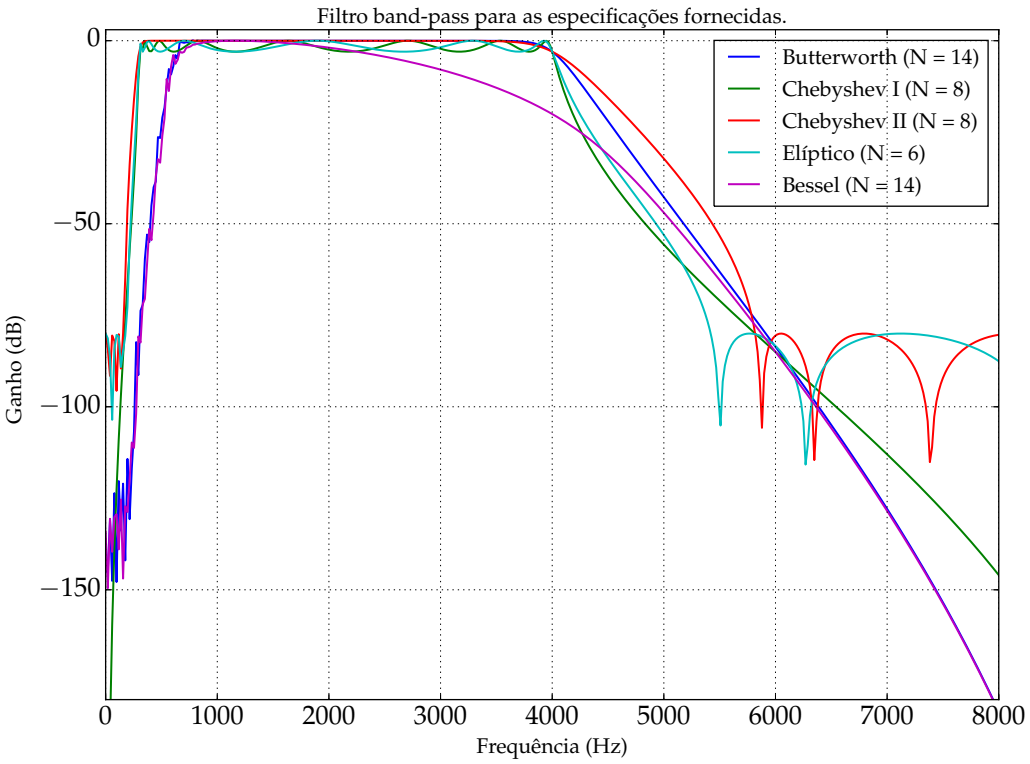

Figura 4.14: Resposta em frequência para filtro band-pass IIR.

Os resultados de ambos projetos são bem similares, valendo as mesmas conclusões que

foram feitas para o projeto analógico.

4.4.3 Projeto Digital: FIR

Foi especificado um filtro ideal, com 0 dB de ganho na faixa de 300 a 4000 Hz e -80 dB fora dela, tendo-se obtido:

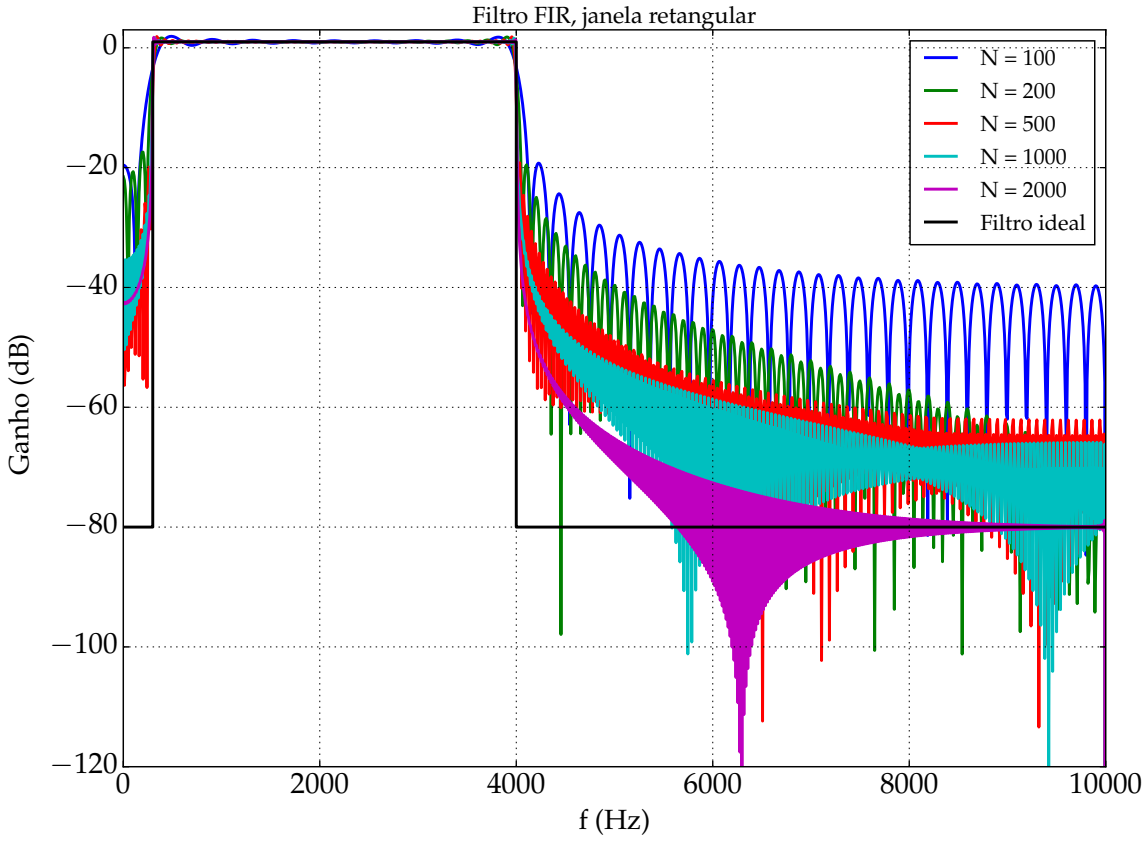

Figura 4.15: Resposta em frequência para filtro band-pass FIR com a função janela retangular.

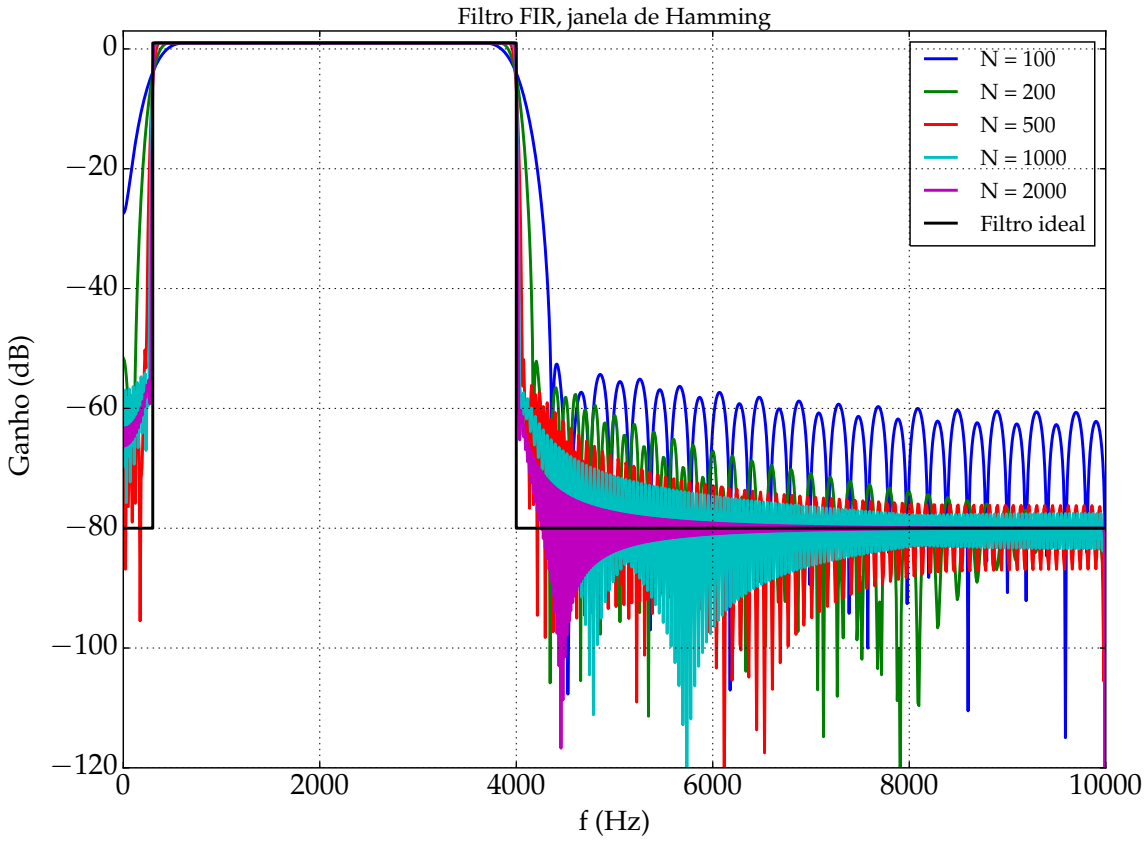

Figura 4.16: Resposta em frequência para filtro band-pass FIR com a função janela de Hamming.

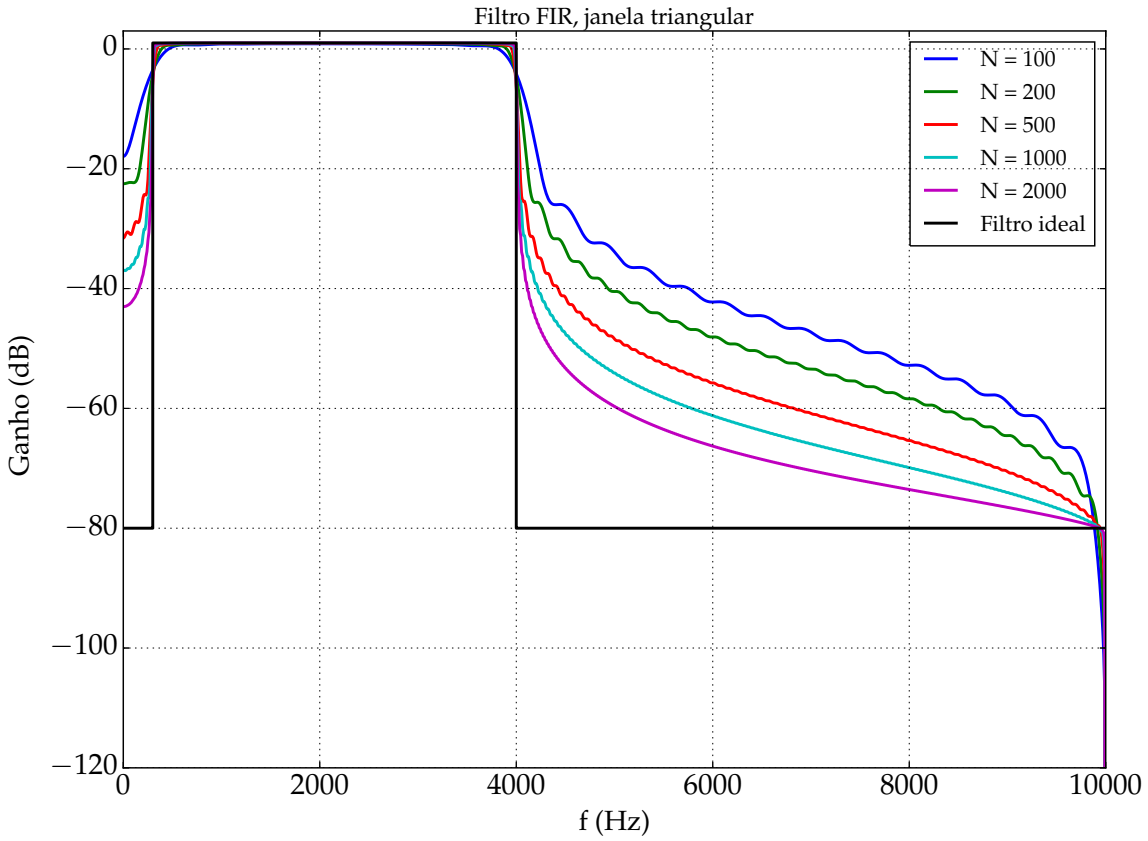

Figura 4.17: Resposta em frequência para filtro band-pass FIR com a função janela triangular.

É interessante notar que a implementação FIR para esse filtro permite obter uma resposta bem próxima da ideal. Com a função janela de Hamming e a retangular, temos oscilações na banda de parada, ao passo que, empregando-se a janela triangular, estas não acontecem ao custo de uma transição menos ágil.

A partir dos resultados obtidos para os três projetos, os quais confirmam aquilo que foi afirmado na revisão teórica, verifica-se o funcionamento adequado da ferramenta para aquilo que foi proposto no início deste trabalho.

4.5 Tempo de Processamento

Como os filtros analógicos e IIR, conforme observado no desenvolvimento deste trabalho, são implementados por meio da aplicação direta de fórmulas, seu tempo de processamento é desprezível e não será abordado.

Determinam-se os seguintes tempos de processamento para o projeto de filtros FIR, quando executado para um filtro simples em um computador com CPU Intel Core i7 rodando a 2.2 GHz, com 6 GB de RAM e rodando o Linux Ubuntu 15.04 com Python 3.4.3:

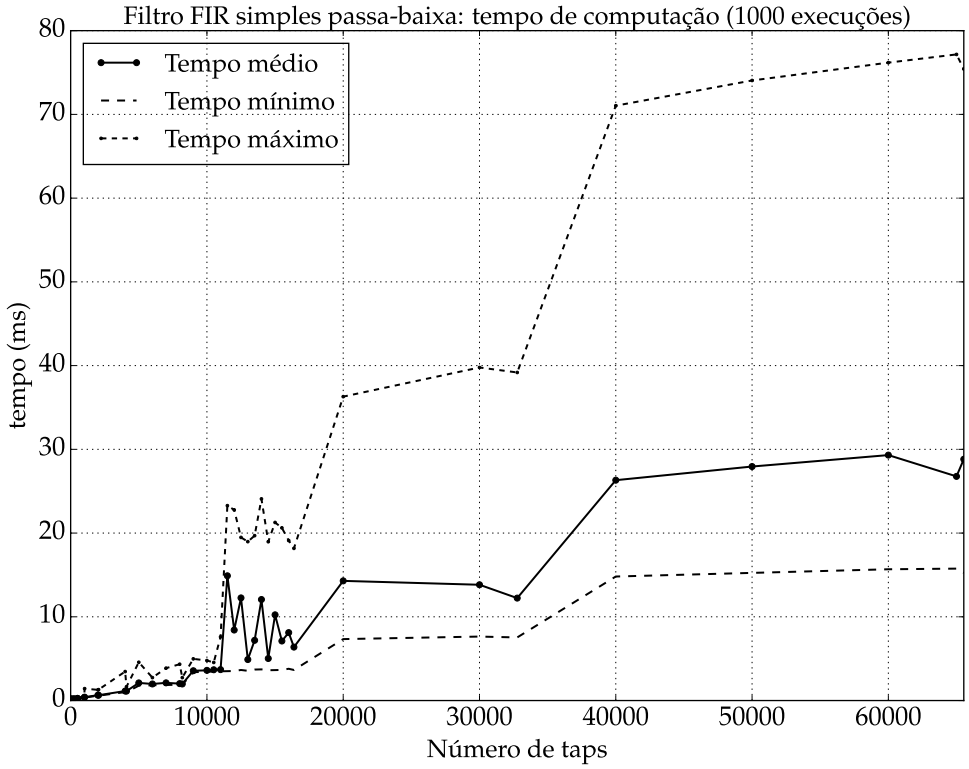

Figura 4.18: Tempo de processamento necessário para o projeto de um filtro FIR com N *taps*, pelo método do janelamento. Para cada N o algoritmo foi executado 1000 vezes.

Para comparação, um filtro similar foi projetado empregando-se o *MATLAB* versão R2013a no mesmo computador, tendo sido obtido o seguinte resultado:

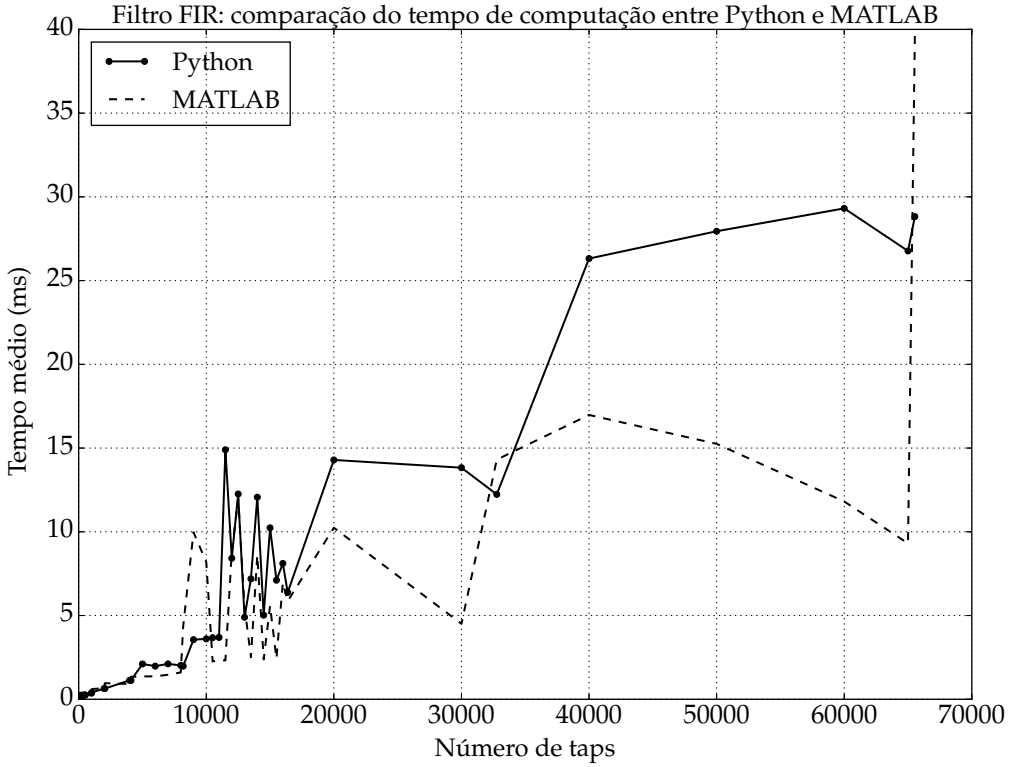

Figura 4.19: Comparação do tempo tempo de processamento necessário entre Python e MATLAB para o projeto, com as mesmas condições do item 4.18.

A partir do gráfico 4.18, podemos afirmar que o tempo de processamento gasto cresce de forma aproximadamente exponencial com o número de *taps* desejado. Já em 4.19, verifica-se que o MATLAB apresenta desempenho idêntico ou melhor ao Python; presume-se que esse fenômeno ocorre devido ao fato do MATLAB ter diversas otimizações em suas bibliotecas, porém não é possível confirmar essa hipótese visto que o MATLAB é um *software* de código fechado.

4.6 Comparação com Ferramentas já Existentes

Para guiar o desenvolvimento das funcionalidades da ferramenta elaborada no presente trabalho, realizou-se um estudo com algumas das diversas ferramentas já existentes no mercado.

Para o caso dos filtros analógicos, elas fornecem o circuito projetado; já para os filtros digitais, os coeficientes são fornecidos.

Também nota-se que o fluxo de trabalho em todas elas é similar: a partir das especificações desejadas, determinam-se funções de transferência ou circuitos que as implementam.

4.6.1 Analog Filter Wizard (Analog Devices)

Esta ferramenta gratuita, disponível em [13] é voltada ao projeto de filtros analógicos e implementa os filtros de Bessel, Butterworth e Chebyshev tipo I, com ordens limitadas a 10, permitindo ao usuário decidir entre menos estágios e menor tempo de acomodação.

Uma função interessante dessa ferramenta é permitir algumas otimizações para o circuito (por exemplo, menor consumo de energia ou menor ruído) e realizar a análise das tolerâncias dos componentes: pode-se verificar como o circuito irá se comportar perante a variabilidade dos resistores e capacitores empregados.

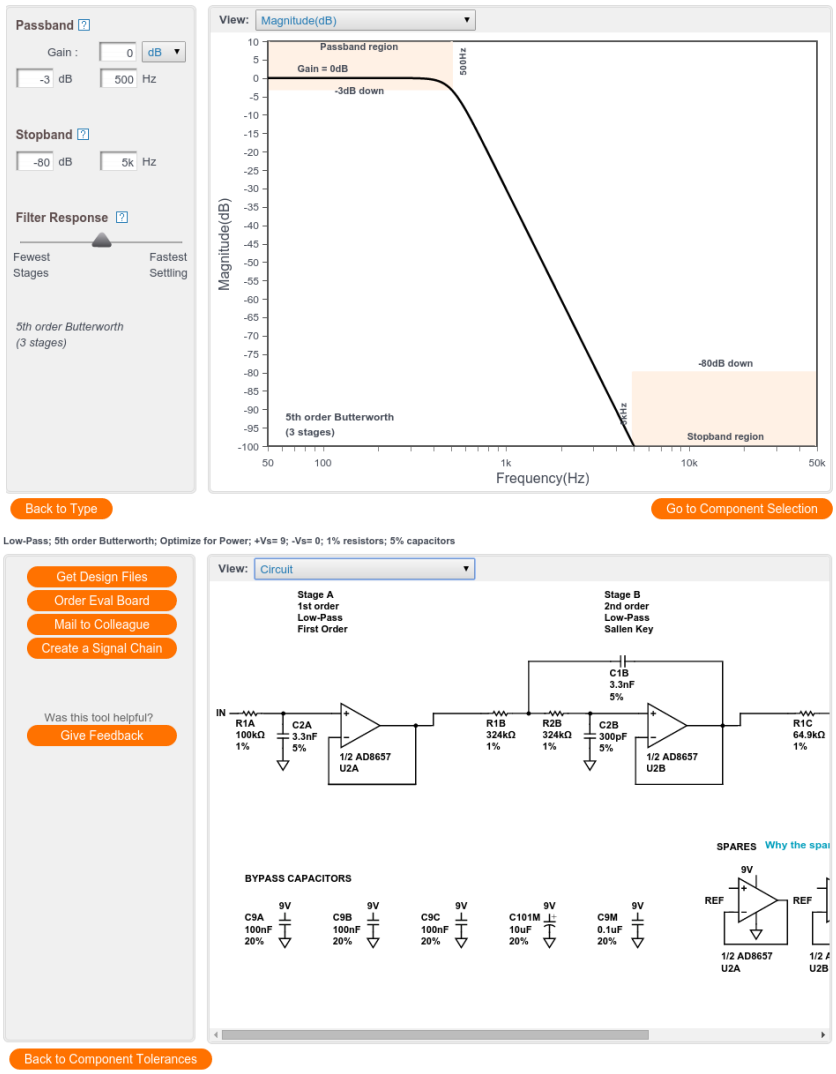

Figura 4.20: Exemplo da interface do *Analog Filter Wizard*: entrada de dados e resultados.

4.6.2 FilterPro (Texas Instruments)

Esta ferramenta disponível em [40] roda no *browser* (é escrita em *Flash*) e permite diversas otimizações no circuito projetado (reduzir o número de componentes ou o custo desses, maximizar a atenuação na banda de parada, minimizar o *ripple* na banda de passagem).

Assim como o *Analog Filter Wizard*, os circuitos são implementados utilizando-se as topologia *multiple feedback* ou *Sallen-Key*.

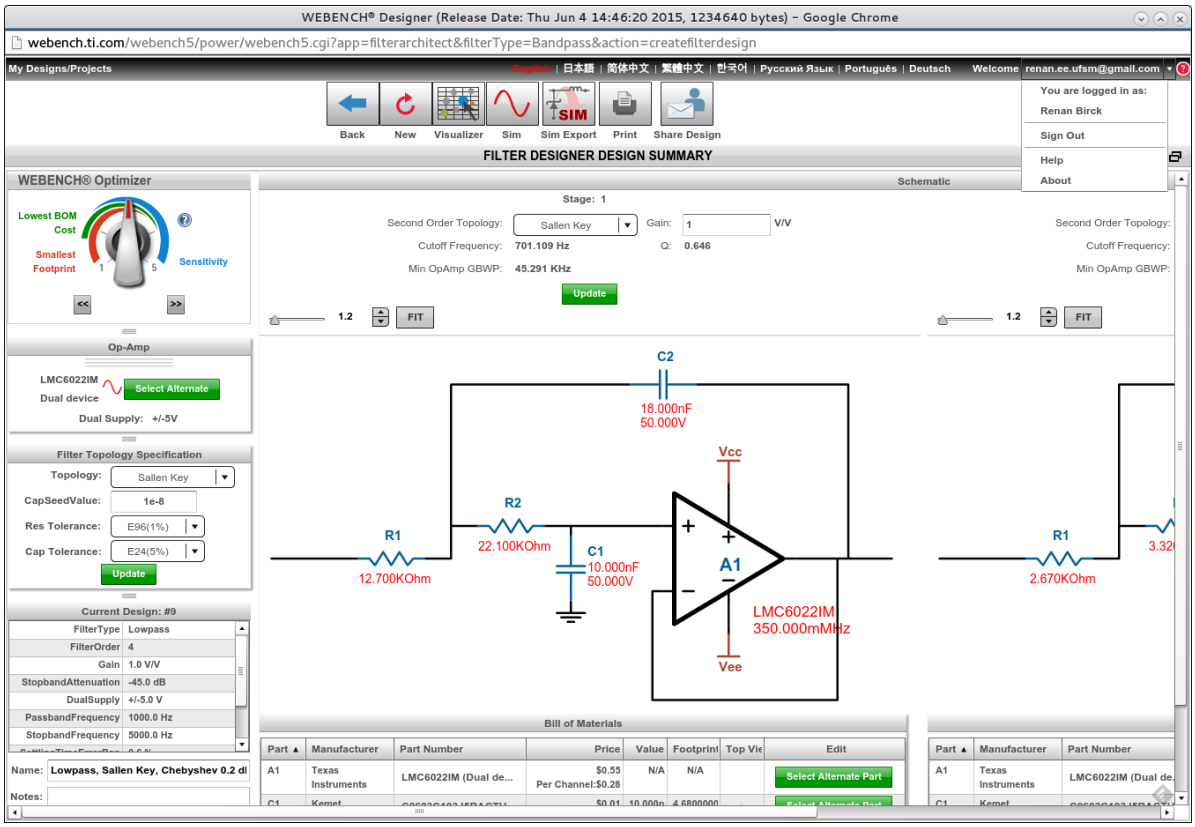

Figura 4.21: Exemplo da interface do *FilterPro*.

4.6.3 DSP System Toolbox (MATLAB)

Este pacote de ferramentas disponível para o MATLAB é voltado ao projeto de filtros digitais, sendo o mais completo dentre as ferramentas analisadas e cobrindo tanto filtros FIR quanto IIR - para os quais estão implementadas diversos algoritmos de cálculo (método da janela, Parks-McClellan, mínimos quadrados etc...).

Pode ser operado em linha de comando e por meio de interface gráfica, sendo que seu funcionamento e a funcionalidade são similares em ambos; enquanto a interface gráfica permite o projeto interativo, a linha de comando permite o emprego dos filtros projetados em rotinas

mais complexas para processamento de sinais.

Uma das suas funcionalidades mais relevantes, considerado que filtros digitais são amplamente empregados em sistemas embarcados, é a sua capacidade de gerar código-fonte em C e em VHDL para uso em CPUs.

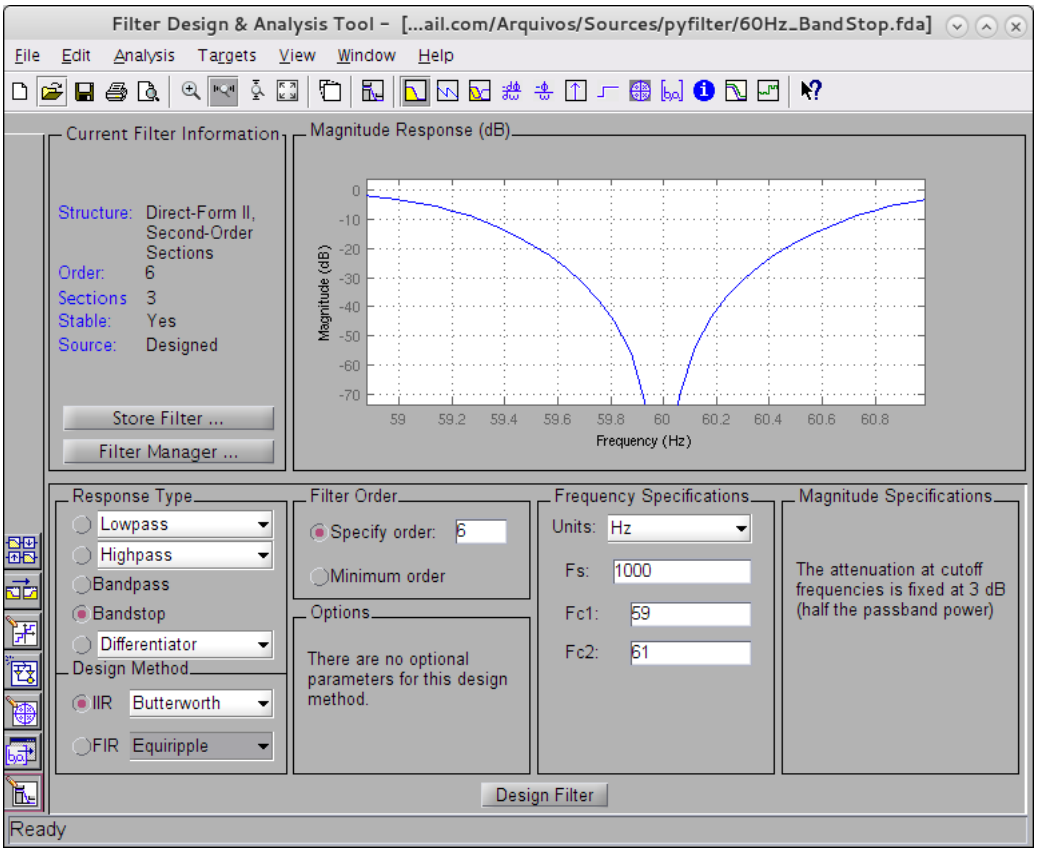

Figura 4.22: Exemplo da interface do *Filter Design and Analysis tool*.

Comparado às ferramentas existentes, a ferramenta apresentada no presente trabalho tem como principal vantagem não ter limitações de ordem do filtro projetado (excetuando-se, conforme descrito no desenvolvimento, o filtro de Bessel que é implementado por meio de tabelas que vão apenas até a 25ª ordem).

Outro importante aspecto é que as ferramentas acima descritas são *caixas pretas* de código fechado. Embora, para o usuário, isso muitas vezes não seja um fator limitante ou incômodo, sua funcionalidade está restrita àquela que os desenvolvedores da ferramenta implementaram: há pouca ou nenhuma possibilidade de expansão ou de integração em um algoritmo ou aplicação mais complexa.

A ferramenta desenvolvida aqui, entretanto, tem seu código aberto (*open source*) e também baseada em bibliotecas abertamente disponíveis. Dessa forma, além da vantagem imediata

do custo [9], ela pode ser modificada e expandida pelos seus usuários, que podem adicionar novas funcionalidades e estudar seu código-fonte para entender o funcionamento.

Todavia, não foram implementadas todas funcionalidades existentes nas ferramentas apresentadas: as ferramentas para projeto de filtros analógicos não realizam a síntese de circuitos, já para projeto digital não é gerado código para implementação em uma CPU. No item 5.1 descrevem-se melhorias que podem ser realizadas com o objetivo de dar continuidade a este trabalho e permitir que ele atinja o patamar das ferramentas existentes.

[9] Uma licença do MATLAB com as bibliotecas de processamento de sinais e de DSP custa em média US$ 5 mil

5 CONCLUSÃO

Neste trabalho, demonstrou-se com sucesso o desenvolvimento de uma ferramenta computacional para a síntese de filtros analógicos e digitais; sua vantagem imediata perante as ferramentas já existentes é ser gratuita e de código aberto, em contraste com as existentes, *caixas pretas* que permitem pouca ou nenhuma modificação ou estudo e, em alguns casos, possuem licenças bastante caras.

O usuário pode estudar o código-fonte da ferramenta e das bibliotecas empregadas e entender o seu funcionamento, os algoritmos empregados e as decisões tomadas durante o desenvolvimento. Isto é facilitado pela linguagem *Python*, cuja sintaxe é bastante clara e acessível mesmo para usuários com pouco conhecimento de programação.

Soma-se a essas vantagens a possibilidade de desenvolvimento de novas funcionalidades, que podem ser contribuídas por qualquer desenvolvedor interessado em dar continuidade ao trabalho: basta que ele desenvolva e teste a função desejada e submeta seu código para o autor do *software*, que se encarregará de incorporá-la ao programa. Da mesma forma, eventuais melhorias ou correções de *bugs* que sejam feitas nas bibliotecas empregadas neste projeto podem ser contribuídas aos seus projetos de origem.

Por fim, os algoritmos e códigos desenvolvidos neste trabalho podem ser aproveitados em outros projetos, assim possibilitando o desenvolvimento de novas ferramentas baseadas no que foi apresentado aqui.

O código-fonte do *software* desenvolvido aqui, juntamente com outras rotinas (por exemplo, *scripts* usados para teste e para geração dos gráficos) está disponível na plataforma GitHub, no endereço `https://github.com/renanbirck/pyfilter`, sob licença de *software* livre, para permitir que seu desenvolvimento seja continuado.

5.1 Futuras melhorias

Para a continuidade deste trabalho com o objetivo de torná-lo uma ferramenta completa para o projeto de filtros, possíveis melhorias incluem, mas não estão limitadas, às seguintes:

5.1.1 Análise de Estabilidade

Como afirmado na fundamentação teórica, filtros analógicos e filtros digitais IIR, por possuírem *feedback*, estão sujeitos à instabilidade, principalmente quando são empregados em sistemas de controle (aplicação comum para a filtragem de sinais).

Dessa forma, torna-se desejável que a ferramenta seja capaz de realizar o estudo da estabilidade do filtro projetado.

5.1.2 Análise de Sensitividade

Circuitos analógicos reais são implementados com componentes não-ideais, ao passo que os projetos de filtros são realizados presumindo-se que os dispositivos empregados são ideais. Como consequência, existe o risco do circuito implementado não corresponder àquilo que foi projetado.

Analisar a saída do circuito perante não-linearidades e variações dos componentes empregados nele torna-se uma funcionalidade necessária, a qual pode ser implementada interfaceando-se a ferramenta com um simulador de circuitos que será encarregado dos cálculos.

5.1.3 Aperfeiçoamentos na Interface Gráfica

Tornar a interface gráfica da ferramenta mais simples de usar e realizar melhorias no quesito de intuitividade e interação com o usuário; verificar a viabilidade do uso de outras bibliotecas de interface gráfica (por exemplo, a IUP - Portable User Interface [17], que também é de código aberto e é voltada a aplicações científicas).

5.1.4 Emprego de Algoritmos de Otimização

Como foi visto, é possível descrever o projeto de filtros como um problema de otimização matemática: deseja-se encontrar parâmetros para um sistema LTI que melhor aproximem uma resposta não-causal.

Esta estratégia, cuja viabilidade já foi demonstrada com sucesso na literatura (é a forma com que opera o algoritmo de Parks-McClellan; outros autores empregam diferentes técnicas de otimização, por exemplo, em [4] são empregados algoritmos genéticos), torna-se particularmente desejável quando são conhecidas as características do filtro desejado, porém não se conhece qual a ordem e quais as frequências que atendem essa especificação.

5.1.5 Geração de Código

Filtros digitais geralmente são implementados em CPUs programadas na linguagem C ou Assembly, ou em FPGAs programados em VHDL. Torna-se, então, desejável que a ferramenta seja capaz de produzir código nessas linguagens.

Isso levanta as considerações de implementação descritas na revisão teórica deste trabalho, como os tipos de dados das variáveis e questões relacionadas à precisão numérica, além das estruturas de implementação (por exemplo, formas diretas I e II) que deverão ser empregadas para evitar polinômios de alta ordem e suas instabilidades numéricas.

5.1.6 Interface em Linha de Comando e Biblioteca de Funções

Embora a interface gráfica seja uma ferramenta prática para o usuário da ferramenta, ela limita o usuário às funcionalidades que o seu desenvolvedor julga úteis.

Dessa forma, é desejável que as funções para projeto possam ser empregadas diretamente em outros códigos, inclusive como parte de um sistema maior.

Uma separação rígida entre interface e algoritmo também melhora a qualidade do código de ambos, pois cada parte pode ser testada, melhorada e modificada isoladamente, reduzindo-se o risco de afetar funcionalidades já existentes.

5.1.7 Realização de Análises

É desejável que o usuário possa analisar o filtro projetado quanto a resposta no tempo (por exemplo, verificar como ele se comporta com uma resposta ao impulso ou ao degrau unitário) e no plano complexo.

Isto permite que o usuário julgue o filtro obtido quanto à estabilidade e fornece uma ferramenta intuitiva para o entendimento de como a posição dos polos e zeros do filtro afeta as características dele.

5.1.8 Síntese de Circuitos

Filtros analógicos podem ser implementados com as topologias vistas na introdução; para colocar a ferramenta apresentada neste trabalho no mesmo patamar daquelas descritas no item 4.6, pode-se considerar que esta funcionalidade é essencial, devendo ser uma das prioridades de desenvolvimento.

Juntamente com um simulador de circuitos, torna-se possível analisar o comportamento dos filtros sob condições não-ideais tais como o uso de *op-amps* reais ou a variabilidade dos componentes.

REFERÊNCIAS

[1] ANICHE, M. **Test-Driven Development**. Disponível em http://tdd.caelum.com.br/. Acesso em 19 mai. 2015.

[2] **Application Note 738: Minimizing Component-Variation Sensitivity in Single Op Amp Filters**. Disponível em http://pdfserv.maximintegrated.com/en/an/AN738.pdf. Acesso em 15 jun. 2015.

[3] **Application Note 1762: A Beginner's Guide to Filter Topologies**. Disponível em http://pdfserv.maximintegrated.com/en/an/AN1762.pdf. Acesso em 01 abr. 2015.

[4] BARROS, A. M. **Projeto de Filtros FIR através de Algoritmos Genéticos**. Dissertação (Mestrado em Engenharia Elétrica e Informática) - Universidade Tecnológica Federal do Paraná. Curitiba, 2006.

[5] BUTTERWORTH, S. **On the Theory of Filter Amplifiers**. *Wireless Engineer*, v. 7, p 536-541, 1930.

[6] CARDOSO, A. **TDD, por que usar?**. Disponível em http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151. Acesso em 22 abr. 2015.

[7] DATTORRO, J. **The Implementation of Recursive Digital Filters for High-Fidelity Audio**. *Journal of the Audio Engineering Society*, v. 36, n. 11, p. 851-878, 1988.

[8] **Digital Signal Processors - Texas Instruments**. Disponível em http://www.ti.com/DSP. Acesso em 15 abr. 2015.

[9] DINIZ, P. S. SILVA, E. A. B. NETTO, S. L. **Processamento digital de sinais: projeto e análise de sistemas**. 2ª edição. Porto Alegre: Bookman, 2014. 1000p.

[10] **dsPIC30F Programmer's Reference Manual**. Disponível em http://ww1.microchip.com/downloads/en/DeviceDoc/sect3_4.pdf. Acesso em 15 abr. 2015.

[11] **FilterLab Filter Design software.** Disponível em http://www.microchip.com/pagehandler/en_us/devtools/filterlab-filter-design-software.html. Acesso em 22 abr. 2015.

[12] FORTUNATO, M. **Circuit Sensitivity With Emphasis On Analog Filters**. Disponível em http://www.ti.com/lit/ml/sprp524/sprp524.pdf. Acesso em 15 jun. 2015.

[13] **Filter Wizard**. Disponível em http://www.analog.com/designtools/en/filterwizard/#/specifications. Acesso em 11 jun. 2015.

[14] HARRIS, F, J. **On the use of windows for harmonic analysis with the discrete Fourier transform**. *Proceedings of the IEEE*, v. 66, n. 1, p. 51-83, 1978.

[15] HAYKIN, S. VAN VEEN, B. **Sinais e Sistemas**. 1ª edição. Porto Alegre: Bookman, 2001. 661 p.

[16] HOROWITZ, P. HILL, W. **The Art of Electronics**. 2ª edição. Cambridge: Cambridge University Press, 1989. 1125p.

[17] **IUP Portable User Interface**. Disponível em http://webserver2.tecgraf.puc-rio.br/iup/. Acesso em 09 jul. 2015.

[18] JUNG, W. **Op Amp Applications Handbook.** Disponível em http://www.analog.com/library/analogDialogue/archives/39-05/op_amp_applications_handbook.html. Acesso em 18 jun. 2015.

[19] KERWIN, W. J. HUELSMAN, L. P., NEWCOMB, R. W. **State Variable Synthesis for Insensitive Integrated Circuit Transfer Functions**. *IEEE Journal of Solid-State Circuits*, v. SC-2, n. 3, p. 87-92, 1967.

[20] MANCINI, R. et al. **Op-Amps for Everyone**. 4ª edição. Burlington: Newnes, 2003. 304p.

[21] MARQUES, A. A. de A. **Projeto e implementação em tempo real de filtros digitais utilizando microprocessadores de sinal**. 1996. 201p. Dissertação (Mestrado em Engenharia Eletrotécnica e de Computadores) - Faculdade de Engenharia da Universidade do Porto. Porto, 1996.

[22] MCCLELLAN, J. PARKS, T. **A unified approach to the design of optimum FIR linear-phase digital filters**. *IEEE Transactions on Circuit Theory*, v. 20, n. 6, p. 697-701, 1973.

[23] MCCLELLAN, J. PARKS, T. RABINER, L. **A Computer Program for Designing Optimum FIR Phase Digital Filters**. *IEEE Transactions on Audio and Electroacoustics*, v. AU-21, n. 6, p. 506-526, 1973.

[24] NumPy and SciPy documentation. Disponível em `http://docs.scipy.org/doc/`. Acesso em 22 mar. 2015.

[25] OPPENHEIM, A. V. WILLSKY, A.S. **Sinais e Sistemas**. 2ª edição. São Paulo: Pearson Prentice-Hall, 2010. 568p.

[26] ORFANIDIS, S. J. **Lecture Notes on Elliptic Filter Design**. Disponível em `http://eceweb1.rutgers.edu/~orfanidi/ece521/notes.pdf`. Acesso em 13 mai. 2015.

[27] PAARMANN, L. D. **Design and Analysis of Analog Filters: A Signal Processing Perspective**. New York: Kluwer Academic Publishers, 2003. 453p.

[28] Python 3.4.3 Documentation. Disponível em `https://docs.python.org/3/`. Acesso em 03 mar. 2015.

[29] QUÉLHAS, M. F. **Projeto de Filtros IIR por Mapeamento de Polos e Zeros**. 2010. 132p. Tese (Doutorado em Engenharia Elétrica) - COPPE, Universidade Federal do Rio de Janeiro. Rio de Janeiro, 2010.

[30] RAZAVI, B. **Fundamentos de Microeletrônica**. 1ª edição. Rio de Janeiro: LTC, 2013. 728p.

[31] ROWELL, D. **Signal Processing: Continuous and Discrete**. Disponível em `http://ocw.mit.edu/courses/mechanical-engineering/2-161-signal-processing-continuous-and-discrete-fall-2008/lecture-notes/`. Acesso em 13 mai. 2015.

[32] SALLEN, R. P. KEY, E. L. **A Practical Method of Designing RC Active Filters**. *IRE Transactions on Circuit Theory*, v. 2, n. 1, p. 74-85, 1955.

[33] SANCHOTENE, R. RODRIGUES, C. R. **A 120s-time-constant 2^{nd} order Butterworth low-pass Gm-C filter based on a novel Reverse Cascode topology**. EMICRO 2015, Santa Maria, 2015.

[34] SEDRA, A. S. SMITH, K. C. **Microeletrônica**. 5ª edição. São Paulo: Pearson Prentice-Hall, 2007. 848p.

[35] SOUZA, A. A. V. B. **Análise de Operadores Explícitos de Migração no Domínio** $\omega - x$. 2010. 60f. Trabalho de Conclusão de Curso (Graduação em Geofísica) - Instituto de Geociências, Universidade Federal da Bahia, Salvador, 2000.

[36] SMITH, S. **The Scientist and Engineer's Guide to Digital Signal Processing**. 1ª edição. s.l: California Technical Publishers, 1997. 626p. Disponível em `http://www.dspguide.com/`. Acesso em 11 mar. 2015.

[37] SymPy Documentation. Disponível em `http://docs.sympy.org/latest/index.html`. Acesso em 25 mai. 2015.

[38] **TUTORIAL 727: Filter Design Using Integrator Blocks**. Disponível em `http://pdfserv.maximintegrated.com/en/an/AN727.pdf`. Acesso em 15 jun. 2015.

[39] **UAF42: Universal Active Filter**. Disponível em `http://www.ti.com/lit/ds/symlink/uaf42.pdf`. Acesso em 01 abr. 2015.

[40] **WEBENCH Filter Designer**. Disponível em `http://www.ti.com/lsds/ti/analog/webench/webench-filters.page`. Acesso em 22 abr. 2015.

[41] ZUMBALEN, H. **Linear Circuit Design Handbook**. 1ª edição. Burlington: Newnes, 2008. 960p.