Equalizador de Áudio com Visualização de Espectro em Tempo Real

Gabriel Tambara

Julho de 2025

1 Introdução

Este relatório apresenta o desenvolvimento de um equalizador gráfico de áudio em tempo real com cinco bandas e visualização espectral baseada em FFT, implementado em Python. A aplicação permite importar arquivos MP3, aplicar filtros FIR em tempo real e visualizar a distribuição de energia em 10 bandas logarítmicas.

2 Objetivos

- Implementar um equalizador com cinco bandas de frequência (100 Hz, 330 Hz, 1 kHz, 3.3 kHz, 10 kHz);
- Calcular os coeficientes dos filtros FIR com janela de Hamming;
- Reproduzir áudio com ganho ajustável por banda;
- Visualizar o espectro de frequência com 10 bandas logarítmicas;
- Criar uma interface gráfica amigável.

3 Parâmetros do Sistema

- Frequência de amostragem: $F_s = 44100$ Hz
- Tamanho do bloco: $N = 1024$ amostras
- Ordem do filtro FIR: $M = 401$
- Bandas principais: 100 Hz, 330 Hz, 1 kHz, 3.3 kHz, 10 kHz

4 Fundamentação Teórica

4.1 Representação e Amostragem do Sinal de Áudio

Um sinal de áudio no domínio contínuo $x(t)$ é convertido para o domínio discreto $x[n]$ através da amostragem, com uma frequência de amostragem $F_s = 44.1$ kHz, que é padrão em mídias digitais como CDs e arquivos MP3.

A relação entre o sinal contínuo e o sinal discreto é dada por:

$$x[n] = x(nT) \quad \text{com } T = \frac{1}{F_s}$$

Neste projeto, processamos blocos de $N = 1024$ amostras por vez.

4.2 Objetivo do Equalizador

O equalizador implementado divide o espectro de frequência do sinal em 5 bandas com frequências centrais em:

$$f_c = \{100, 330, 1000, 3300, 10000\} \text{ Hz}$$

As bandas são obtidas por filtros passa-banda com frequências de corte definidas entre as frequências centrais de forma logarítmica:

$$f_{ci} = \sqrt{f_{c_i} \cdot f_{c_{i+1}}}$$

Isso resulta em 6 frequências de corte:

$$\{0, f_{c1}, f_{c2}, f_{c3}, f_{c4}, F_s/2\}$$

Cada banda será filtrada por um filtro $h_i[n]$ e amplificada/atenuada por um ganho A_i definido pelo usuário.

4.3 Projeto dos Filtros FIR com Janela de Hamming

Filtros do tipo FIR (Finite Impulse Response) são muito utilizados em aplicações de processamento digital de sinais devido à sua estabilidade incondicional e facilidade de implementação. Neste projeto, implementamos cinco filtros FIR do tipo passa-banda, cada um projetado para atuar sobre uma banda específica do espectro audível.

4.3.1 Conceito Teórico

O objetivo de um filtro passa-banda é permitir a passagem de frequências dentro de um intervalo $[f_{\text{low}}, f_{\text{high}}]$ e atenuar as demais. A resposta ao impulso ideal de um filtro desse tipo é obtida pela diferença entre dois filtros passa-baixa ideais, um com frequência de corte f_{high} e outro com f_{low}.

A equação da resposta ao impulso ideal é dada por:

$$h_{\text{ideal}}[n] = \frac{\sin\left(2\pi f_{\text{high}} \cdot m/F_s\right) - \sin\left(2\pi f_{\text{low}} \cdot m/F_s\right)}{\pi m}$$

onde:

- F_s é a frequência de amostragem do sistema (44100 Hz neste projeto);
- f_{low} e f_{high} são as frequências de corte inferior e superior da banda;
- $m = n - \frac{M-1}{2}$ é o índice centralizado da sequência do filtro;
- M é a ordem do filtro (número total de coeficientes).

Essa fórmula resulta de aplicar a transformada inversa de Fourier sobre a função de resposta em frequência ideal (retangular) de um filtro passa-banda.

4.3.2 Problemas da Implementação Direta

A função $h_{\text{ideal}}[n]$ não é realizável diretamente porque possui suporte infinito (ou seja, é uma sequência infinita). Para torná-la implementável, fazemos um recorte (janela) da sequência em um número finito de amostras M. No entanto, esse recorte abrupto gera efeitos indesejados como *ripple* e *gibbs phenomenon*.

Para suavizar essas descontinuidades, aplicamos uma **janela de suavização** — neste caso, a **janela de Hamming**:

$$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{M-1}\right), \quad 0 \leq n < M$$

Essa janela reduz as componentes de alta frequência introduzidas pelo recorte abrupto, ao suavizar as bordas da função $h_{\text{ideal}}[n]$.

4.3.3 Filtro Final Realizável

A resposta ao impulso final do filtro é dada por:

$$h[n] = h_{\text{ideal}}[n] \cdot w[n]$$

O produto ponto a ponto entre a sequência ideal e a janela de Hamming gera uma versão suavizada e truncada do filtro, agora com comprimento finito M.

4.3.4 Escolha da Ordem do Filtro

A ordem do filtro M determina sua resolução espectral (largura de transição) e sua capacidade de atenuar bandas indesejadas. Neste projeto, utilizamos $M = 401$, o que oferece uma boa separação entre bandas e rejeição adequada fora da faixa de passagem.

A largura da transição Δf entre a faixa de passagem e a faixa de rejeição de um filtro FIR com janela pode ser aproximadamente estimada por:

$$\Delta f \approx \frac{3.1 \cdot F_s}{M}$$

Substituindo os valores:

$$\Delta f \approx \frac{3.1 \cdot 44100}{401} \approx 341 \text{ Hz}$$

Isso significa que cada filtro terá uma transição suave entre as bandas adjacentes, evitando descontinuidades abruptas no espectro, o que é desejável para aplicações de áudio se tratando de sonoridade.

4.3.5 Cobertura Contínua do Espectro

Para garantir que as bandas se conectem suavemente, as frequências de corte dos filtros foram definidas como a **média geométrica** entre as frequências centrais adjacentes:

$$f_{ci} = \sqrt{f_{c_i} \cdot f_{c_{i+1}}}$$

Com isso, temos:

Corte inferior da banda 1: 0 Hz

Corte superior da banda 1: $\sqrt{100 \cdot 330} \approx 181.6$ Hz

E assim por diante, até a última banda, cuja frequência de corte superior é $F_s/2 = 22050$ Hz.

Dessa forma, as 5 bandas se sobrepõem levemente, garantindo uma transição contínua de energia entre regiões espectrais, o que é especialmente importante para evitar artefatos audíveis durante ajustes de ganho.

4.4 Modelo Matemático do Equalizador

O sinal de saída de cada banda é obtido pela convolução do sinal de entrada com o filtro correspondente, seguida de amplificação:

$$x_i[n] = x[n] * h_i[n] \quad \text{e} \quad y_i[n] = A_i \cdot x_i[n]$$

A saída final do equalizador é a soma dos sinais amplificados:

$$y[n] = \sum_{i=1}^{5} A_i \cdot (x[n] * h_i[n])$$

Pela propriedade distributiva da convolução:

$$y[n] = x[n] * \left(\sum_{i=1}^{5} A_i \cdot h_i[n] \right) = x[n] * h[n]$$

onde

$$h[n] = \sum_{i=1}^{5} A_i \cdot h_i[n]$$

Essa abordagem reduz o custo computacional ao aplicar apenas uma convolução com o filtro combinado $h[n]$.

O filtro final aplicado é a combinação linear ponderada dos cinco filtros FIR passa-banda:

$$h[n] = \sum_{i=1}^{5} A_i \cdot h_i[n]$$

permitindo aplicar uma única convolução eficiente.

4.5 Controle de Ganho em dB

A interface do equalizador permite controle do ganho em dB. A conversão entre ganho em decibéis e ganho linear é dada por:

$$A_i = 10^{G_i/20}$$

onde G_i é o ganho ajustado pelo usuário em decibéis na banda i.

Com essa transformação, o usuário pode atenuar ou amplificar cada banda de forma intuitiva.

4.6 Análise Espectral em Tempo Real

Para exibir o conteúdo espectral do sinal de saída em tempo real, implementamos um analisador de espectro de 10 bandas. Foi implementada manualmente uma Transformada Rápida de Fourier (FFT) eficiente, garantindo o entendimento detalhado do processo. Antes da FFT, o sinal é multiplicado por uma janela de Hamming para minimizar efeitos de descontinuidade e vazamento espectral.

O sinal de saída $y[n]$ é multiplicado por uma janela de Hamming e passa por uma Transformada Rápida de Fourier (FFT), resultando em $Y[k]$:

$$Y[k] = \sum_{n=0}^{N-1} y[n] \cdot w[n] \cdot e^{-j2\pi kn/N}$$

O espectro de magnitude é obtido por:

$$|Y[k]| = \sqrt{\Re(Y[k])^2 + \Im(Y[k])^2}$$

Agrupamos as componentes em 10 bandas logarítmicas uniformes de 20 Hz até 22.05 kHz (a Nyquist frequency), e computamos a média RMS de cada banda para determinar a altura das barras do gráfico. As barras azuis mostram o espectro original, enquanto as barras brancas representam o espectro após equalização. Quando o valor da barra branca excede o da barra azul correspondente, a região de excesso é destacada em vermelho, facilitando a visualização de ganhos adicionais em cada banda.

4.7 Resumo da Arquitetura

O sistema implementado segue o seguinte fluxo de processamento:

1. O usuário importa um sinal de áudio em 44.1 kHz (MP3).
2. O sinal é dividido em blocos de $N = 1024$ amostras.
3. Cada bloco é armazenado em um buffer de convolução e processado por:
   - Geração do filtro composto $h[n] = \sum A_i h_i[n]$.
   - Convolução eficiente do bloco com $h[n]$.
4. O sinal de saída é enviado ao áudio e ao analisador de espectro.
5. O espectro de 10 bandas é atualizado em tempo real.

5 Resultados

5.1 Interface gráfica

A seguinte figura ilustra o resultado final com uma música pop já sob efeito dos filtros com diferentes ganhos. Percebe-se, portanto, a qualidade do sliders e a visualização em tempo-real.

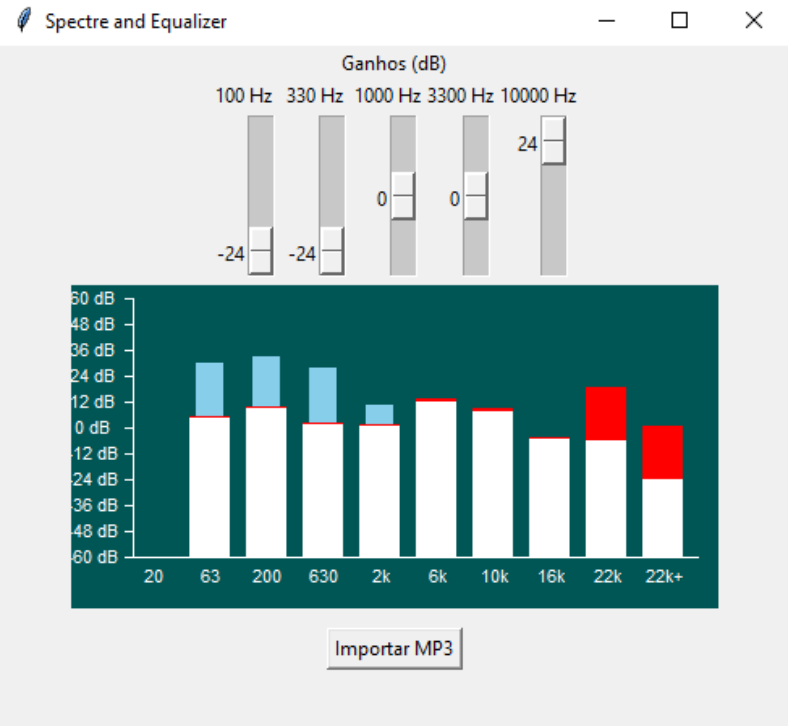

Figura 1: Display da interface gráfica em uso com filtros em ação.

5.2 Análise dos Filtros FIR

Para visualizar o comportamento dos filtros projetados, traçamos a resposta em frequência de cada filtro FIR passa-banda no intervalo de frequências de interesse. O gráfico abaixo ilustra as magnitudes (em dB) dos cinco filtros com seus respectivos ganhos unitários.

Observa-se que os filtros apresentam boa seletividade em suas bandas centrais (100 Hz, 330 Hz, 1 kHz, 3.3 kHz e 10 kHz), com as bordas suavemente transicionando devido ao uso da janela de Hamming. A sobreposição (overlap) entre as bandas adjacentes é controlada pela definição das frequências de corte como médias geométricas entre os centros, garantindo cobertura contínua do espectro sem lacunas.

Esse overlap permite uma transição suave no ganho ao ajustar os sliders do equalizador, evitando descontinuidades bruscas na resposta em frequência total do sistema. Em compensação, percebe-se interferência em frequências mais distantes, mesmo que em pequena intensidade.

5.3 Considerações Finais

O sistema foi projetado com foco em:

- **Desempenho em tempo real**, com convolução vetorizada eficiente.
- **Alta qualidade sonora**, com filtros FIR longos (M=201) e janela de Hamming.
- **Interatividade**, com controle intuitivo de ganhos por bandas.
- **Precisão**, com FFT implementada manualmente para análise espectral sem bibliotecas externas.

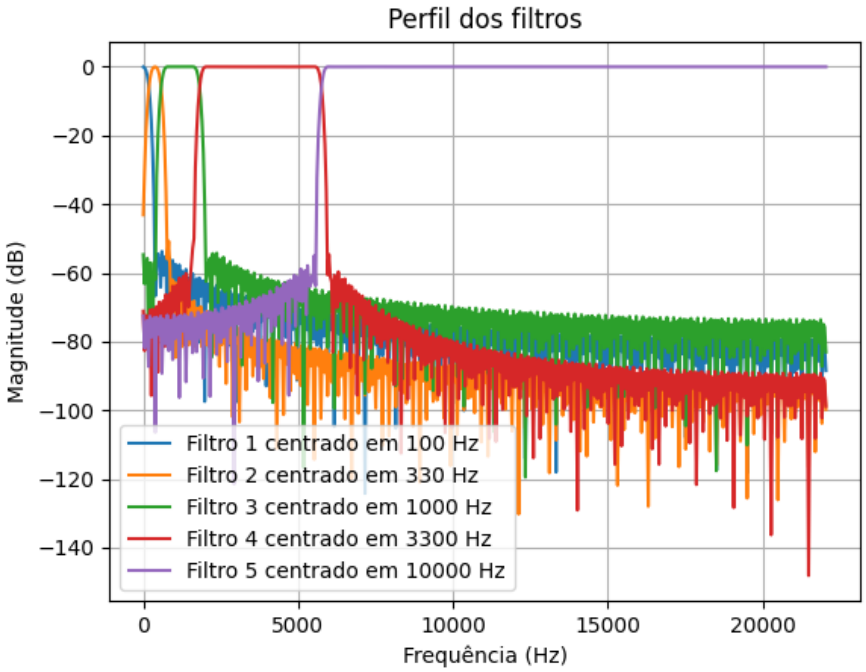

Figura 2: Resposta em frequência dos filtros FIR passa-banda com janela de Hamming e ordem $M = 401$. As linhas mostram a magnitude em dB em função da frequência.

A implementação respeita os princípios teóricos do processamento digital de sinais e proporciona uma base sólida para futuras extensões, como mais bandas, filtros IIR ou análise espectral mais refinada.

6 Interface Gráfica

A interface foi construída com `Tkinter`. Ela inclui:

- Sliders verticais para ajuste de ganho por banda;
- Um botão para importar arquivos MP3;
- Um canvas com 10 barras representando a análise espectral em tempo real.

7 Conclusão

O projeto demonstra o uso de processamento digital de sinais em tempo real, combinando filtros FIR com visualização FFT e GUI em Python. O sistema é útil como ferramenta de ensino e análise de áudio, permitindo ajustes dinâmicos e visualização intuitiva.

8 Código-Fonte

Listing 1: Código do projeto

```
import numpy as np
import sounddevice as sd
import tkinter as tk
from tkinter import ttk, filedialog
from pydub import AudioSegment
import threading
import queue
import matplotlib.pyplot as plt

# Constants
FS = 44100
N = 1024
M = 401
BANDS = [100, 330, 1000, 3300, 10000]

# Calculate cutoff frequencies as midpoints (geometric mean) between
    centers
CUTOFFS = [np.sqrt(BANDS[i] * BANDS[i+1]) for i in range(len(BANDS)-1)]
CUTOFFS = [0] + CUTOFFS + [FS/2]

# FIR Bandpass filter design with Hamming window
def design_bandpass(low, high, fs, M):
    h = []
    for n in range(M):
        m = n - (M-1)/2
        if m == 0:
            val = 2*(high - low)/fs
        else:
            val = (np.sin(2*np.pi*high*m/fs) - np.sin(2*np.pi*low*m/fs))
                / (np.pi*m)
        w = 0.54 - 0.46*np.cos(2*np.pi*n/(M-1))
        h.append(val * w)
    return np.array(h)

filters = [design_bandpass(CUTOFFS[i], CUTOFFS[i+1], FS, M) for i in
    range(len(BANDS))]

root = tk.Tk()
root.title("Spectre and Equalizer")

gains_dB = [tk.DoubleVar(value=0.0) for _ in range(5)]
def dB_to_linear(db): return 10**(db/20)

input_buffer = np.zeros(N + M - 1)
imported_audio = np.array([])
audio_pos = 0
spec_queue = queue.Queue()

def fast_convolve(x, h):
    x = np.asarray(x)
    h = np.asarray(h)[::-1]
    strides = np.lib.stride_tricks.sliding_window_view(x, len(h))
```

```
    return np.sum(strides * h, axis=1)

def fft(x):
    x = np.asarray(x, dtype=complex)
    N = len(x)
    if N == 0:
        return np.array([], dtype=complex)
    if N == 1:
        return x
    if N % 2 != 0:
        next_pow2 = 1 << (N - 1).bit_length()
        x = np.pad(x, (0, next_pow2 - N), mode='constant')
        N = next_pow2
    even = fft(x[::2])
    odd = fft(x[1::2])
    min_len = min(len(even), len(odd))
    even = even[:min_len]
    odd = odd[:min_len]
    terms = np.exp(-2j * np.pi * np.arange(min_len) / (2 * min_len)) *
       odd
    return np.concatenate([even + terms, even - terms])

def rfft(x):
    x = np.asarray(x, dtype=float)
    N = x.shape[0]
    X = fft(x)
    return X[:N//2 + 1]

def rfftfreq(n, d=1.0):
    return np.arange(n//2 + 1) / (n * d)

def next_power_of_2(n):
    return 1 << (n - 1).bit_length()

def compute_fft_bands(samples):
    window = np.hamming(len(samples))
    windowed = samples * window
    n = len(windowed)
    if (n & (n - 1)) != 0:
        target_len = next_power_of_2(n)
        pad_width = target_len - n
        windowed = np.pad(windowed, (0, pad_width), mode='constant')
    spectrum = rfft(windowed)
    freqs = rfftfreq(len(windowed), 1 / FS)
    magnitudes = np.abs(spectrum)
    bands = np.logspace(np.log10(20), np.log10(FS / 2), num=11)
    bar_values = []
    for i in range(10):
        idx = np.where((freqs >= bands[i]) & (freqs < bands[i+1]))[0]
        if len(idx) > 0:
            bar_values.append(np.sqrt(np.mean(magnitudes[idx] ** 2)))
        else:
            bar_values.append(0)
    return bar_values

def audio_callback(outdata, frames, time, status):
    global input_buffer, audio_pos, imported_audio
    if len(imported_audio) > 0:
```

```
        if audio_pos + N > len(imported_audio):
            audio_pos = 0
        block = imported_audio[audio_pos:audio_pos+N]
        audio_pos += N
    else:
        block = np.random.normal(0, 1e-6, N).astype(np.float32)

    input_buffer[:-N] = input_buffer[N:]
    input_buffer[-N:] = block

    combined_filter = np.zeros(M)
    for i in range(5):
        gain = dB_to_linear(gains_dB[i].get())
        combined_filter += gain * filters[i]

    y = fast_convolve(input_buffer, combined_filter)
    y = np.clip(y, -1, 1).astype(np.float32)

    outdata[:, 0] = y
    spec_queue.put((block.copy(), y.copy()))

def spectrum_visualizer(canvas, bars_eq, bars_red):
    while True:
        try:
            # Flush old items, keep only the most recent
            while True:
                item = spec_queue.get(timeout=0.1)
                if spec_queue.empty():
                    break
        except queue.Empty:
            continue

        original, equalized = item

        mags_orig = compute_fft_bands(original)
        mags_eq = compute_fft_bands(equalized)

        for i in range(10):
            # convert magnitude to dB, clamp to Y range
            mag_orig = max(mags_orig[i], 1e-10)
            mag_eq = max(mags_eq[i], 1e-10)

            db_orig = 20 * np.log10(mag_orig)
            db_eq = 20 * np.log10(mag_eq)

            db_orig = max(Y_TICKS_DB[0], min(db_orig, Y_TICKS_DB[-1]))
            db_eq = max(Y_TICKS_DB[0], min(db_eq, Y_TICKS_DB[-1]))

            rel_orig = (db_orig - Y_TICKS_DB[0]) / (Y_TICKS_DB[-1] -
                Y_TICKS_DB[0])
            rel_eq = (db_eq - Y_TICKS_DB[0]) / (Y_TICKS_DB[-1] -
                Y_TICKS_DB[0])

            height_orig = rel_orig * (CANVAS_HEIGHT - BOTTOM_MARGIN -
                10)
            height_eq = rel_eq * (CANVAS_HEIGHT - BOTTOM_MARGIN - 10)

            x0 = LEFT_MARGIN + i * (BAR_WIDTH + BAR_SPACING)
```

```
            x1 = x0 + BAR_WIDTH

            # Blue bar (original)
            canvas.coords(bars_orig[i], x0 + 4, CANVAS_HEIGHT -
               BOTTOM_MARGIN, x1 - 4, CANVAS_HEIGHT - BOTTOM_MARGIN -
               height_orig)

            # White part of equalized bar: minimum of eq and orig
               heights
            white_height = min(height_eq, height_orig)
            canvas.coords(bars_eq[i], x0, CANVAS_HEIGHT - BOTTOM_MARGIN,
                x1, CANVAS_HEIGHT - BOTTOM_MARGIN - white_height)

            # Red part of equalized bar: excess over original (if any)
            red_height = max(0, height_eq - height_orig)
            canvas.coords(bars_red[i], x0, CANVAS_HEIGHT - BOTTOM_MARGIN
                - white_height, x1, CANVAS_HEIGHT - BOTTOM_MARGIN -
               white_height - red_height)

# Load MP3
def load_mp3():
    global imported_audio, audio_pos
    file_path = filedialog.askopenfilename(filetypes=[("MP3 files", "*.
       mp3")])
    if file_path:
        print("Importando:", file_path)
        audio = AudioSegment.from_file(file_path).set_channels(1).
           set_frame_rate(FS)
        samples = np.array(audio.get_array_of_samples()).astype(np.
           float32) / (2**15)
        imported_audio = samples
        audio_pos = 0

# GUI setup
tk.Label(root, text="Ganhos (dB)").pack()
frame = tk.Frame(root)
frame.pack()
for i, freq in enumerate(BANDS):
    ttk.Label(frame, text=f"{freq} Hz").grid(row=0, column=i)
    scale = tk.Scale(frame, from_=24, to=-24, variable=gains_dB[i])
    scale.grid(row=1, column=i)

# Canvas setup
CANVAS_WIDTH = 400
CANVAS_HEIGHT = 200
BAR_WIDTH = 25
BAR_SPACING = 10
BOTTOM_MARGIN = 30
LEFT_MARGIN = 40

canvas = tk.Canvas(root, width=CANVAS_WIDTH, height=CANVAS_HEIGHT, bg="
   #005555")
canvas.pack()

bars_eq = []
bars_orig = []
bars_red = []
```

```
for i in range(10):
    x0 = LEFT_MARGIN + i * (BAR_WIDTH + BAR_SPACING)
    x1 = x0 + BAR_WIDTH

    bars_orig.append(canvas.create_rectangle(x0 + 4, CANVAS_HEIGHT -
        BOTTOM_MARGIN, x1 - 4, CANVAS_HEIGHT - BOTTOM_MARGIN, fill='
        skyblue', width=0))
    bars_eq.append(canvas.create_rectangle(x0, CANVAS_HEIGHT -
        BOTTOM_MARGIN, x1, CANVAS_HEIGHT - BOTTOM_MARGIN, fill='white',
        width=0))
    bars_red.append(canvas.create_rectangle(x0, CANVAS_HEIGHT -
        BOTTOM_MARGIN, x1, CANVAS_HEIGHT - BOTTOM_MARGIN, fill='red',
        width=0))

# Draw axes
canvas.create_line(LEFT_MARGIN, 10, LEFT_MARGIN, CANVAS_HEIGHT -
    BOTTOM_MARGIN, fill='white')  # Y axis
canvas.create_line(LEFT_MARGIN, CANVAS_HEIGHT - BOTTOM_MARGIN,
    CANVAS_WIDTH - 10, CANVAS_HEIGHT - BOTTOM_MARGIN, fill='white')  # X
    axis

# Y-axis ticks and labels in dB
Y_TICKS_DB = list(range(-60, 61, 12))  # Uniform spacing for nice visual
for db in Y_TICKS_DB:
    rel = (db - Y_TICKS_DB[0]) / (Y_TICKS_DB[-1] - Y_TICKS_DB[0])
    y = CANVAS_HEIGHT - BOTTOM_MARGIN - rel * (CANVAS_HEIGHT -
        BOTTOM_MARGIN - 10)
    canvas.create_line(LEFT_MARGIN - 5, y, LEFT_MARGIN, y, fill='white')
    canvas.create_text(LEFT_MARGIN - 25, y, text=f"{db} dB", fill='white
        ', font=("Arial", 8))

# X-axis frequency labels
bands_labels = ["20", "63", "200", "630", "2k", "6k", "10k", "16k", "22k
    ", "22k+"]
for i in range(10):
    x = LEFT_MARGIN + i * (BAR_WIDTH + BAR_SPACING) + BAR_WIDTH / 2
    canvas.create_text(x, CANVAS_HEIGHT - BOTTOM_MARGIN + 12, text=
        bands_labels[i], fill='white', font=("Arial", 8))

# MP3 Button
btn = tk.Button(root, text="Importar MP3", command=load_mp3)
btn.pack(pady=10)

# Start visualizer thread and audio stream
threading.Thread(target=spectrum_visualizer, args=(canvas, bars_eq,
    bars_red), daemon=True).start()
stream = sd.OutputStream(samplerate=FS, blocksize=N, dtype='float32',
    channels=1, callback=audio_callback)
stream.start()
root.mainloop()
stream.stop()

# Plot filters magnitude response (optional)
for i, filt in enumerate(filters):
    w = np.linspace(0, np.pi, 512)
    H = np.abs(np.fft.fft(filt, 1024))[:512]
    freqs = w * FS / (2 * np.pi)
```

```
    plt.plot(freqs, 20 * np.log10(H + 1e-10), label=f"Filtro {i+1}
        centrado em {BANDS[i]} Hz")

plt.title("Perfil dos filtros")
plt.xlabel("Frequ ncia (Hz)")
plt.ylabel("Magnitude (dB)")
plt.legend()
plt.grid()
plt.show()
```