HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA: CÔNG NGHỆ THÔNG TIN

HỌC PHẦN: XỬ LÍ TÍN HIỆU SỐ

MÃ MÔN HỌC: ELE1330

BÀI TẬP LỚN CUỐI KHÓA MÔN XỬ LÍ TÍN HIỆU SỐ

GIẢNG VIÊN : TRẦN TUẤN ANH

HỌ VÀ TÊN : NGUYỄN THẾ HÙNG

MSV : B24DCCN256

SĐT : 0364904312

Hà nội - Tháng 10/2025

Lời nói đầu

Trong lúc ngồi suy nghĩ chủ đề gì có thể gây đủ ấn tượng với thầy vì ấn tượng ban đầu của em là thầy sẽ khó tính khi chấm bài, em đã nảy ra một ý tưởng. Em là một người cực kỳ thích nghe nhạc và có một vài lần đến lớp em thấy thầy cũng mở nhạc trước giờ học nên em đoán là thầy có sở thích nghe nhạc ạ! Xưa trước khi có Apple Music hay Sportify, thời còn phải tải nhạc về xong lưu và xuất nhạc sang máy nghe nhạc thủ công, em luôn thắc mắc là tại sao Windows lại không có một công cụ để mình có thể chỉnh âm thanh của file nhạc theo ý mình (hoặc là lúc nhỏ em chưa đủ hiểu biết để mò ra tính năng đó của Winxp), mỗi bài hát em đều cảm thấy nó thiếu một cái gì đấy để hoàn hảo, thiếu một chút bass, trebe, ... Rồi đến khi đã có kiến thức về phần này, kiến thức nền tảng về ngành của mình (CNTT) và một chút sự giúp đỡ của Gemini (vì em mới học C++ mà phần này cần thêm Python), em nhận ra mình có thể tự mình làm ra một công cụ của riêng của bản thân mình, từ đó em đã đưa ra quyết định chọn chủ đề này để hoàn thiện bài tập lớn cuối bộ môn! Xây dựng một app EQ để mình chỉnh bất cứ dải nào của bài nhạc mà mình muốn! Đây cũng là lần đầu em viết báo cáo bài tập lớn cho một dự án dài và nhiều dữ kiện, thời gian trải dài như thế này nên không thể tránh khỏi khả năng sẽ có những sai sót, mong thầy em rất mong thầy có thể thông cảm và châm chước. Em rất trân trọng những góp ý và chỉ bảo thêm của thầy để có thể học hỏi và rút kinh nghiệm cho bản thân ạ!

Mục lục

7. Thực hành 3: Xây dựng Ứng dụng GUI Xử lý EQ Offline 5-Band

- 7.1. Mục tiêu và công nghệ (Tkinter, Soundfile)
- 7.2. Quy trình hoạt động của ứng dụng
- 7.3. Hình ảnh giao diện và giải thích
- 7.4. Phân tích kết quả (Spectrogram Trước và Sau khi xử lý)

8. Đánh giá và Kết luận

- 8.1. Đánh giá ưu/nhược điểm
- 8.2. Kết luận
- 8.3. Hướng phát triển

9. Tài liệu tham khảo

10. Phụ lục: Toàn bộ mã nguồn ứng dụng offline_eq_app.py

1. Giới thiệu nội dung

1.1. Đặt vấn đề, lời nói đầu

Trong xử lý tín hiệu âm thanh số, Equalizer (EQ) là một trong những công cụ cơ bản và quan trọng nhất. Từ các hệ thống Hi-Fi, studio thu âm, đến các ứng dụng nghe nhạc, EQ đều đóng vai trò then chốt trong việc định hình chất lượng âm thanh, cho phép người dùng tăng hoặc giảm các dải tần số cụ thể.

Các bộ lọc FIR (Finite Impulse Response), tuy có ưu điểm lớn là đảm bảo pha tuyến tính , nhưng thường yêu cầu bậc lọc rất cao để đạt được đáp ứng tần số sắc nét , dẫn đến chi phí tính toán lớn và độ trễ cao. Đối với các ứng dụng âm thanh như EQ, nơi cần sự linh hoạt và hiệu suất cao, bộ lọc IIR (Infinite Impulse Response) là một lựa chọn tối ưu hơn.

1.2. Mục tiêu của báo cáo

Báo cáo này được thực hiện với mục tiêu nâng cấp và mở rộng kiến thức từ bộ lọc số cơ bản (như bộ lọc FIR trong báo cáo gốc) sang một hệ thống ứng dụng thực tế:

1. **Nghiên cứu** sự khác biệt giữa FIR và IIR, lý giải tại sao IIR là lựa chọn tối ưu cho ứng dụng EQ.
2. **Tìm hiểu sâu** về cấu trúc **Biquad (bậc hai)**, đơn vị cơ bản để xây dựng các bộ lọc IIR hiệu suất cao và ổn định.
3. **Trình bày** các công thức toán học ("Audio EQ Cookbook") để chuyển đổi các tham số EQ (Gain, f_c, Q) thành các hệ số a_k, b_k của bộ lọc Biquad.
4. **Xây dựng** một hệ thống EQ 5-Band hoàn chỉnh, có khả năng xử lý file âm thanh stereo.
5. **Phát triển** một Giao diện đồ họa (GUI) bằng Python (Tkinter) để người dùng có thể tải file nhạc, điều chỉnh 5 dải tần và xuất file kết quả.

1.3. Cấu trúc báo cáo

Báo cáo được chia thành 3 phần chính:

- **Cơ sở lý thuyết (Chương 2, 3, 4):** Trình bày song song lý thuyết về IIR, cấu trúc Biquad, và các công thức thiết kế EQ.
- **Thực hành (Chương 5, 6, 7):** Triển khai các lớp (Class) Python cho bộ lọc Biquad, chuỗi EQ 5-band, và xây dựng ứng dụng GUI hoàn chỉnh. Mô phỏng đáp ứng tần số và áp dụng lọc tệp âm thanh thực tế.
- **Tổng kết (Chương 8):** Đánh giá kết quả, so sánh lý thuyết và đề ra hướng phát triển.

2. Cơ sở lý thuyết 1: Tổng quan về Bộ lọc số và Equalizer

2.1. Khái niệm bộ lọc số

Một hệ thống xử lý tín hiệu số được gọi là một bộ lọc số nếu nó thay đổi đặc tính của tín hiệu đầu vào theo một cách mong muốn. Các khái niệm cơ bản bao gồm:

- **Dải thông (Passband):** Là dải tần số mà bộ lọc cho tín hiệu đi qua.
- **Dải chặn (Stopband):** Là dải tần số mà bộ lọc không cho tín hiệu đi qua.
- **Tần số cắt (f_c):** Tần số giới hạn giữa dải thông và dải chặn.
- **Dải quá độ (Transition band):** Vùng tần số chuyển tiếp giữa dải thông và dải chặn.

2.2. Phân loại bộ lọc số: FIR và IIR

Theo dạng của đặc tính xung $h(n)$, bộ lọc số được chia làm hai loại chính:

1. **Bộ lọc số FIR (Finite Impulse Response):** Có đặc tính xung hữu hạn. Hàm truyền $H(z)$ chỉ bao gồm các số 0 (zeros) và các cực (poles) tại gốc tọa độ.
2. **Bộ lọc số IIR (Infinite Impulse Response):** Có đặc tính xung vô hạn. Hàm truyền $H(z)$ có cả các số 0 và các cực, sử dụng cơ chế phản hồi (feedback).

2.3. So sánh và lựa chọn IIR cho ứng dụng EQ

Đặc tính	Bộ lọc FIR	Bộ lọc IIR
Bậc lọc (N)	Rất cao cho đáp ứng sắc nét	Rất thấp cho đáp ứng tương tự
Pha (Phase)	Có thể đạt pha tuyến tính	Thường là phi tuyến tính (non-linear)
Tính ổn định	Luôn ổn định	Phải kiểm tra (cực trong vòng tròn đơn vị)
Độ trễ (Latency	Cao (tỷ lệ với N)	Rất thấp (tỷ lệ với N)
Hiệu suất	Chi phí tính toán cao	Chi phí tính toán rất thấp

Lựa chọn: Đối với Equalizer âm thanh, yêu cầu quan trọng nhất là khả năng điều chỉnh đáp ứng biên độ một cách linh hoạt và hiệu quả về tính toán. Méo pha là chấp nhận được (tai người ít nhạy cảm với méo pha hơn méo biên độ). Do đó, **bộ lọc IIR là lựa chọn tối ưu** để xây dựng EQ.

2.4. Giới thiệu về Parametric EQ

Parametric EQ (EQ tham số) là loại EQ linh hoạt nhất, cho phép người dùng toàn quyền kiểm soát 3 tham số cho mỗi băng tần:

1. f_c **(Center Frequency):** Tần số trung tâm của bộ lọc (ví dụ: 1000 Hz).
2. G **(Gain):** Lượng tăng (boost, $G > 0$ dB) hoặc giảm (cut, $G < 0$ dB) tại f_c.
3. Q **(Quality Factor):** Độ rộng của băng tần.
    - **Q cao** (ví dụ: Q=10): Băng tần rất hẹp (sắc), ảnh hưởng một vùng tần số nhỏ.
    - **Q thấp** (ví dụ: Q=0.7): Băng tần rất rộng (tù), ảnh hưởng một vùng tần số lớn.

3. Cơ sở lý thuyết 2: Bộ lọc IIR và cấu trúc Biquad

3.1. Hàm truyền của bộ lọc IIR

Bộ lọc IIR có hàm truyền $H(z)$ dạng hàm phân thức, biểu diễn mối quan hệ giữa biến đổi Z của tín hiệu ra $Y(z)$ và tín hiệu vào $X(z)$:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1 + \sum_{k=1}^{N} a_k z^{-k}}$$

Các a_k là các hệ số phản hồi (feedback) và b_k là các hệ số truyền thẳng (feedforward).

3.2. Tính ổn định và cấu trúc xếp tầng (Cascade)

Bộ lọc IIR ổn định khi và chỉ khi tất cả các **cực** (poles) của $H(z)$ (nghiệm của mẫu số) nằm **bên trong** vòng tròn đơn vị trên mặt phẳng Z.

Khi thiết kế bộ lọc IIR bậc cao (N lớn), các cực rất nhạy cảm với sai số lượng tử (do làm tròn số), dễ dẫn đến mất ổn định. Để giải quyết vấn đề này, người ta chia bộ lọc bậc N thành một chuỗi (cascade) các bộ lọc bậc 1 hoặc bậc 2 (Second-Order Sections - SOS).

3.3. Cấu trúc lọc Biquad (Bậc hai - SOS)

Cấu trúc Biquad (viết tắt của "bi-quadratic") là một bộ lọc SOS (IIR bậc 2), có hàm truyền:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

Đây là đơn vị xây dựng cơ bản cho hầu hết các EQ số vì nó:

1. **Rất ổn định:** Dễ dàng kiểm soát vị trí 2 cực của bộ lọc.
2. **Hiệu quả:** Chỉ cần 5 hệ số, 5 phép nhân và 4 phép cộng cho mỗi mẫu.
3. **Linh hoạt:** Hầu hết mọi loại EQ (Peaking, Shelf,...) đều có thể biểu diễn dưới dạng Biquad.

3.4. Phương trình sai phân (Direct Form I)

Từ hàm truyền $H(z) = Y(z) / X(z)$, ta biến đổi ngược về miền thời gian n để có phương trình sai phân, là công thức để lập trình:

$$Y(z)(1 + a_1 z^{-1} + a_2 z^{-2}) = X(z)(b_0 + b_1 z^{-1} + b_2 z^{-2})$$

$$Y(z) = (b_0 X(z) + b_1 z^{-1} X(z) + b_2 z^{-2} X(z)) - (a_1 z^{-1} Y(z) + a_2 z^{-2} Y(z))$$

Biến đổi Z ngược:

$$y[n] = b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2]$$

Tuy nhiên, cấu trúc này không hiệu quả về bộ nhớ đệm. Cấu trúc Direct Form I thường được dùng hơn, bằng cách định nghĩa một tín hiệu trung gian $w[n]$:

$$w[n] = x[n] - a_1 w[n-1] - a_2 w[n-2]$$

$$y[n] = b_0 w[n] + b_1 w[n-1] + b_2 w[n-2]$$

Đây chính là cấu trúc được sử dụng trong phần thực hành. Nó chỉ cần 2 biến trạng thái (bộ đệm) là $w[n-1]$ và $w[n-2]$.

4. Cơ sở lý thuyết 3: Thiết kế hệ số Biquad cho Parametric EQ

4.1. Bài toán: Chuyển đổi tham số EQ sang hệ số Biquad

Đây là mấu chốt của báo cáo. Người dùng cung cấp 4 tham số: F_s (Tần số lấy mẫu), f_0 (Tần số trung tâm), G (Gain, dB), và Q.

Chúng ta cần tìm ra 5 hệ số b_0, b_1, b_2, a_1, a_2 (với $a_0 = 1$) để đưa vào phương trình sai phân.

4.2. Công thức "Audio EQ Cookbook"

Đây là bộ công thức chuẩn mực trong ngành xử lý âm thanh số, được phát triển bởi Robert Bristow-Johnson. Các công thức này dựa trên việc ánh xạ từ bộ lọc analog miền S sang bộ lọc số miền Z thông qua phép biến đổi song tuyến tính (Bilinear Transform), đảm bảo đáp ứng tần số được bảo toàn.

4.3. Các biến trung gian

Trước khi tính 5 hệ số, chúng ta cần tính các biến trung gian từ tham số đầu vào:

1. Gain tuyến tính (A):
    - Cho Peaking EQ: $A = 10^{(G/40)}$

    - Cho Shelf EQ: $A = 10^{(G/20)}$
      (Lý do khác nhau liên quan đến cách A được sử dụng trong công thức)
2. Tần số góc chuẩn hóa (ω_0):
   $\omega_0 = \frac{2\pi f_0}{F_s}$
3. Hệ số α (liên quan đến Q và ω_0):
   $\alpha = \frac{\sin(\omega_0)}{2Q}$
   (Công thức α sẽ khác một chút cho các loại lọc Shelf)

4.4. Công thức cho các loại lọc cơ bản

Khi đã có A, ω_0, α, chúng ta tính các hệ số (chưa chuẩn hóa, $a_0 \neq 1$).

4.4.1. Peaking EQ (Lọc tăng/giảm đỉnh)

- $b_0 = 1 + \alpha A$
- $b_1 = -2\cos(\omega_0)$
- $b_2 = 1 - \alpha A$
- $a_0 = 1 + \alpha / A$
- $a_1 = -2\cos(\omega_0)$
- $a_2 = 1 - \alpha / A$

4.4.2. Low-Shelf Filter (Lọc tăng/giảm bass)

(Với $A = 10^{(G/20)}$ và α được tính khác một chút, liên quan đến A)

- $b_0 = A \cdot ((A+1) - (A-1)\cos(\omega_0) + 2\sqrt{A}\alpha)$
- $b_1 = 2A \cdot ((A-1) - (A+1)\cos(\omega_0))$
- $b_2 = A \cdot ((A+1) - (A-1)\cos(\omega_0) - 2\sqrt{A}\alpha)$
- $a_0 = (A+1) + (A-1)\cos(\omega_0) + 2\sqrt{A}\alpha$
- $a_1 = -2 \cdot ((A-1) + (A+1)\cos(\omega_0))$
- $a_2 = (A+1) + (A-1)\cos(\omega_0) - 2\sqrt{A}\alpha$

4.4.3. High-Shelf Filter (Lọc tăng/giảm treble)

(Tương tự Low-Shelf nhưng công thức có thay đổi về dấu)

- $b_0 = A \cdot ((A+1) + (A-1)\cos(\omega_0) + 2\sqrt{A}\alpha)$
- $b_1 = -2A \cdot ((A-1) + (A+1)\cos(\omega_0))$
- $b_2 = A \cdot ((A+1) + (A-1)\cos(\omega_0) - 2\sqrt{A}\alpha)$
- $a_0 = (A+1) - (A-1)\cos(\omega_0) + 2\sqrt{A}\alpha$
- $a_1 = 2 \cdot ((A-1) - (A+1)\cos(\omega_0))$
- $a_2 = (A+1) - (A-1)\cos(\omega_0) - 2\sqrt{A}\alpha$

Bước cuối cùng: Tất cả các hệ số phải được chuẩn hóa bằng cách chia cho a_0 để đưa về dạng hàm truyền có $a_0 = 1$.

$b'_0 = b_0/a_0 \quad b'_1 = b_1/a_0 \quad b'_2 = b_2/a_0 \quad a'_1 = a_1/a_0 \quad a'_2 = a_2/a_0$

5. Thực hành 1: Xây dựng các khối lọc Biquad cơ bản

5.1. Mã nguồn Python: Lớp (Class) BiquadFilter

Lớp này đóng gói phương trình sai phân (Direct Form I) và quản lý 2 biến trạng thái.

```
class BiquadFilter:
    """
    Lớp triển khai bộ lọc Biquad (IIR bậc 2)
    Sử dụng cấu trúc Direct Form I.
    """
    def __init__(self):
        # Khởi tạo là bộ lọc "pass-through" (không làm gì cả)
        # b0=1, các hệ số khác = 0
        self.b0, self.b1, self.b2 = 1.0, 0.0, 0.0
        self.a1, self.a2 = 0.0, 0.0

        # Khởi tạo các biến trạng thái (bộ đệm trễ)
        self.w_n_1 = 0.0  # w[n-1]
        self.w_n_2 = 0.0  # w[n-2]

    def set_coeffs(self, b0, b1, b2, a1, a2):
        """Hàm để cập nhật hệ số cho bộ lọc"""
        self.b0, self.b1, self.b2 = b0, b1, b2
        self.a1, self.a2 = a1, a2

    def process(self, x_n):
        """
        Lọc một mẫu đầu vào x_n và trả về mẫu đầu ra y_n
        Theo phương trình sai phân:
        w[n] = x[n] - a1*w[n-1] - a2*w[n-2]
        y[n] = b0*w[n] + b1*w[n-1] + b2*w[n-2]
        """

        # Tính w[n]
        w_n = x_n - self.a1 * self.w_n_1 - self.a2 * self.w_n_2

        # Tính y[n]
        y_n = self.b0 * w_n + self.b1 * self.w_n_1 + self.b2 * self.w_n_2

        # Cập nhật trạng thái cho lần lặp kế tiếp
        self.w_n_2 = self.w_n_1
        self.w_n_1 = w_n

        return y_n
```

5.2. Giải thích mã nguồn

- __init__: Khởi tạo bộ lọc ở trạng thái "trong suốt" ($b_0=1$, các hệ số khác bằng 0), để $y[n] = x[n]$. Các biến trạng thái w_n_1 và w_n_2 (tương ứng $w[n-1]$ và $w[n-2]$) được khởi tạo bằng 0.
- set_coeffs: Phương thức này cho phép cập nhật 5 hệ số của bộ lọc (đã được chuẩn hóa $a_0=1$).
- process: Đây là hàm xử lý chính, nhận vào một mẫu $x[n]$ và trả về $y[n]$. Nó thực hiện chính xác hai phương trình sai phân của Direct Form I và cập nhật các biến trạng thái.

5.3. Mô phỏng đáp ứng tần số

Để kiểm tra xem các công thức và lớp BiquadFilter hoạt động chính xác hay không, chúng ta sử dụng hàm scipy.signal.freqz để vẽ đáp ứng tần số của một bộ lọc Peaking EQ.

5.4. Kết quả mô phỏng (Peaking EQ)

Chúng ta mô phỏng một bộ lọc Peaking EQ với các tham số:

- $F_s = 48000$ Hz
- $f_0 = 1000$ Hz
- $G = +9.0$ dB
- $Q = 1.41$

Giải thích: Biểu đồ đáp ứng tần số cho thấy bộ lọc tăng chính xác +9 dB tại tần số trung tâm 1000 Hz. Các tần số xa (bass và treble) có biên độ là 0 dB (không bị ảnh hưởng). Độ rộng của "ngọn núi" được kiểm soát bởi Q. Điều này chứng minh các công thức "Cookbook" và lớp BiquadFilter đã được triển khai chính xác.

6. Thực hành 2: Xây dựng hệ thống EQ đa băng (Cascaded EQ)

6.1. Ý tưởng: Xếp tầng (Cascade) 5 bộ lọc Biquad

Một bộ lọc Biquad chỉ thực hiện được 1 nhiệm vụ (ví dụ: tăng 9dB ở 1000Hz). Một EQ 5-band phức tạp được xây dựng bằng cách **nối tầng (cascade)** 5 bộ lọc Biquad. Tín hiệu đầu vào đi qua bộ lọc 1, đầu ra của bộ lọc 1 trở thành đầu vào của bộ lọc 2, v.v.

$$x[n] \rightarrow [Biquad1] \rightarrow y_1[n] \rightarrow [Biquad2] \rightarrow y_2[n] \rightarrow \cdots \rightarrow [Biquad5] \rightarrow y[n]$$

Đáp ứng tần số tổng cộng $H(e^{j\omega})$ sẽ là tích của các đáp ứng tần số riêng lẻ:

$$H_{total}(e^{j\omega}) = H_1(e^{j\omega}) \cdot H_2(e^{j\omega}) \cdot H_3(e^{j\omega}) \cdot H_4(e^{j\omega}) \cdot H_5(e^{j\omega})$$

Khi biểu diễn bằng Decibel (dB), nó trở thành tổng:

$$H_{total,dB} = H_{1,dB} + H_{2,dB} + H_{3,dB} + H_{4,dB} + H_{5,dB}$$

6.2. Xử lý âm thanh Stereo

Hầu hết các tệp nhạc đều là âm thanh nổi (Stereo), bao gồm 2 kênh: Trái (L) và Phải (R). Để xử lý chính xác, chúng ta phải tạo ra **hai chuỗi EQ 5-band riêng biệt**, một chuỗi cho kênh Trái và một chuỗi cho kênh Phải. Hai chuỗi này có cùng các hệ số a_k, b_k nhưng phải có các biến trạng thái $w[n]$ riêng biệt.

6.3. Mã nguồn Python: Lớp (Class) StereoEQChain_Offline

Lớp này quản lý toàn bộ chuỗi 5-band cho cả hai kênh L/R và cung cấp một hàm process_signal_offline để xử lý toàn bộ tệp âm thanh.

```
import numpy as np
class StereoEQChain_Offline:
    def __init__(self, fs):
        self.fs = fs
        # Định nghĩa 5 dải tần EQ (f0, Q cố định, G=0.0)
        self.bands = [
            {'type': 'lowshelf',  'f0': 100.0,  'Q': 0.7, 'G': 0.0},
            {'type': 'peaking',   'f0': 400.0,  'Q': 1.0, 'G': 0.0},
            {'type': 'peaking',   'f0': 1500.0, 'Q': 1.5, 'G': 0.0},
            {'type': 'peaking',   'f0': 5000.0, 'Q': 2.0, 'G': 0.0},
            {'type': 'highshelf', 'f0': 10000.0, 'Q': 0.7, 'G': 0.0},
        ]

        # Tạo 2 chuỗi lọc (5 Biquads cho L, 5 Biquads cho R)
        self.filters_L = [BiquadFilter() for _ in self.bands]
        self.filters_R = [BiquadFilter() for _ in self.bands]

    def set_gains(self, gain_list_db):
        """Nhận một danh sách 5 giá trị Gain (dB) từ GUI"""
        for i, gain_db in enumerate(gain_list_db):
            self.bands[i]['G'] = gain_db
        self.recalculate_all_coeffs() # Tính lại hệ số khi gain thay đổi

    def calculate_coeffs(self, band):
        """Hàm tính toán hệ số Biquad (như Chương 4)"""
        G, f0, Q, filter_type = band['G'], band['f0'], band['Q'], band['type']

        # (... Toàn bộ công thức "Cookbook" từ Chương 4 ... )
        A = 10**(G / 40.0) if filter_type == 'peaking' else 10**(G / 20.0)
        w0 = 2 * math.pi * f0 / self.fs
        cos_w0 = math.cos(w0); sin_w0 = math.sin(w0)
        alpha = sin_w0 / (2.0 * Q)
        # (if/elif/else cho peaking, lowshelf, highshelf...)
        # (...)
        # return b0/a0, b1/a0, b2/a0, a1/a0, a2/a0 # (Hệ số đã chuẩn hóa)

        # --- Bắt đầu code mẫu (thay bằng công thức thật) ---
        if filter_type == 'peaking':
            alpha = sin_w0 / (2.0 * Q)
            b0 = 1 + alpha * A; b1 = -2 * cos_w0; b2 = 1 - alpha * A
            a0 = 1 + alpha / A; a1 = -2 * cos_w0; a2 = 1 - alpha / A
        elif filter_type == 'lowshelf':
            alpha = sin_w0 / (2 * Q) * (A**0.5)
            b0=A*((A+1)-(A-1)*cos_w0+2*(A**0.5)*alpha); b1=2*A*((A-1)-(A+1)*cos_w0);
b2=A*((A+1)-(A-1)*cos_w0-2*(A**0.5)*alpha)
            a0=(A+1)+(A-1)*cos_w0+2*(A**0.5)*alpha; a1=-2*((A-1)+(A+1)*cos_w0);
a2=(A+1)+(A-1)*cos_w0-2*(A**0.5)*alpha
```

```
        elif filter_type == 'highshelf':
            alpha = sin_w0 / (2 * Q) * (A**0.5)
            b0=A*((A+1)+(A-1)*cos_w0+2*(A**0.5)*alpha); b1=-2*A*((A-1)+(A+1)*cos_w0);
b2=A*((A+1)+(A-1)*cos_w0-2*(A**0.5)*alpha)
            a0=(A+1)-(A-1)*cos_w0+2*(A**0.5)*alpha; a1=2*((A-1)-(A+1)*cos_w0);
a2=(A+1)-(A-1)*cos_w0-2*(A**0.5)*alpha
        else: # Pass-through
            b0, b1, b2, a0, a1, a2 = 1.0, 0.0, 0.0, 1.0, 0.0, 0.0

        if a0 == 0: a0 = 1e-16 # Tránh chia cho 0
        return b0/a0, b1/a0, b2/a0, a1/a0, a2/a0
        # --- Kết thúc code mẫu ---

    def recalculate_all_coeffs(self):
        """Cập nhật hệ số cho tất cả 10 bộ lọc (5 L và 5 R)"""
        for i, band in enumerate(self.bands):
            b0, b1, b2, a1, a2 = self.calculate_coeffs(band)
            self.filters_L[i].set_coeffs(b0, b1, b2, a1, a2)
            self.filters_R[i].set_coeffs(b0, b1, b2, a1, a2)

    def process_signal_offline(self, signal_in):
        """XỬ LÝ TÍN HIỆU SỐ OFFLINE (Trái tim của BTL)"""

        # Đảm bảo tín hiệu đầu vào luôn là 2D (stereo)
        if signal_in.ndim == 1:
            # Nếu là mono, nhân đôi thành stereo
            signal_in = np.stack([signal_in, signal_in], axis=1)

        signal_out = np.zeros_like(signal_in)
        num_samples = len(signal_in)

        # Lặp qua từng mẫu (bám sát lý thuyết phương trình sai phân)
        for i in range(num_samples):
            # Lấy mẫu L/R
            sample_L = signal_in[i, 0]
            sample_R = signal_in[i, 1]

            # Cho qua chuỗi cascade 5-band L
            for filt_L in self.filters_L:
                sample_L = filt_L.process(sample_L)

            # Cho qua chuỗi cascade 5-band R
            for filt_R in self.filters_R:
                sample_R = filt_R.process(sample_R)

            signal_out[i, 0] = sample_L
            signal_out[i, 1] = sample_R

        return signal_out
```

6.4. Giải thích phương thức process_signal_offline
Đây là hàm thực hiện chính của báo cáo.

1. Nó nhận vào một mảng numpy signal_in (toàn bộ file nhạc).
2. Kiểm tra nếu file là 1 kênh (mono), nó tự nhân đôi thành 2 kênh (stereo).
3. Tạo một mảng signal_out rỗng.
4. Lặp qua từng mẫu i từ 0 đến num_samples.
5. Tại mỗi mẫu, nó lấy ra sample_L (kênh Trái) và sample_R (kênh Phải).
6. sample_L được cho đi qua 5 bộ lọc BiquadFilter trong chuỗi filters_L.
7. sample_R được cho đi qua 5 bộ lọc BiquadFilter trong chuỗi filters_R.
8. Kết quả cuối cùng sample_L và sample_R (đã qua 5 lần lọc) được gán vào signal_out[i].
9. Sau khi lặp xong, trả về signal_out (toàn bộ file nhạc đã được lọc).

7. Thực hành 3: Xây dựng Ứng dụng GUI Xử lý EQ Offline 5-Band

Để hiện thực hóa mục tiêu "chỉnh nhạc" một cách trực quan, phần thực hành này xây dựng một ứng dụng hoàn chỉnh với Giao diện đồ họa (GUI).

7.1. Mục tiêu và công nghệ

- **Mục tiêu:** Tạo một giao diện đơn giản cho phép người dùng tải file nhạc, điều chỉnh 5 dải tần EQ, và xuất file kết quả.
- **Công nghệ:**
    - **Tkinter:** Thư viện GUI tiêu chuẩn của Python, dùng để tạo cửa sổ, nút bấm và thanh trượt.
    - **Soundfile:** Thư viện mạnh mẽ để đọc và ghi nhiều định dạng file âm thanh (như .wav, .flac, .ogg) dưới dạng mảng numpy.
    - **Numpy:** Dùng để xử lý các mảng tín hiệu số một cách hiệu quả.

7.2. Quy trình hoạt động của ứng dụng

1. **Nhập liệu (Load):** Người dùng nhấn nút "Import File Nhạc". Ứng dụng dùng filedialog để chọn tệp. soundfile.read() đọc tệp vào mảng numpy (dạng float32) và lấy F_s.
2. **Thiết kế (Adjust):** Giao diện cung cấp 5 thanh trượt (sliders) cho 5 dải tần (Low Shelf, 3 Peaking, High Shelf). Người dùng điều chỉnh Gain (từ -12dB đến +12dB).
3. Xử lý (Process): Khi người dùng nhấn "Process & Export File":
   a. Ứng dụng lấy 5 giá trị Gain (dB) từ GUI.
   b. Khởi tạo đối tượng StereoEQChain_Offline với F_s đã đọc.
   c. Gọi set_gains() để tính toán lại toàn bộ hệ số a_k, b_k cho 10 bộ lọc Biquad.
   d. Gọi process_signal_offline() để lặp qua từng mẫu của tín hiệu gốc, áp dụng chuỗi 5 bộ lọc cascade cho cả kênh Trái và Phải.
   e. Chuẩn hóa (Normalization): Tín hiệu sau xử lý được kiểm tra. Nếu biên độ tối đa vượt quá 1.0 (gây vỡ tiếng), toàn bộ tín hiệu sẽ được chuẩn hóa (chia cho biên độ tối đa) để đảm bảo không bị clipping.
   f. Xuất liệu (Export): Mảng numpy đã xử lý được lưu ra tệp mới bằng soundfile.write().

7.3. Hình ảnh giao diện và giải thích

Giải thích giao diện:

- **1. Import File Nhạc:** Nút để mở cửa sổ chọn tệp.
- **2. Process & Export File:** Nút để bắt đầu xử lý và lưu tệp sau khi đã điều chỉnh.
- **Thanh trạng thái (Status Label):** Hiển thị thông tin tệp đã tải hoặc trạng thái xử lý.
- **5 Thanh trượt:** Mỗi thanh trượt tương ứng với một băng tần EQ, cho phép điều chỉnh Gain từ -12dB (cắt) đến +12dB (tăng). Nhãn dB bên cạnh cập nhật giá trị theo thời gian thực.

(Mã nguồn đầy đủ cho lớp OfflineEQApp được đính kèm trong Phụ lục).

7.4. Phân tích kết quả (Spectrogram Trước và Sau khi xử lý)

Để kiểm chứng hiệu quả, chúng ta sử dụng một tệp âm thanh có cả tiếng ồn (tần số cao) và âm nhạc (tần số thấp). Chúng ta thực hiện một thiết lập EQ "dọn dẹp" (cleanup):

- **Low Shelf:** +3.0 dB (Tăng bass)
- **Mid:** -6.0 dB (Giảm mid để làm rõ)
- **High Shelf:** -9.0 dB (Cắt tiếng ồn tần số cao)

Giải thích Spectrogram (Hình 3):

- **Biểu đồ trên (Trước khi lọc):** Cho thấy năng lượng tín hiệu trải rộng. Vùng tần số cao (ví dụ trên 8kHz) có nhiều năng lượng màu vàng, đây có thể là tiếng ồn (hiss).
- **Biểu đồ dưới (Sau khi lọc):** Cho thấy rõ ràng hiệu quả của EQ.
    1. Vùng tần số thấp (dưới 100Hz) sáng hơn (vàng hơn), thể hiện tác dụng của Low Shelf +3dB.
    2. Vùng tần số trung (khoảng 1500Hz) tối hơn, thể hiện tác dụng của Peaking -6dB.
    3. Vùng tần số cao (trên 10kHz) tối đi rõ rệt (chuyển sang xanh/tím), thể hiện tácDụng của High Shelf -9dB, đã loại bỏ thành công tiếng ồn.

8. Đánh giá và Kết luận

8.1. Đánh giá ưu/nhược điểm

- **Ưu điểm:**
    - Hệ thống EQ IIR Biquad cực kỳ **hiệu quả về tính toán**. Mỗi băng tần chỉ yêu cầu 5 phép nhân cho mỗi mẫu, cho phép xử lý các tệp nhạc dài một cách nhanh chóng.
    - **Linh hoạt:** Cấu trúc lớp (class) cho phép dễ dàng thay đổi f_0, G, Q của 5 dải tần.
    - **Ổn định:** Cấu trúc Biquad xếp tầng đảm bảo tính ổn định số học, tránh các vấn đề của bộ lọc IIR bậc cao.
    - **Trực quan:** Giao diện GUI giúp người dùng không chuyên về kỹ thuật vẫn có thể sử dụng và kiểm chứng kết quả.
- **Nhược điểm:**
    - **Méo pha (Phase Distortion):** Đây là đặc tính cố hữu của bộ lọc IIR. Khi tăng/giảm biên độ, pha của tín hiệu cũng bị thay đổi, đặc biệt là quanh tần số f_0. Tuy nhiên, trong ứng dụng "chỉnh nhạc", sự đánh đổi này là chấp nhận được để đổi lấy hiệu suất.
    - **Thiết lập cố định:** f_0 và Q của 5 dải tần đang được gán cố định trong mã nguồn.

8.2. Kết luận

Báo cáo đã thành công "nâng cấp" từ lý thuyết bộ lọc số cơ bản sang một hệ thống Parametric EQ 5-band phức tạp và có tính ứng dụng cao. Báo cáo đã:

1. Phân biệt rõ ràng ưu nhược điểm của FIR và IIR, lý giải lựa chọn IIR cho EQ.
2. Trình bày và triển khai thành công cấu trúc Biquad (IIR bậc 2) dựa trên các công thức "Audio EQ Cookbook".
3. Xây dựng một hệ thống StereoEQChain_Offline có khả năng xử lý âm thanh stereo bằng phương pháp xếp tầng (cascade) 5 bộ lọc.
4. Hoàn thiện một ứng dụng Giao diện đồ họa (GUI) cho phép người dùng tải, điều chỉnh EQ và xuất tệp âm thanh.
5. Kiểm chứng kết quả bằng phân tích Spectrogram, cho thấy bộ lọc hoạt động chính xác như lý thuyết.

Qua đó, báo cáo đã thể hiện sự hiểu biết sâu sắc từ lý thuyết (phép biến đổi Z, hàm truyền, tính ổn định) đến ứng dụng thực tế (lập trình Python, GUI, xử lý file âm thanh) của môn học Xử lý tín hiệu số.

8.3. Hướng phát triển

- **Parametric đầy đủ:** Nâng cấp GUI để cho phép người dùng điều chỉnh cả f_0 và Q cho mỗi dải tần, thay vì chỉ Gain.

- **Xử lý thời gian thực:** Nâng cấp ứng dụng (sử dụng thư viện sounddevice và threading) để cho phép người dùng nghe (monitor) sự thay đổi của EQ ngay lập tức khi kéo thanh trượt.
- **Vẽ đáp ứng tần số:** Thêm một cửa sổ matplotlib vào GUI để vẽ đáp ứng tần số tổng hợp (tổng dB của 5 dải) và cập nhật khi người dùng kéo thanh trượt.

9. Tài liệu tham khảo

1. Giáo trình Xử lý tín hiệu số - Học viện Công nghệ Bưu chính Viễn thông.
2. Bristow-Johnson, R. (1994). *Audio EQ Cookbook*. Lấy từ http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt.
3. Proakis, J. G., & Manolakis, D. G. (2007). *Digital Signal Processing: Principles, Algorithms, and Applications*.
4. Oppenheim, A. V., & Schafer, R. W. (2009). *Discrete-Time Signal Processing*.
5. Tài liệu thư viện Python: Soundfile, Numpy, Tkinter.

10. Phụ lục: Toàn bộ mã nguồn ứng dụng offline_eq_app.py

```
import tkinter as tk
from tkinter import filedialog, ttk, messagebox
import numpy as np
import soundfile as sf
import os
import math
class BiquadFilter:
    def __init__(self):
        self.b0, self.b1, self.b2 = 1.0, 0.0, 0.0
        self.a1, self.a2 = 0.0, 0.0
        self.w_n_1 = 0.0  # w[n-1]
        self.w_n_2 = 0.0  # w[n-2]

    def set_coeffs(self, b0, b1, b2, a1, a2):
        self.b0, self.b1, self.b2 = b0, b1, b2
        self.a1, self.a2 = a1, a2

    def process(self, x_n):
        w_n = x_n - self.a1 * self.w_n_1 - self.a2 * self.w_n_2
        y_n = self.b0 * w_n + self.b1 * self.w_n_1 + self.b2 * self.w_n_2
        self.w_n_2 = self.w_n_1
        self.w_n_1 = w_n
        return y_n

class StereoEQChain_Offline:
    def __init__(self, fs):
        self.fs = fs
        self.bands = [
            {'type': 'lowshelf',  'f0': 100.0,  'Q': 0.7, 'G': 0.0}, # Bass
            {'type': 'peaking',   'f0': 400.0,  'Q': 1.0, 'G': 0.0}, # Low-Mid
            {'type': 'peaking',   'f0': 1500.0, 'Q': 1.5, 'G': 0.0}, # Mid
            {'type': 'peaking',   'f0': 5000.0, 'Q': 2.0, 'G': 0.0}, # High-Mid
            {'type': 'highshelf', 'f0': 10000.0, 'Q': 0.7, 'G': 0.0}, # Treble
        ]
```

```
        self.filters_L = [BiquadFilter() for _ in self.bands]
        self.filters_R = [BiquadFilter() for _ in self.bands]

    def set_gains(self, gain_list_db):
        for i, gain_db in enumerate(gain_list_db):
            self.bands[i]['G'] = gain_db
        self.recalculate_all_coeffs()

    def calculate_coeffs(self, band):
        G, f0, Q, filter_type = band['G'], band['f0'], band['Q'], band['type']

        A = 10**(G / 40.0) if filter_type == 'peaking' else 10**(G / 20.0)
        w0 = 2 * math.pi * f0 / self.fs
        cos_w0 = math.cos(w0)
        sin_w0 = math.sin(w0)

        if Q <= 0: Q = 1e-16 # Đảm bảo Q > 0
        if self.fs <= 0: return 1.0, 0.0, 0.0, 0.0, 0.0 # An toàn

        if filter_type == 'peaking':
            alpha = sin_w0 / (2.0 * Q)
            b0 = 1 + alpha * A; b1 = -2 * cos_w0; b2 = 1 - alpha * A
            a0 = 1 + alpha / A; a1 = -2 * cos_w0; a2 = 1 - alpha / A

        elif filter_type == 'lowshelf':
            alpha = sin_w0 / (2 * Q) * (A**0.5)
            b0 = A * ((A + 1) - (A - 1) * cos_w0 + 2 * (A**0.5) * alpha)
            b1 = 2 * A * ((A - 1) - (A + 1) * cos_w0)
            b2 = A * ((A + 1) - (A - 1) * cos_w0 - 2 * (A**0.5) * alpha)
            a0 = (A + 1) + (A - 1) * cos_w0 + 2 * (A**0.5) * alpha
            a1 = -2 * ((A - 1) + (A + 1) * cos_w0)
            a2 = (A + 1) + (A - 1) * cos_w0 - 2 * (A**0.5) * alpha

        elif filter_type == 'highshelf':
            alpha = sin_w0 / (2 * Q) * (A**0.5)
            b0 = A * ((A + 1) + (A - 1) * cos_w0 + 2 * (A**0.5) * alpha)
            b1 = -2 * A * ((A - 1) + (A + 1) * cos_w0)
            b2 = A * ((A + 1) + (A - 1) * cos_w0 - 2 * (A**0.5) * alpha)
            a0 = (A + 1) - (A - 1) * cos_w0 + 2 * (A**0.5) * alpha
            a1 = 2 * ((A - 1) - (A + 1) * cos_w0)
            a2 = (A + 1) - (A - 1) * cos_w0 - 2 * (A**0.5) * alpha
        else: # Pass-through
            b0, b1, b2, a0, a1, a2 = 1.0, 0.0, 0.0, 1.0, 0.0, 0.0

        if a0 == 0.0: a0 = 1e-16 # Tránh chia cho 0

        return b0/a0, b1/a0, b2/a0, a1/a0, a2/a0

    def recalculate_all_coeffs(self):
        for i, band in enumerate(self.bands):
            b0, b1, b2, a1, a2 = self.calculate_coeffs(band)
```

```
            self.filters_L[i].set_coeffs(b0, b1, b2, a1, a2)
            self.filters_R[i].set_coeffs(b0, b1, b2, a1, a2)

    def process_signal_offline(self, signal_in):
        if signal_in.ndim == 1:
            signal_in = np.stack([signal_in, signal_in], axis=1)

        signal_out = np.zeros_like(signal_in)
        num_samples = len(signal_in)

        for i in range(num_samples):
            # Lấy mẫu L/R
            sample_L = signal_in[i, 0]
            sample_R = signal_in[i, 1]

            for filt_L in self.filters_L:
                sample_L = filt_L.process(sample_L)

            for filt_R in self.filters_R:
                sample_R = filt_R.process(sample_R)

            signal_out[i, 0] = sample_L
            signal_out[i, 1] = sample_R

        return signal_out


class OfflineEQApp:
    def __init__(self, root):
        self.root = root
        self.root.title("BTL Xử lý tín hiệu số - Made by Feanor")
        self.root.geometry("600x450")

        self.audio_data = None
        self.fs = 0
        self.input_filepath = ""

        # Frame chính
        main_frame = ttk.Frame(root, padding=10)
        main_frame.pack(expand=True, fill=tk.BOTH)

        # 1. Điều khiển File
        file_frame = ttk.LabelFrame(main_frame, text="Điều khiển")
        file_frame.pack(fill=tk.X, padx=5, pady=5)

        self.btn_load = ttk.Button(file_frame, text="1. Import File Nhạc", command=self.load_file)
        self.btn_load.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5, pady=5)

        self.btn_process = ttk.Button(file_frame, text="2. Xử lý & Export File", command=self.process_file,
state=tk.DISABLED)
        self.btn_process.pack(side=tk.LEFT, fill=tk.X, expand=True, padx=5, pady=5)
```

```
        self.status_label = ttk.Label(main_frame, text="Chưa tải file...")
        self.status_label.pack(fill=tk.X, padx=10, pady=5)

        # 2. Bảng điều khiển EQ 5-Band
        eq_frame = ttk.LabelFrame(main_frame, text="Bảng điều khiển EQ 5-Band")
        eq_frame.pack(expand=True, fill=tk.BOTH, padx=5, pady=5)

        self.sliders = []
        self.slider_labels = []

        # Các dải tần (tên, f0)
        bands_info = [
            ("Low Shelf (100Hz)", 0),
            ("Low-Mid (400Hz)", 1),
            ("Mid (1500Hz)", 2),
            ("High-Mid (5000Hz)", 3),
            ("High Shelf (10kHz)", 4)
        ]

        for name, index in bands_info:
            slider_frame = ttk.Frame(eq_frame)
            slider_frame.pack(fill=tk.X, padx=10, pady=10)

            label = ttk.Label(slider_frame, text=f"{name:18s}", width=20)
            label.pack(side=tk.LEFT)

            # Thanh trượt từ -12dB đến +12dB
            slider = ttk.Scale(slider_frame, from_=-12.0, to=12.0, orient=tk.HORIZONTAL,
                               command=lambda v, i=index: self.on_slider_change(i, v))
            slider.set(0.0) # Giá trị mặc định
            slider.pack(side=tk.LEFT, fill=tk.X, expand=True)

            value_label = ttk.Label(slider_frame, text=" 0.0 dB", width=8)
            value_label.pack(side=tk.LEFT)

            self.sliders.append(slider)
            self.slider_labels.append(value_label)

    def on_slider_change(self, index, value):
        gain_db = float(value)
        self.slider_labels[index].config(text=f"{gain_db:6.1f} dB")

    def load_file(self):
        file_path = filedialog.askopenfilename(filetypes=[("Audio Files", "*.wav *.flac *.ogg"), ("All Files",
"*.*")])
        if not file_path:
            return

        try:
            self.audio_data, self.fs = sf.read(file_path, dtype='float32')
            self.input_filepath = file_path
```

```
            self.status_label.config(text=f"Đã tải: {os.path.basename(file_path)} | {self.fs} Hz |
{self.audio_data.shape}")
            self.btn_process.config(state=tk.NORMAL)

        except Exception as e:
            messagebox.showerror("Lỗi", f"Không thể đọc file: {e}")
            self.btn_process.config(state=tk.DISABLED)

    def process_file(self):
        if self.audio_data is None:
            messagebox.showerror("Lỗi", "Chưa tải file nhạc!")
            return

        # Hỏi người dùng nơi lưu file
        save_path = filedialog.asksaveasfilename(
            defaultextension=".wav",
            filetypes=[("WAV File", "*.wav"), ("FLAC File", "*.flac")],
            initialfile=f"{os.path.splitext(os.path.basename(self.input_filepath))[0]}_eq"
        )
        if not save_path:
            return

        gains_db = [slider.get() for slider in self.sliders]

        self.status_label.config(text="Đang xử lý... Vui lòng chờ...")
        self.root.update_idletasks() # Cập nhật GUI ngay lập tức

        try:
            eq = StereoEQChain_Offline(fs=self.fs)
            eq.set_gains(gains_db)
            processed_data = eq.process_signal_offline(self.audio_data)
            max_val = np.max(np.abs(processed_data))
            if max_val > 1.0:
                processed_data = processed_data / max_val
                print(f"Cảnh báo: Tín hiệu bị clipping! Đã tự động chuẩn hóa (Peak: {max_val:.2f})")

            sf.write(save_path, processed_data, self.fs)

            self.status_label.config(text=f"Xong! Đã lưu tại: {save_path}")
            messagebox.showinfo("Thành công", "Đã xử lý và lưu file thành công!")

        except Exception as e:
            messagebox.showerror("Lỗi xử lý", f"Đã xảy ra lỗi: {e}")
            self.status_label.config(text="Xử lý thất bại!")

# Tinh hoa hội tụ ở đây, cuối cùng đã xong
if __name__ == "__main__":
    try:
        import numpy
        import soundfile
    except ImportError:
        print("LỖI: Vui lòng cài đặt các thư viện cần thiết.")
```

```
    print("Chạy lệnh: pip install numpy soundfile")
    exit()

root = tk.Tk()
app = OfflineEQApp(root)
root.mainloop()
```