MỤC LỤC

Chương 6. Lập trình cơ bản phần 2.....5
6.1 Vị trí của chương trong cuốn sách.....5
6.2 Fade và đường bao — chữa tiếng "tách".....8
6.3 Crossfade — nối hai đoạn không cùng nội dung.....15
6.4 Phép trễ.....19
6.5 Đảo thời gian.....25
6.6 Xử lý theo khối cho tệp dài.....27
6.7 Đóng gói thành mô-đun dùng lại được.....32
6.8 Dự án thực hành — giai đoạn 1.....37
6.9 Thực hành, bài tập và tổng kết.....38

DANH MỤC HÌNH VẼ

Hình 6.1 Sáu thao tác theo trục thời gian của chương 6.....6
Hình 6.2 Tiếng "tách" ở chỗ nối, nhìn theo thời gian và theo tần số.....6
Hình 6.3 Ba dạng đường bao, nhìn theo biên độ và theo decibel.....9
Hình 6.4 Chỗ nối trước và sau khi fade — dạng sóng và hiệu hai mẫu liền.....10
Hình 6.5 Hai áp lực ngược chiều khi chọn độ dài fade.....11
Hình 6.6 Bốn khoảng độ dài fade và tình huống tương ứng.....12
Hình 6.7 Ghép có fade giữ nguyên độ dài; crossfade thì không.....13
Hình 6.8 Một cách đặt fade đúng và hai cách sai không báo lỗi.....13
Hình 6.9 Hai đường bao ngược chiều, và mức đo được ở chỗ chuyển tiếp.....16
Hình 6.10 Sai lệch mức ở chỗ chuyển tiếp khi chọn nhầm đường bao.....17
Hình 6.11 Sai lệch độ dài tích luỹ khi nối nhiều đoạn bằng crossfade.....18
Hình 6.12 Trễ 80 ms, giữ nguyên độ dài và không giữ nguyên độ dài.....20
Hình 6.13 Một tiếng gõ và bản có thêm tiếng vọng trễ 250 ms, hệ số 0,4.....21
Hình 6.14 Từ khoảng cách sang độ trễ, và vài mốc quen thuộc.....22
Hình 6.15 Trễ nửa mẫu bằng nội suy tuyến tính và đáp ứng biên độ của nó.....23
Hình 6.16 Bốn lỗi thường gặp quanh phép trễ và cách chặn.....23
Hình 6.17 Một tiếng gõ và bản đảo của nó.....25
Hình 6.18 Phổ biên độ, hiệu hai phổ, và pha của tín hiệu gốc và tín hiệu đảo.....26
Hình 6.19 Bốn chỗ dùng thật của phép đảo thời gian.....26
Hình 6.20 RAM cần cho cả tệp so với RAM cần cho một khối.....27
Hình 6.21 Một tệp 60 giây chia thành 323 khối 8 192 mẫu.....28
Hình 6.22 Lọc theo khối có mang và không mang trạng thái.....29
Hình 6.23 Phân loại các phép theo có trạng thái hay không.....30
Hình 6.24 Trễ và chi phí vòng lặp theo kích thước khối.....31
Hình 6.25 Bốn bản sao của một đoạn mã, và một mô-đun được nhiều kịch bản gọi.....33
Hình 6.26 Một hàm khó dùng lại và một hàm dùng lại được.....33
Hình 6.27 Bốn loại phép kiểm dùng cho mô-đun am.....34
Hình 6.28 Bốn tệp và mười tám hàm của mô-đun am.....35
Hình 6.29 Năm bước của giai đoạn 1.....37
Hình 6.30 Sáu câu tổng kết chương 6.....40

DANH MỤC BẢNG BIỂU

Bảng 6.1 Bốn thứ gần giống mà không thuộc chương này....8
Bảng 6.2 Ba đường bao và chỗ dùng của chúng....10
Bảng 6.3 Độ dài fade theo tình huống....12
Bảng 6.4 Bốn tổ hợp đường bao và nguồn....17
Bảng 6.5 Giữ độ dài hay không....20
Bảng 6.6 Quy đổi giữa mili giây, số mẫu và khoảng cách....24
Bảng 6.7 Chọn kích thước khối theo mục đích....31
Bảng 6.8 Bốn loại phép kiểm và ví dụ trong chương này....34
Bảng 6.9 Kết quả cần có....37
Bảng 6.10 Bốn tiêu chí tự kiểm giai đoạn 1....38
Bảng 6.11 Bài tập của chương 6....39

CHƯƠNG 6
LẬP TRÌNH CƠ BẢN PHẦN 2

Mục tiêu của chương

Sau chương này, người đọc phải làm được những việc sau.

- Giải thích được bằng số vì sao chỗ nối hai đoạn phát ra tiếng "tách", và chữa dứt điểm bằng fade.
- Chọn được dạng đường bao và độ dài fade phù hợp với từng tình huống, và nêu được cái giá của mỗi lựa chọn.
- Viết được crossfade, chọn đúng đường bao theo hai nguồn có tương quan hay không, và tính được độ dài kết quả.
- Viết được phép trễ nguyên mẫu và trễ lẻ mẫu, đổi được giữa mili giây, số mẫu và khoảng cách, và nêu được cái giá của nội suy.
- Đảo được trục thời gian đúng cách, và giải thích được vì sao phổ biên độ không đổi trong khi tai nghe khác hẳn.
- Xử lý được một tệp dài hơn bộ nhớ bằng cách chia khối, và phân biệt được phép có trạng thái với phép không nhớ.
- Mang được trạng thái bộ lọc qua các khối, và chứng minh bằng phép kiểm rằng kết quả bằng đúng khi lọc cả mảng.
- Gói được các hàm đã viết thành mô-đun có bố cục rõ ràng, kèm bốn loại phép kiểm.

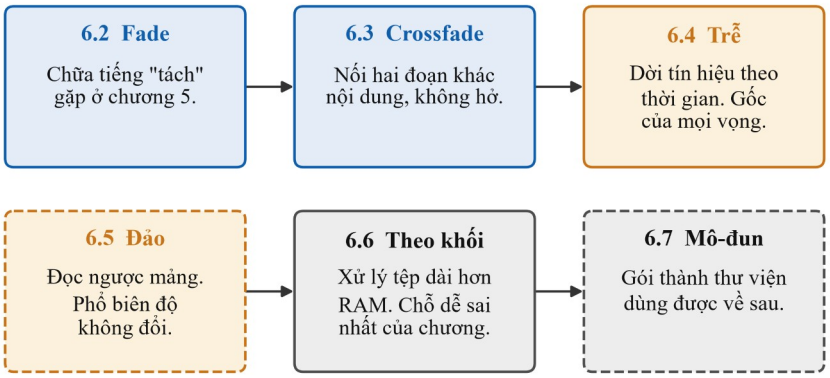
> **Chương này khác chương 5 ở chỗ nào**
>
> Chương 5 làm việc với *biên độ* — giá trị của từng mẫu. Chương 6 làm việc với *trục thời gian* — thứ tự và vị trí của các mẫu.
>
> Mỗi mục của chương này bắt đầu từ một hiện tượng *nghe được* rồi mới đi tới công thức: tiếng tách, lỗ im lặng ở chỗ chuyển tiếp, tiếng vọng, tiếng lách tách đều đặn ở chỗ nối khối.
>
> Mục 6.7 không dạy xử lý tín hiệu mà dạy cách gói mã thành mô-đun. Nó quan trọng ngang các mục kỹ thuật, vì bốn chương tới đều xây trên mô-đun ấy.

6.1 Vị trí của chương trong cuốn sách

6.1.1 Chương khép lại phần lập trình cơ bản

Chương 5 mở phần "Lập trình cơ bản" bằng sáu thao tác trên biên độ: nhân hệ số, cộng hai mảng, cắt lát, chuẩn hoá. Tất cả đều thao tác trên *giá trị* của từng mẫu, và không thao tác nào đụng tới thứ tự các mẫu.

Chương 6 làm phần còn lại: những thao tác trên *trục thời gian*. Dời tín hiệu đi muộn hơn, đọc ngược nó, cho hai đoạn chồng lên nhau, và cắt một tệp dài thành từng khối để xử lý. Hết chương này, phần lập trình cơ bản khép lại và người đọc có đủ công cụ để làm giai đoạn một của dự án thực hành.

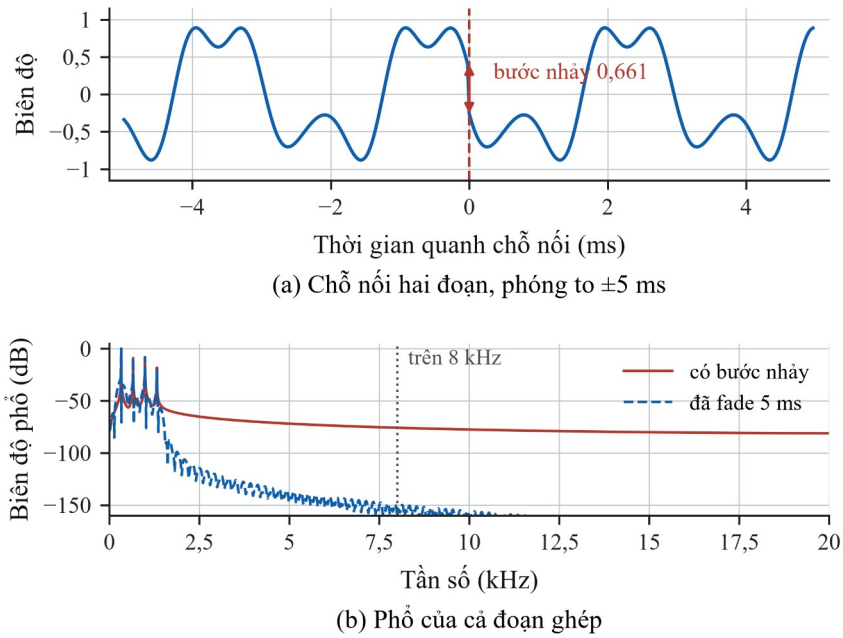


Hình 6.1 Sáu thao tác theo trục thời gian của chương 6

6.1.2 Món nợ để lại từ chương 5

Mục 5.6 kết thúc bằng một câu hỏi bỏ ngỏ. Khi cắt hai đoạn âm thanh rồi nối chúng lại, chỗ nối thường phát ra một tiếng "tách" rất ngắn. Chương 5 đã chỉ ra nguyên nhân trực tiếp — một bước nhảy đột ngột giữa mẫu cuối của đoạn trước và mẫu đầu của đoạn sau — nhưng chưa chữa.

Mục 6.2 chữa nó, và chữa theo cách đo được. Hình dưới đây là số đo của chính món nợ ấy: một chỗ nối giữa hai nốt nhạc, nhìn phóng to trên trục thời gian và nhìn trên trục tần số.



Hình 6.2 Tiếng "tách" ở chỗ nối, nhìn theo thời gian và theo tần số

Điểm đáng chú ý ở hình bên phải: nguồn dùng để dựng hình chỉ có bốn hài, hài cao nhất ở 1,32 kHz. Vậy mà đường màu đỏ vẫn có năng lượng đáng kể ở 10

kHz, 15 kHz, 20 kHz. Năng lượng ấy không có trong tư liệu; nó do chính chỗ nối sinh ra. Sau khi fade 5 mili giây, đường xanh tụt xuống dưới −140 dB ở cùng dải ấy.

> **Một cách đo tiếng tách" bằng số**
>
> Tiếng "tách" là năng lượng cao tần do một bước nhảy sinh ra. Đo phần năng lượng nằm trên 8 kHz so với tổng năng lượng là đo được nó, mà không cần nghe.
>
> Trên ví dụ của hình A02: ghép thẳng cho −44,9 dB; ghép sau khi fade 5 ms cho −128,6 dB. Chênh nhau gần 84 dB.
>
> Đây là đại lượng dùng lại nhiều lần trong chương này. Trong sigtools nó là hàm nang_luong_cao_tan(x, tu_hz=8000).

6.1.3 Cách học chương này

- **Nghe trước, đo sau.** Mỗi mục của chương này bắt đầu bằng một hiện tượng *nghe được* — tiếng tách, lỗ im lặng, tiếng vọng, tiếng rè ở chỗ nối khối. Hãy tạo ra hiện tượng ấy trên máy của mình trước khi đọc phần giải thích.
- **Bốn chương tới sẽ gọi lại các hàm của chương này.** Bộ lọc ở chương 8, hiệu ứng ở chương 10, thời gian thực ở chương 11 đều đứng trên phép trễ và phép xử lý theo khối viết ở đây. Viết cẩu thả bây giờ thì bốn chương sau phải sửa.
- **Mục 6.7 quan trọng ngang các mục kỹ thuật.** Nó không dạy xử lý tín hiệu mà dạy cách gói những gì đã viết thành mô-đun. Không có nó, đến chương 12 bạn sẽ có khoảng ba mươi hàm nằm rải trong hai chục tệp sao chép lẫn nhau, và dự án thực hành không hoàn thành được.

6.1.4 Những gì chương này không làm

Biết ranh giới của một chương cũng có ích ngang biết nội dung của nó. Bốn thứ dưới đây rất gần với nội dung chương 6 và rất hay bị gộp nhầm vào, nhưng đều thuộc về các chương sau.

Bảng 6.1 Bốn thứ gần giống mà không thuộc chương này

Thứ dễ gộp nhầm	Vì sao không thuộc chương 6	Thuộc chương nào
Vọng lặp nhiều lần, hồi âm phòng	Cần vòng phản hồi từ đầu ra về đầu vào — tức là một bộ lọc có phản hồi, chưa học cách viết cho ổn định	Chương 8 và chương 10
Thay đổi độ dài mà không đổi cao độ	Là bài toán khác hẳn: phải phân tích rồi tổng hợp lại theo khung, dùng công cụ của chương 9	Chương 9
Xử lý khi đang phát, không có tệp	Cần vòng lặp gọi lại theo thời gian thực và ràng buộc về thời hạn; xử lý theo khối ở mục 6.6 chỉ là một nửa của nó	Chương 11
Cân độ to giữa nhiều bản thu	Đỉnh và RMS không đủ; cần thước đo độ to LUFS	Chương 7

Điểm chung của cả bốn: chúng đều *xây trên* những gì làm ở chương này. Vọng lặp là phép trễ đặt trong vòng phản hồi. Xử lý thời gian thực là xử lý theo khối cộng thêm ràng buộc thời hạn. Vì thế, học chắc chương 6 làm cho bốn chương kia nhẹ đi rất nhiều.

6.2 Fade và đường bao — chữa tiếng "tách"

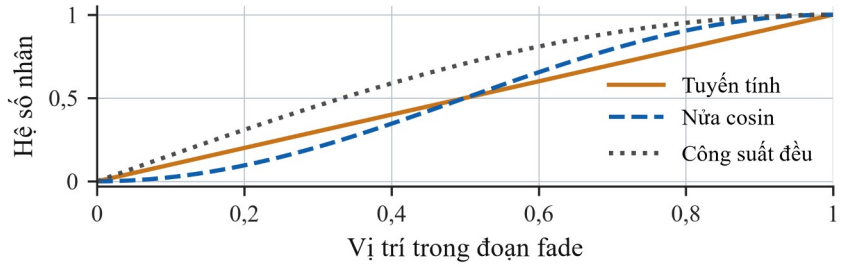
6.2.1 Fade là một phép nhân, không phải một hiệu ứng

Trong phần mềm biên tập, fade hiện ra như một hình tam giác kéo được ở đầu hoặc cuối một đoạn. Trong mã, nó chỉ là phép nhân từng phần tử giữa tín hiệu và một mảng hệ số đi từ 0 lên 1 — mảng ấy gọi là *đường bao*.

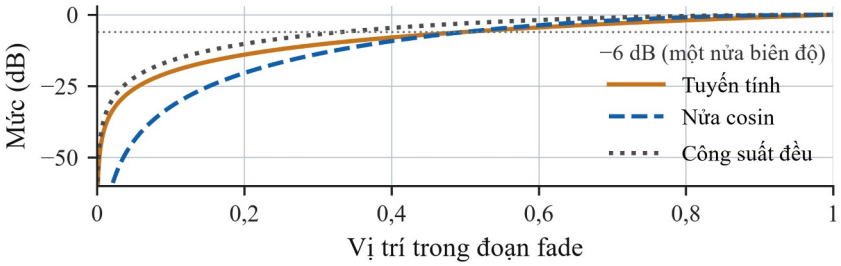
$$y[n] = w[n] \cdot x[n]\,; \qquad w_{\mathrm{tt}}[n] = \frac{n}{N-1}, \quad w_{\cos}[n] = \frac{1}{2}\left[1 - \cos\frac{\pi n}{N-1}\right] \tag{6.1}$$

Trong đó w là đường bao dài N mẫu; w_{tt} là đường bao tuyến tính; w_{cos} là đường bao nửa cosin, còn gọi là nửa cửa sổ Hann. Chỉ số dưới "tt" là tuyến tính, "cos" là nửa cosin.

Cả hai đường bao đều đi từ 0 tới 1 và đều dùng được. Chúng khác nhau ở *đạo hàm* tại hai đầu, và chính đạo hàm ấy quyết định còn nghe thấy tiếng tách hay không.



(a) Theo biên độ



(b) Cũng ba đường ấy, đọc theo dB

Hình 6.3 Ba dạng đường bao, nhìn theo biên độ và theo decibel

Nhìn hình bên trái, đường tuyến tính có vẻ "đều" nhất. Nhìn hình bên phải thì rõ là không: theo thang decibel — thang mà tai người thật sự cảm nhận — đường tuyến tính lao xuống rất nhanh ở đoạn cuối rồi chững lại. Nghe ra thì đó là một fade "hụt" ở cuối.

Đường nửa cosin có đạo hàm bằng không ở cả hai đầu. Nghĩa là ở chỗ đường bao chạm 0 và chỗ nó chạm 1, nó không tạo ra góc nhọn nào trong tín hiệu. Đó là lý do nó là lựa chọn mặc định cho việc chống tiếng tách.

6.2.2 Đường bao thứ ba: công suất đều

Có một dạng đường bao thứ ba, dùng cho một việc khác hẳn. Nó là *căn bậc hai* của đường bao nửa cosin.

$$w_{cs}[n] = \sqrt{w_{cos}[n]}\ ; \qquad w_{cs}^2[n] + w_{cs}^2[N-1-n] = 1 \tag{6.2}$$

Tính chất đáng nói nằm ở dòng thứ hai: bình phương của đường bao tại vị trí n cộng với bình phương của nó tại vị trí đối xứng bằng đúng 1. Bình phương biên độ là công suất, nên câu ấy nói rằng khi dùng đường bao này cho hai nguồn chồng lên nhau, *tổng công suất* luôn giữ nguyên.

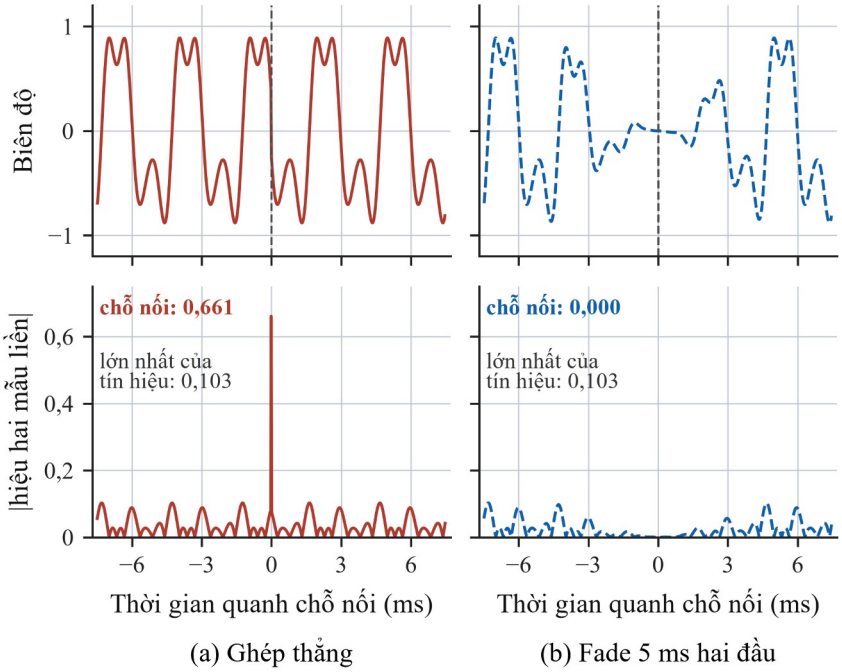
Vì sao điều đó quan trọng — và vì sao dùng nhầm nó lại làm sai chỗ chuyển tiếp — sẽ nói ở mục 6.3. Ở đây chỉ cần ghi nhận: đường bao công suất đều *không* dùng cho fade vào và fade ra thông thường. Nó chỉ dành cho crossfade giữa hai nguồn độc lập.

Bảng 6.2 Ba đường bao và chỗ dùng của chúng

Đường bao	Công thức	Dùng khi nào
Tuyến tính	w = n / (N − 1)	Hầu như không nên dùng cho fade ngắn; chỉ hợp lý cho fade dài vài giây khi muốn mức giảm đều theo biên độ
Nửa cosin	w = ½ · [1 − cos(π n / (N − 1))]	*Mặc định*. Fade vào, fade ra, fade hai đầu khi cắt ghép, và crossfade giữa hai đoạn của cùng một bản thu
Công suất đều	w = căn của nửa cosin	*Chỉ* dùng cho crossfade giữa hai nguồn độc lập — hai bản thu khác nhau, hai bài hát khác nhau

6.2.3 Fade làm gì với tiếng "tách"

Cơ chế rất đơn giản. Nếu mẫu cuối của đoạn A đã được đưa về 0 và mẫu đầu của đoạn B cũng đang ở 0, thì tại chỗ nối không còn bước nhảy nào cả. Bước nhảy biến mất thì năng lượng cao tần do nó sinh ra cũng biến mất theo.



Hình 6.4 Chỗ nối trước và sau khi fade — dạng sóng và hiệu hai mẫu liền

Hàng dưới của hình đo đúng thứ mà tai nghe thấy: mức thay đổi giữa hai mẫu liền nhau. Ở bản ghép thẳng, chỗ nối để lại một gai 0,660, gấp sáu lần biến thiên lớn nhất của chính tín hiệu (0,110). Sau khi fade, bước nhảy tại chỗ nối xuống *đúng bằng không*.

Vì sao con số là đúng bằng không" chứ không phải rất nhỏ"

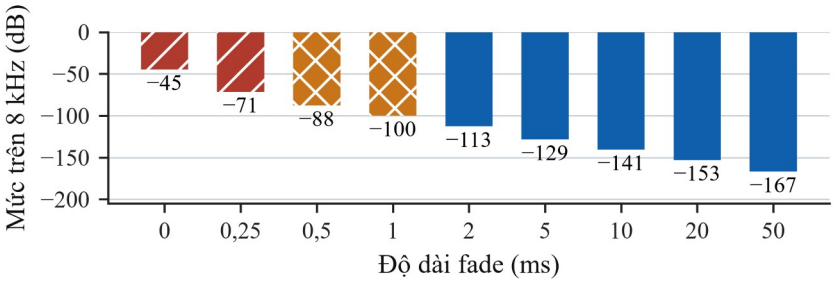
Đường bao nửa cosin có giá trị đầu tiên đúng bằng 0 và giá trị cuối cùng đúng bằng 1. Nhân với 0 thì ra 0 — không có sai số làm tròn nào ở đây.

Vì thế mẫu cuối của A sau khi fade ra bằng 0, mẫu đầu của B sau khi fade vào cũng bằng 0, và hiệu của chúng bằng 0 chính xác.

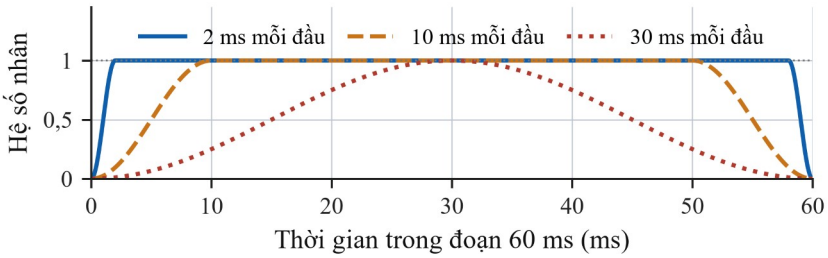
Đây là một trong số ít chỗ trong cuốn sách mà kết quả kiểm thử là một dấu bằng tuyệt đối chứ không phải một ngưỡng dung sai.

6.2.4 Chọn độ dài fade

Câu hỏi thực tế duy nhất của mục này là: fade bao nhiêu mili giây. Có hai áp lực kéo ngược chiều nhau.



(a) Fade càng dài, tiếng tách càng nhỏ



(b) Đường bao trên một đoạn chỉ dài 60 ms

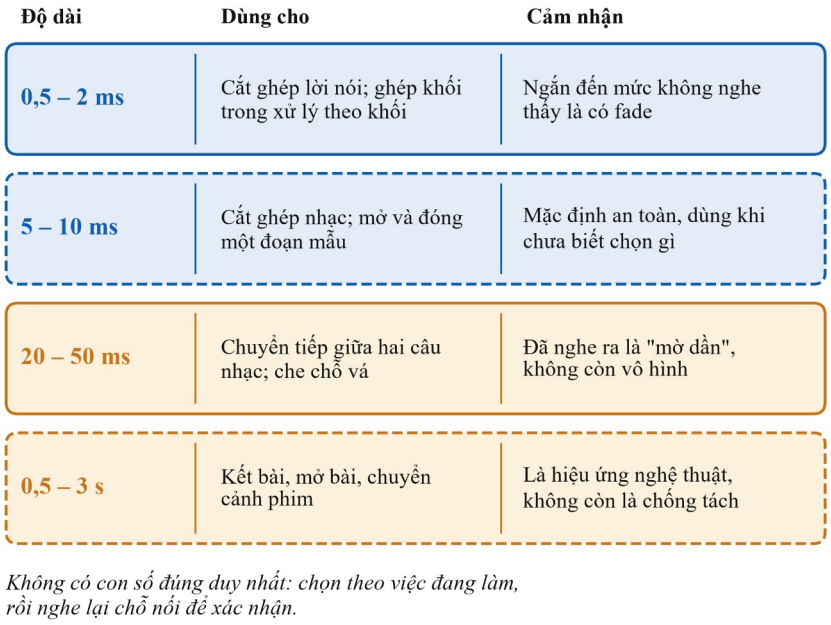
Hình 6.5 Hai áp lực ngược chiều khi chọn độ dài fade

Áp lực thứ nhất đẩy về phía dài. Fade càng dài thì bước nhảy càng được trải ra trên nhiều mẫu, và năng lượng cao tần càng nhỏ. Từ không fade tới fade 5 mili giây, con số đi từ −45 dB xuống −129 dB.

Áp lực thứ hai đẩy về phía ngắn. Fade lấn vào chính nội dung. Trên một đoạn chỉ dài 60 mili giây — độ dài rất thường gặp khi cắt từng âm tiết hoặc từng tiếng trống — fade 30 mili giây mỗi đầu đã chiếm trọn đoạn: không còn mẫu nào giữ nguyên biên độ, và phần đầu tiếng, chỗ tai nhận ra nhạc cụ, bị làm nhỏ đi.

Bảng 6.3 Độ dài fade theo tình huống

Độ dài	Dùng cho	Người nghe cảm nhận thế nào
0,5 – 2 ms	Cắt ghép lời nói; ghép hai khối trong xử lý theo khối	Không nghe thấy là có fade — chỉ thấy hết tiếng tách
5 – 10 ms	Cắt ghép nhạc; mở và đóng một đoạn mẫu	Vẫn không nghe thấy. Đây là mặc định an toàn
20 – 50 ms	Chuyển tiếp giữa hai câu nhạc; che một chỗ vá	Bắt đầu nghe ra là "mờ dần", không còn vô hình
0,5 – 3 s	Kết bài, mở bài, chuyển cảnh phim	Là một hiệu ứng nghệ thuật có chủ ý

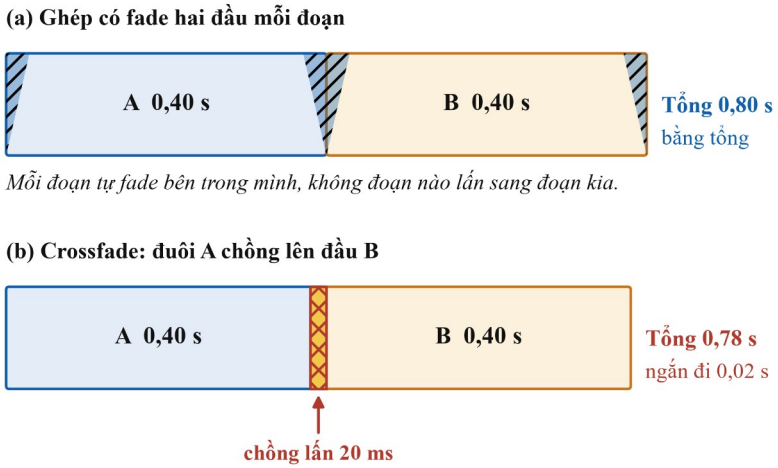


Hình 6.6 Bốn khoảng độ dài fade và tình huống tương ứng

Quy ước dùng cho cả cuốn sách: khi chưa có lý do gì khác, fade 5 mili giây. Con số ấy tương ứng 220 mẫu ở tần số lấy mẫu 44,1 kHz, đủ ngắn để không nghe thấy và đủ dài để dập hết bước nhảy.

6.2.5 Fade không làm đổi độ dài

Đây là điểm dễ nhầm nhất của cả mục, và nhầm ở đây thì lệch nhịp toàn bản dựng.



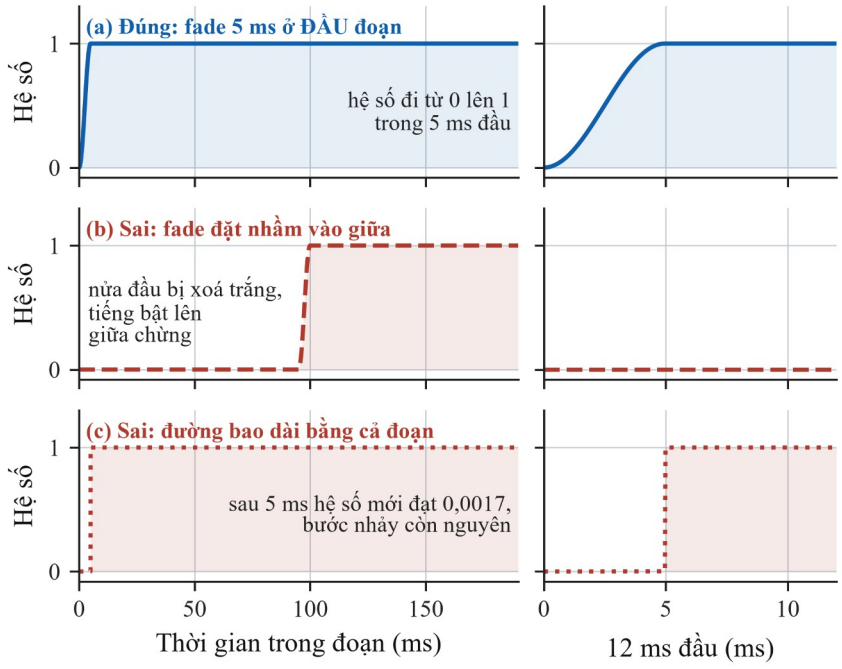
Hình 6.7 Ghép có fade giữ nguyên độ dài; crossfade thì không

Fade nhân đường bao vào các mẫu đã có sẵn của đoạn. Không mẫu nào bị thêm vào, không mẫu nào bị bỏ đi. Ghép ba đoạn 0,40 giây đã fade thì được đúng 1,20 giây.

Crossfade thì khác hẳn: nó cho đuôi đoạn trước và đầu đoạn sau *chồng* lên nhau, nên kết quả ngắn đi đúng bằng phần chồng lấn ở mỗi chỗ nối. Mục 6.3 nói kỹ chuyện này.

6.2.6 Ba cách đặt fade sai mà máy không báo lỗi

Cả ba đoạn mã dưới đây đều chạy trót lọt, không sinh ra ngoại lệ nào, và đều ghi ra một tệp WAV mở được. Chỉ khi nghe hoặc khi vẽ đường bao ra mới biết là sai.



Hình 6.8 Một cách đặt fade đúng và hai cách sai không báo lỗi

– **Đặt nhầm vị trí.** Nhân đường bao vào giữa mảng thay vì vào đầu. Kết quả là nửa đầu bị xoá trắng và tiếng bật lên giữa chừng — nghe ra ngay, nhưng nếu chỉ nhìn tệp kết quả trong trình quản lý tệp thì không thấy gì bất thường.

– **Dùng đường bao dài bằng cả đoạn.** Sinh đường bao dài bằng độ dài mảng rồi chỉ lấy 220 phần tử đầu của nó. Sau 5 mili giây, hệ số mới đạt 0,0017 — nghĩa là 220 mẫu đầu gần như bị xoá, còn bước nhảy thì vẫn nguyên ở mẫu thứ 221. Vẫn còn tiếng tách.
– **Fade một đầu mà bỏ sót đầu kia.** Fade ra cho đoạn A nhưng bỏ sót fade vào cho đoạn B. Bước nhảy giảm đi khoảng một nửa nên tiếng tách nhỏ hơn — vừa đủ để người viết tưởng là đã xong, mà vẫn còn nghe được trên tai nghe tốt.

> **Cách tự kiểm trong ba mươi giây**
> Sau khi fade, in ra ba con số: mẫu đầu, mẫu cuối, và hiệu tuyệt đối lớn nhất giữa hai mẫu liền nhau ở lân cận chỗ nối.
> Nếu fade đúng thì mẫu đầu và mẫu cuối bằng 0, và hiệu tại đúng chỗ nối bằng 0.
> Ba dòng print ấy bắt được cả ba lỗi trên. Không cần vẽ hình, không cần nghe.

6.2.7 Mã

Liệt kê 6.1 Đường bao fade và ba hàm dùng nó

```
import numpy as np

def duong_bao_fade(n, kieu="cosin", len_dan=True):
    """Đường bao dài n mẫu, đi lên nếu len_dan=True."""
    if n <= 0:  return np.zeros(0)
    if n == 1:  return np.ones(1)
    u = np.linspace(0.0, 1.0, n)
    if kieu == "tuyen_tinh":
        w = u
    elif kieu == "cong_suat":
        w = np.sqrt(0.5 * (1.0 - np.cos(np.pi * u)))
    else:                       # "cosin" – mặc định
        w = 0.5 * (1.0 - np.cos(np.pi * u))
    return w if len_dan else w[::-1]


def fade_vao(x, ms=5.0, sr=44100, kieu="cosin"):
    a = np.array(x, dtype=float, copy=True)
    n = min(int(round(ms / 1000 * sr)), a.shape[0])
    if n <= 0:  return a
    w = duong_bao_fade(n, kieu, True)
    a[:n] *= w if a.ndim == 1 else w[:, None]
    return a


def fade_ra(x, ms=5.0, sr=44100, kieu="cosin"):
    a = np.array(x, dtype=float, copy=True)
    n = min(int(round(ms / 1000 * sr)), a.shape[0])
    if n <= 0:  return a
    w = duong_bao_fade(n, kieu, False)
    a[-n:] *= w if a.ndim == 1 else w[:, None]
    return a


def fade_hai_dau(x, ms=5.0, sr=44100, kieu="cosin"):
    return fade_ra(fade_vao(x, ms, sr, kieu), ms, sr, kieu)
```

Ba chi tiết nhỏ trong đoạn mã trên đáng chú ý, vì cả ba đều là thói quen nên giữ cho mọi hàm viết sau này.

- **copy=True.** Hàm không sửa mảng của người gọi. Nếu bỏ tham số này, một dòng fade_vao(x) sẽ âm thầm làm thay đổi chính x, và lần chạy thứ hai cho kết quả khác lần thứ nhất.
- **min(..., a.shape[0]).** Chặn trường hợp fade dài hơn cả đoạn. Không có nó thì một đoạn 3 mili giây với fade 5 mili giây sẽ sinh lỗi chỉ số.
- **w[:, None] cho mảng hai chiều.** Đường bao là mảng một chiều còn tín hiệu stereo là mảng hai chiều. Thêm một trục để numpy nhân đúng đường bao ấy vào cả hai kênh.

Liệt kê 6.2 Ghép nhiều đoạn mà không có tiếng tách

```
def ghep_muot(doan, ms=5.0, sr=44100, kieu="cosin"):
    """Fade hai đầu MỖI đoạn rồi mới nối. Không đổi độ dài."""
    return np.concatenate(
        [fade_hai_dau(x, ms, sr, kieu) for x in doan], axis=0)


# kiểm ngay tại chỗ:
a, b = doan[0], doan[1]
truoc = abs(b[0] - a[-1])
fa, fb = fade_hai_dau(a), fade_hai_dau(b)
sau = abs(fb[0] - fa[-1])
print(f"bước nhảy: {truoc:.3f} -> {sau:.3f}")
# bước nhảy: 0.660 -> 0.000
```

6.3 Crossfade — nối hai đoạn không cùng nội dung

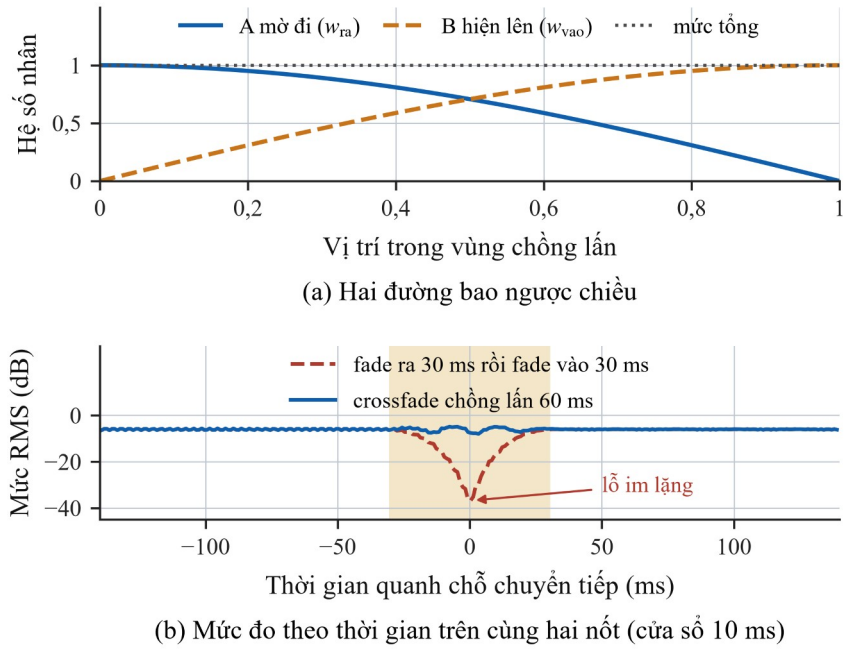
6.3.1 Khi fade hai đầu không đủ

Cách ghép ở mục 6.2 giải quyết dứt điểm tiếng tách, nhưng nó có một cái giá mà trên tư liệu ngắn thì không thấy, trên tư liệu dài thì thấy ngay: ở chỗ nối, âm thanh tụt về không rồi mới lên lại. Với fade 5 mili giây thì cái hõm ấy quá ngắn để nghe ra. Với fade 30 mili giây thì nó thành một lỗ im lặng rõ rệt.

Crossfade giải quyết bằng cách khác: cho đuôi đoạn trước và đầu đoạn sau *chồng* lên nhau. Trong vùng chồng lấn, cả hai đoạn cùng phát — một đoạn nhỏ dần, đoạn kia lớn dần — nên mức tổng không bao giờ tụt về không.

$$y[n] = a[n]\,w_{\text{ra}}[n] + b[n]\,w_{\text{vào}}[n]\,; \qquad N_{\text{ra}} = N_a + N_b - N_{\text{chồng}} \tag{6.3}$$

Dòng thứ hai là điều phải nhớ: độ dài kết quả bằng tổng hai độ dài *trừ đi* phần chồng lấn. N_{ra} là số mẫu kết quả, N_a và N_b là số mẫu của hai đoạn, N_{chong} là số mẫu chồng lấn.



Hình 6.9 Hai đường bao ngược chiều, và mức đo được ở chỗ chuyển tiếp

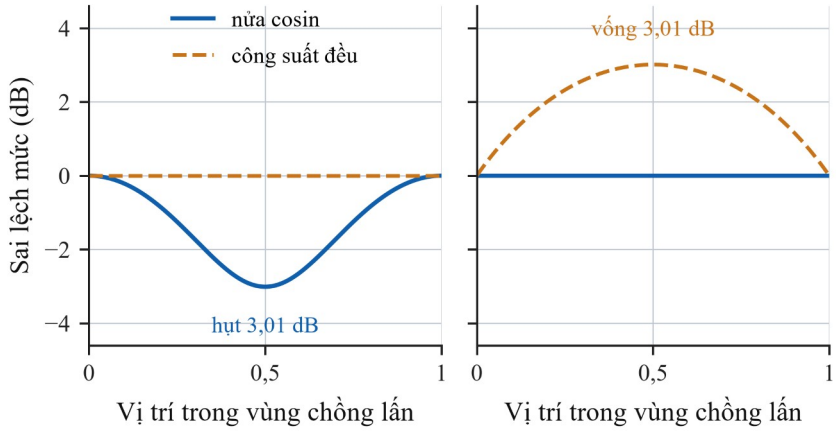
Hình bên phải là bằng chứng. Đường đỏ là cách ghép "fade ra rồi fade vào" với 30 mili giây mỗi bên: mức RMS tụt xuống dưới −35 dB ở giữa. Đường xanh là crossfade chồng lấn 60 mili giây trên cùng hai nốt ấy: mức đi ngang qua chỗ chuyển tiếp gần như không đổi.

6.3.2 Chọn đúng đường bao — ba decibel đi đâu mất

Đây là chỗ mà đường bao công suất đều ở mục 6.2.2 phát huy tác dụng, và cũng là chỗ chọn nhầm thì sai.

Trong vùng chồng lấn, hai tín hiệu cộng vào nhau. Cộng như thế nào thì phụ thuộc vào chúng có *tương quan* với nhau hay không — chính là câu chuyện đã học ở mục 5.5 khi bàn về trộn.

- **Hai nguồn độc lập** — hai bản thu khác nhau, hai bài hát khác nhau. Biên độ cộng theo căn bậc hai của tổng bình phương. Muốn mức tổng không đổi thì hai đường bao phải thoả $w^2 + v^2 = 1$, tức là đường bao *công suất đều*.
- **Hai nguồn tương quan** — hai lát cắt của cùng một bản thu, hoặc cùng một đoạn được xử lý hai kiểu. Biên độ cộng thẳng. Muốn mức tổng không đổi thì phải thoả $w + v = 1$, tức là đường bao *nửa cosin*.



(a) Hai nguồn độc lập (b) Hai nguồn tương quan

Hình 6.10 Sai lệch mức ở chỗ chuyển tiếp khi chọn nhầm đường bao

Chọn nhầm không làm chương trình báo lỗi. Nó chỉ làm mức ở giữa chỗ chuyển tiếp lệch đi đúng 3,01 decibel — hụt nếu dùng nửa cosin cho hai nguồn độc lập, vống nếu dùng công suất đều cho hai nguồn tương quan.

Bảng 6.4 Bốn tổ hợp đường bao và nguồn

Nguồn	Nửa cosin	Công suất đều
Độc lập (hai bản thu khác nhau)	*Hụt* 3,01 dB ở giữa	Đúng — mức không đổi
Tương quan (cùng một bản thu)	Đúng — mức không đổi	*Vống* 3,01 dB ở giữa

Ba decibel có đáng bận tâm không

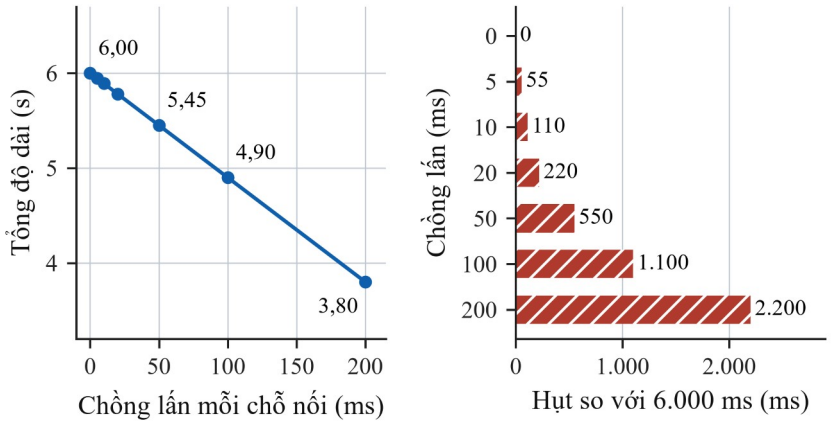
Ba decibel là ngưỡng mà phần lớn người nghe bắt đầu nhận ra có thay đổi về độ to. Một chỗ chuyển tiếp lẻ thì có thể bỏ qua.

Nhưng một bản dựng phát thanh có hàng chục chỗ chuyển tiếp. Khi tất cả đều hụt 3 dB ở giữa, người nghe không nói được là "chỗ nối bị sai" — họ chỉ thấy bản dựng "nghe không chắc tay".

Quy tắc thực dụng: nếu hai đoạn là hai tệp khác nhau thì dùng công suất đều; nếu là hai lát của cùng một tệp thì dùng nửa cosin. Trong sigtools, tham số kieu mặc định của hàm crossfade là "cong_suat" đúng vì trường hợp đầu phổ biến hơn.

6.3.3 Crossfade rút ngắn bản dựng

Vì hai đoạn chồng lên nhau, mỗi chỗ nối làm mất đúng phần chồng lấn. Trên một chỗ nối thì không đáng kể; trên một bản dựng nhiều chỗ nối thì tích luỹ lại thành sai lệch thấy được.



(a) Độ dài bản dựng (b) Sai lệch tích luỹ

Hình 6.11 Sai lệch độ dài tích luỹ khi nối nhiều đoạn bằng crossfade

Mười hai đoạn nửa giây, nối bằng crossfade 200 mili giây, cho ra 3,80 giây chứ không phải 6,00 giây. Ở nhịp 120 nhịp một phút, 2,2 giây hụt là hơn bốn phách — đủ để cả bản dựng lệch khỏi nhạc nền.

Cách xử lý là tính ngược ngay từ đầu: nếu cần K đoạn ghép lại thành đúng một độ dài định trước, hãy cắt mỗi đoạn dài thêm đúng phần sẽ mất. Với K đoạn thì có K − 1 chỗ nối, nên tổng phần mất là (K − 1) nhân với độ dài chồng lấn.

6.3.4 Mã

Liệt kê 6.3 Crossfade hai đoạn

```
def crossfade(a, b, ms=20.0, sr=44100, kieu="cong_suat"):
    """Đuôi a mờ đi trong khi đầu b hiện lên.
    Kết quả NGẮN hơn tổng hai độ dài đúng bằng phần chồng lấn."""
    a = np.asarray(a, dtype=float)
    b = np.asarray(b, dtype=float)
    n = min(int(round(ms / 1000 * sr)), a.shape[0], b.shape[0])
    if n <= 0:
        return np.concatenate([a, b], axis=0)
    xuong = duong_bao_fade(n, kieu, False)
    len_  = duong_bao_fade(n, kieu, True)
    if a.ndim == 2:
        xuong, len_ = xuong[:, None], len_[:, None]
    giua = a[-n:] * xuong + b[:n] * len_
    return np.concatenate([a[:-n], giua, b[n:]], axis=0)


# kiểm độ dài — phép kiểm bắt buộc cho hàm này:
y = crossfade(a, b, ms=20.0)
assert len(y) == len(a) + len(b) - int(round(0.020 * 44100))
```

Dòng assert cuối cùng không phải để cho đẹp. Nó là phép kiểm bắt được chính cái lỗi hay gặp nhất của hàm này: bỏ sót việc trừ phần chồng lấn khi tính vị trí các đoạn tiếp theo trong bản dựng.

6.3.5 Khi nào không dùng crossfade

Crossfade nghe êm hơn nên rất dễ thành thói quen dùng ở mọi chỗ nối. Có ba tình huống mà nó là lựa chọn sai.

- **Khi độ dài phải đúng tuyệt đối.** Dựng theo nhịp, lồng tiếng khớp hình, hoặc ghép các khối trong xử lý theo khối. Ở những chỗ này, ghép có fade giữ nguyên độ dài mới là thứ cần.
- **Khi hai đoạn cùng nội dung ở vùng chồng lấn.** Nếu đuôi A và đầu B là hai bản của cùng một tiếng, chúng gần như chắc chắn lệch pha với nhau. Cộng hai bản lệch pha của cùng một tiếng cho ra hiện tượng lược tần số: âm nghe rỗng đi trong đúng vùng chồng lấn. Đây là hiện tượng đã gặp ở mục 5.5 khi trộn hai nguồn tương quan.
- **Khi vùng chồng lấn rơi vào một tiếng gõ.** Đầu tiếng của trống hay của một nốt đàn rất ngắn và rất mạnh. Cho nó mờ dần trong 50 mili giây thì nó thành một tiếng "thụt". Chuyển tiếp nên đặt vào chỗ *ít* nội dung nhất, không phải chỗ tiện nhất.

> **Cách chọn chỗ đặt chuyển tiếp**
>
> Nhìn dạng sóng, tìm chỗ biên độ nhỏ nhất trong vài trăm mili giây quanh điểm định nối. Đặt vùng chồng lấn vào đấy.
>
> Với lời nói, chỗ ấy thường là khoảng lặng giữa hai từ. Với nhạc, thường là ngay *trước* một phách chứ không phải ngay sau.
>
> Khi đã quen, việc này làm bằng mắt trong vài giây. Đó là một trong ít chỗ của cuốn sách mà nhìn dạng sóng nhanh hơn tính toán.

6.4 Phép trễ

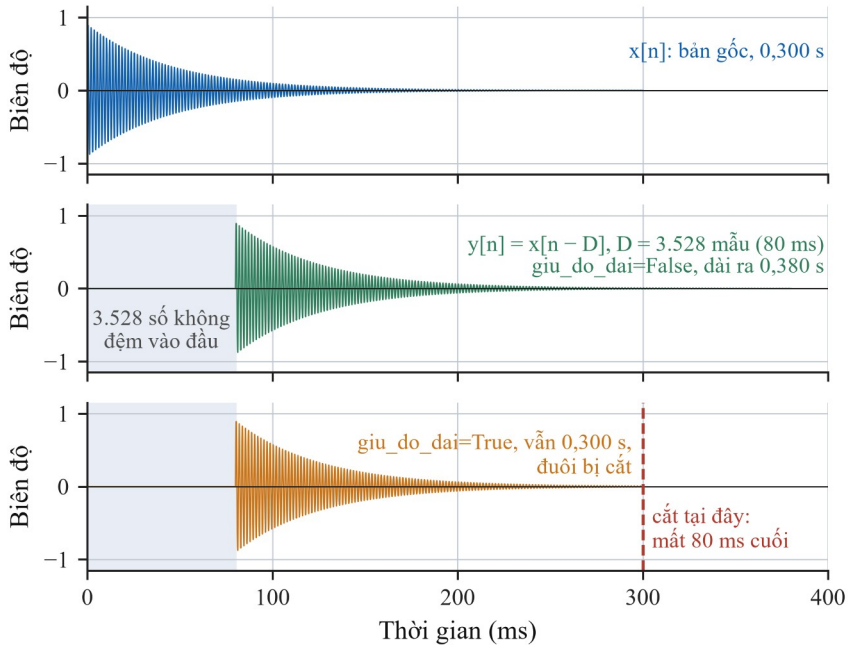
6.4.1 Trễ là đệm số không vào đầu mảng

Trễ một tín hiệu nghĩa là làm cho nó đến muộn hơn. Trên một mảng số, "muộn hơn" chỉ có nghĩa là dịch mọi mẫu sang phải, và chỗ trống bên trái được lấp bằng số không.

$$y[n] = x[n - D], \qquad D = \left\lfloor \frac{t_{\text{ms}}}{1000} \cdot f_{\text{s}} \right\rfloor \tag{6.4}$$

D là số mẫu trễ, t_{ms} là độ trễ tính bằng mili giây, f_s là tần số lấy mẫu. Dấu ngoặc vuông trên dưới lệch nhau là phép làm tròn về số nguyên gần nhất — vì trễ trên một mảng số chỉ nhận giá trị nguyên mẫu; mục 6.4.4 nói về phần lẻ.

Ở tần số lấy mẫu 44,1 kHz, trễ 20 mili giây là 882 mẫu, trễ 250 mili giây là 11 025 mẫu, trễ 80 mili giây là 3 528 mẫu.



Hình 6.12 Trễ 80 ms, giữ nguyên độ dài và không giữ nguyên độ dài

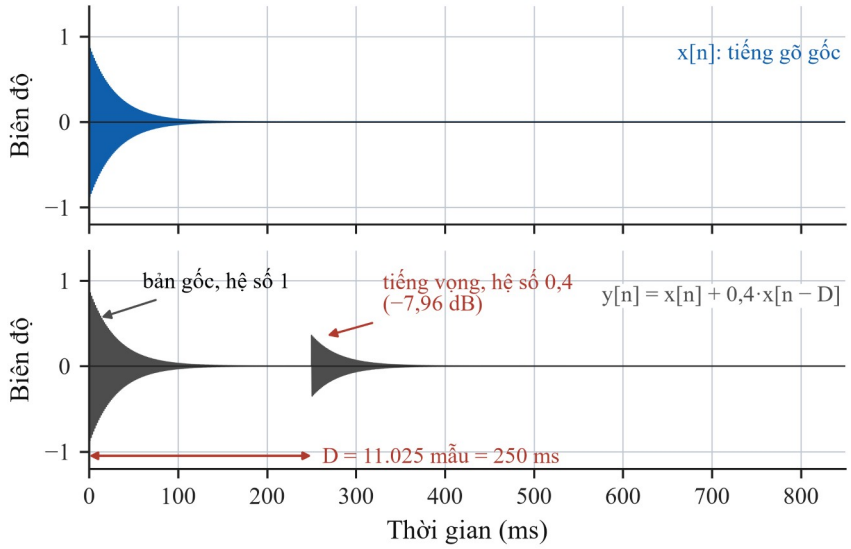
Câu hỏi phải quyết định mỗi lần gọi hàm trễ là có giữ nguyên độ dài mảng hay không. Hai lựa chọn đều đúng, tuỳ việc.

Bảng 6.5 Giữ độ dài hay không

Lựa chọn	Được gì	Mất gì
Giữ nguyên độ dài (cắt đuôi)	Mảng vẫn khớp với các mảng khác trong cùng bản dựng, cộng được ngay mà không phải đệm	Mất đúng D mẫu ở cuối — nếu đuôi tiếng còn vang thì bị cụt
Để mảng dài ra	Không mất mẫu nào; đúng về mặt tín hiệu	Mọi phép cộng sau đó phải đệm mảng ngắn cho bằng, nếu không sẽ lỗi hình dạng

6.4.2 Tiếng vọng đơn

Ứng dụng đầu tiên và trực tiếp nhất của phép trễ: trễ tín hiệu, giảm biên độ, rồi cộng lại với bản gốc. Kết quả là một tiếng vọng.



Hình 6.13 Một tiếng gõ và bản có thêm tiếng vọng trễ 250 ms, hệ số 0,4

Hệ số 0,4 tương ứng −7,96 decibel, nên tiếng vọng nghe rõ là nhỏ hơn tiếng gốc. Ba dòng mã là đủ, và không có khái niệm nào mới so với chương 5: chỉ có một phép trễ và một phép trộn.

Liệt kê 6.4 Tiếng vọng đơn

```
def echo_don(x, ms=250.0, g=0.4, sr=44100):
    """Một tiếng vọng duy nhất."""
    tre = tre_ms(x, ms, sr)          # dài hơn x đúng D mẫu
    return tron(x, g * tre)          # tron() tự đệm cho bằng
```

Đây là hạt nhân của chương 8

Nếu thay vì cộng tiếng vọng vào một lần, chúng ta đưa *đầu ra* vòng ngược trở lại đầu vào, thì mỗi vòng lại sinh thêm một tiếng vọng nhỏ hơn. Đó là bộ vọng lặp, và cũng chính là dạng đơn giản nhất của bộ lọc IIR đã gặp ở mục 4.6.

Nói cách khác: phép trễ viết ở đây là viên gạch của tất cả những gì sẽ làm ở chương 8 (bộ lọc) và chương 10 (hiệu ứng).

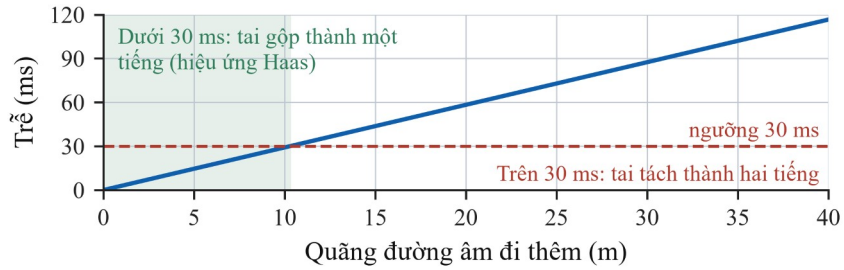
Vì thế nên viết hàm trễ cho gọn và có kiểm thử ngay từ bây giờ.

6.4.3 Trễ và khoảng cách

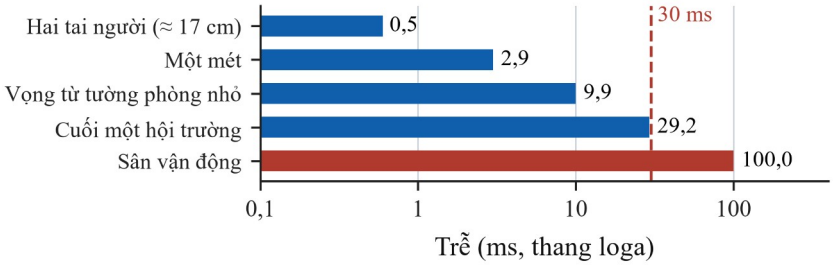
Âm thanh trong không khí đi được khoảng 343 mét mỗi giây ở nhiệt độ phòng. Con số ấy biến mọi khoảng cách trong một căn phòng thành một độ trễ tính được.

$$t_{\text{ms}} = \frac{d}{c} \cdot 1000, \qquad c = 343 \text{ m/s} \tag{6.5}$$

Trong đó d là quãng đường tính bằng mét, c là tốc độ âm thanh, và t_{ms} là độ trễ tính bằng mili giây. Một mét là 2,9 mili giây; mười mét là 29,2 mili giây.



(a) Trễ theo quãng đường, c = 343 m/s



(b) Vài mốc quen thuộc

Hình 6.14 Từ khoảng cách sang độ trễ, và vài mốc quen thuộc

Ngưỡng 30 mili giây trên hình là một mốc tâm lý âm học quan trọng, gọi là hiệu ứng Haas. Dưới ngưỡng ấy, tai người gộp tiếng trực tiếp và tiếng trễ thành *một* tiếng duy nhất — chỉ cảm thấy nó "rộng hơn" hoặc "lệch về một bên". Trên ngưỡng ấy, tai tách thành hai tiếng và người nghe gọi đó là tiếng vọng.

Ba con số nên thuộc

Khoảng cách hai tai người khoảng 17 xăng-ti-mét, tương ứng 0,5 mili giây. Đó là độ trễ lớn nhất mà bộ não dùng để định hướng nguồn âm theo phương ngang.

Trễ 20 mili giây tương ứng quãng đường 6,9 mét — vẫn dưới ngưỡng 30 mili giây, nên vẫn nghe là một tiếng.

Trễ 100 mili giây tương ứng 34,3 mét. Đó là lý do ở một sân vận động, tiếng từ loa xa vọng lại nghe rõ là một tiếng riêng.

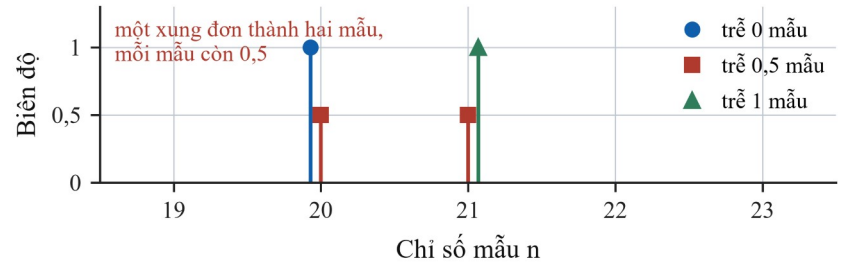
6.4.4 Trễ lẻ mẫu và cái giá của nó

Trễ nguyên mẫu là phép dời chỗ: không mẫu nào bị thay đổi giá trị, nên không mất gì. Nhưng nhiều việc thực tế đòi trễ một phần lẻ của mẫu — chẳng hạn đặt một nhạc cụ lệch 0,3 mili giây, hoặc một bộ trễ biến thiên liên tục như hiệu ứng flanger.

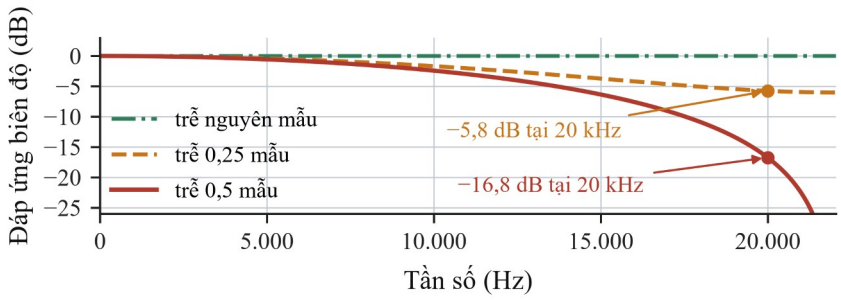
Muốn thế thì phải *đoán* giá trị nằm giữa hai mẫu đã có. Cách đoán rẻ nhất là nội suy tuyến tính.

$$y[n] = (1-\alpha)\,x[n-D] + \alpha\,x[n-D-1], \qquad 0 \le \alpha < 1 \tag{6.6}$$

Alpha là phần lẻ của độ trễ, nằm giữa 0 và 1. Khi alpha bằng 0 thì công thức trở về trễ nguyên mẫu.



(a) Trễ nửa mẫu bằng nội suy tuyến tính



(b) Cái giá của nội suy tuyến tính

Hình 6.15 Trễ nửa mẫu bằng nội suy tuyến tính và đáp ứng biên độ của nó

Hình bên trái cho thấy hậu quả trên một xung đơn: xung bị "trải" thành hai mẫu, mỗi mẫu còn một nửa. Hình bên phải cho thấy điều ấy nhìn trên trục tần số chính là một bộ lọc thông thấp: với trễ nửa mẫu, đáp ứng ở 20 kHz đã tụt 16,8 decibel.

Đây là lý do các bộ trễ biến thiên chất lượng cao không dùng nội suy tuyến tính mà dùng nội suy bậc cao hơn, hoặc bộ lọc thông toàn dải. Với cuốn sách này thì chỉ cần biết cái giá ấy tồn tại và tránh dùng trễ lẻ mẫu khi không cần.

6.4.5 Bốn cách làm sai phép trễ

Cộng hai mảng khác độ dài

`ValueError: operands could not be broadcast`

→ Đệm mảng ngắn cho bằng, hoặc dùng tron() vốn tự đệm

Trễ số âm

Không báo lỗi: hàm trả về nguyên mảng cũ

→ Trễ âm là "sớm": phải cắt bỏ đầu, không phải đệm

Trễ tính bằng giây, truyền vào chỗ nhận mẫu

Không báo lỗi: trễ 0,25 mẫu thay vì 11.025 mẫu

→ Đặt tên tham số cho rõ: ms hay mau, đừng để "d"

Trễ rồi cộng, đỉnh vượt 1,0

Không báo lỗi: chỉ sai tiếng khi ghi ra tệp int16

→ Kiểm tra đỉnh sau khi cộng; giảm hệ số khuếch đại trước khi ghi

Hình 6.16 Bốn lỗi thường gặp quanh phép trễ và cách chặn

Đáng chú ý là ba trong bốn lỗi trên *không* báo lỗi khi chạy. Đó là đặc điểm chung của lập trình âm thanh mà chương 5 đã nêu và chương này gặp lại ở mọi mục: chương trình chạy xong, ghi ra tệp, và tệp ấy sai.

Liệt kê 6.5 Trễ, có chặn ba lỗi trên

```
def tre_mau_so(x, n, giu_do_dai=False):
    """Đẩy tín hiệu muộn đi n mẫu."""
    a = np.asarray(x, dtype=float)
    n = int(n)
    if n < 0:
        raise ValueError("trễ âm – dùng hàm som() nếu cần cắt đầu")
    if n == 0:
        return a
    pad = (n, 0) if a.ndim == 1 else ((n, 0), (0, 0))
    ra = np.pad(a, pad)
    return ra[:a.shape[0]] if giu_do_dai else ra


def tre_ms(x, ms, sr=44100, giu_do_dai=False):
    """Tên tham số là ms – không nhận giây, không nhận mẫu."""
    return tre_mau_so(x, int(round(ms / 1000 * sr)), giu_do_dai)
```

Hai chi tiết nhỏ nhưng đáng giữ. Thứ nhất, hàm ném ngoại lệ khi nhận trễ âm thay vì lặng lẽ trả về mảng cũ — biến một lỗi im lặng thành một lỗi ồn ào. Thứ hai, tên tham số là "ms" chứ không phải "d" hay "t": người gọi không thể nhầm đơn vị nếu tên tham số nói rõ đơn vị.

6.4.6 Bảng quy đổi để dùng nhanh

Ba đại lượng — mili giây, số mẫu, khoảng cách — quy đổi qua lại suốt chương này và cả chương 8, chương 10. Bảng dưới đây gom lại để tra nhanh khi làm bài.

Bảng 6.6 Quy đổi giữa mili giây, số mẫu và khoảng cách

Trễ	Ở 44,1 kHz	Ở 48 kHz	Ở 96 kHz	Quãng đường	Tai nghe thấy gì
0,5 ms	22 mẫu	24 mẫu	48 mẫu	17 cm	Định hướng trái phải
1 ms	44 mẫu	48 mẫu	96 mẫu	34 cm	Vẫn là một tiếng
5 ms	220 mẫu	240 mẫu	480 mẫu	1,7 m	Độ dài fade mặc định
20 ms	882 mẫu	960 mẫu	1 920 mẫu	6,9 m	Một tiếng, nghe rộng ra
30 ms	1 323 mẫu	1 440 mẫu	2 880 mẫu	10,3 m	Ngưỡng tách thành hai tiếng
100 ms	4 410 mẫu	4 800 mẫu	9 600 mẫu	34,3 m	Rõ là tiếng vọng riêng
250 ms	11 025 mẫu	12 000 mẫu	24 000 mẫu	85,8 m	Tiếng vọng của mục 6.4.2

Hai cột giữa cho thấy vì sao không được viết cứng số mẫu trong mã: cùng một độ trễ 5 mili giây là 220 mẫu ở 44,1 kHz nhưng 480 mẫu ở 96 kHz. Đây chính là quy tắc thứ hai của mục 6.7.2 — mọi con số phải là tham số, và số mẫu phải tính từ mili giây cùng với tần số lấy mẫu, không bao giờ viết sẵn.

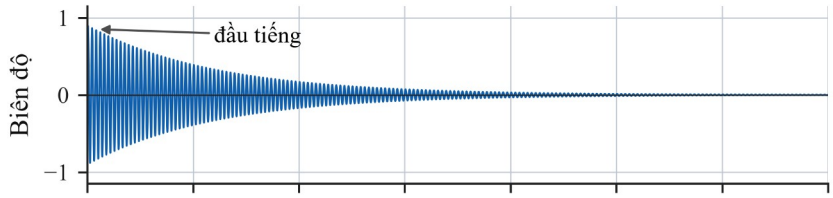
6.5 Đảo thời gian

6.5.1 Một dòng mã, một câu hỏi hay

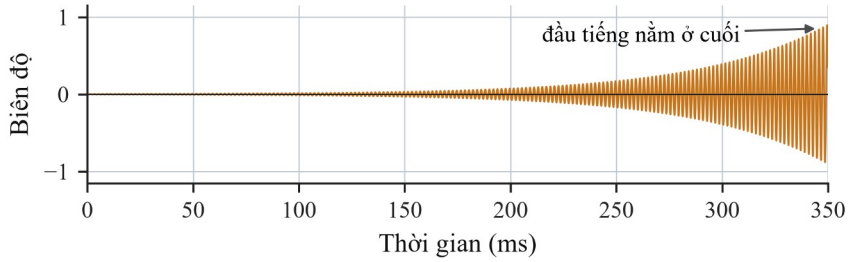
Đảo thời gian là phép đơn giản nhất của cả chương: đọc mảng theo thứ tự ngược. Trong numpy đó là x[::-1].

$$y[n] = x[N-1-n]\,; \qquad |Y[k]| = |X[k]| \tag{6.7}$$

Dòng thứ hai là chỗ đáng nói: phổ *biên độ* của tín hiệu đảo bằng đúng phổ biên độ của tín hiệu gốc. Chỉ có pha thay đổi.



(a) Tiếng gõ $x[n]$: đánh mạnh rồi tắt dần



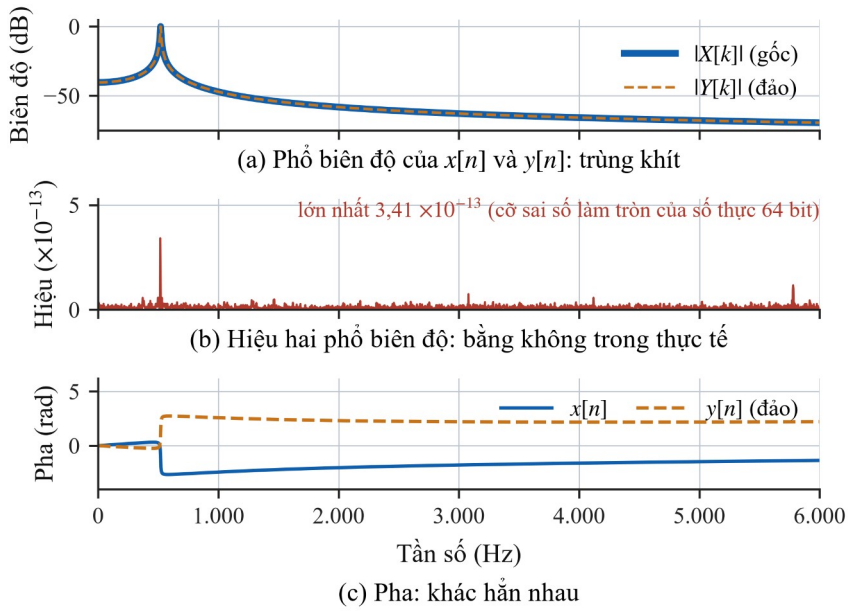
(b) Bản đảo $y[n] = x[N-1-n]$: lớn dần rồi dứt đột ngột

Hình 6.17 Một tiếng gõ và bản đảo của nó

Nghe thì hai tín hiệu khác nhau hoàn toàn. Tiếng gõ gốc đánh mạnh rồi tắt dần — tai nhận ra ngay là một vật bị gõ. Bản đảo lớn dần rồi dứt đột ngột — tai nhận ra là "âm thanh chạy ngược", một cảm giác không có trong tự nhiên.

6.5.2 Bằng chứng cho một điều đã nói ở chương 4

Mục 4.5 khẳng định rằng phổ biên độ *không* mô tả đủ một tín hiệu, và đưa ra hai ví dụ dựng sẵn. Đảo thời gian là ví dụ thứ ba, và là ví dụ thuyết phục nhất, vì hai tín hiệu nghe khác nhau đến thế mà phổ biên độ thì trùng khít.



(a) Phổ biên độ của $x[n]$ và $y[n]$: trùng khít

(b) Hiệu hai phổ biên độ: bằng không trong thực tế

(c) Pha: khác hẳn nhau

Hình 6.18 Phổ biên độ, hiệu hai phổ, và pha của tín hiệu gốc và tín hiệu đảo

Hình giữa là phép đo: hiệu lớn nhất giữa hai phổ biên độ ở cỡ 10 mũ trừ 13 — đúng bằng sai số làm tròn của số thực 64 bit, nghĩa là bằng không trong thực tế. Hình bên phải cho thấy pha thì khác hẳn.

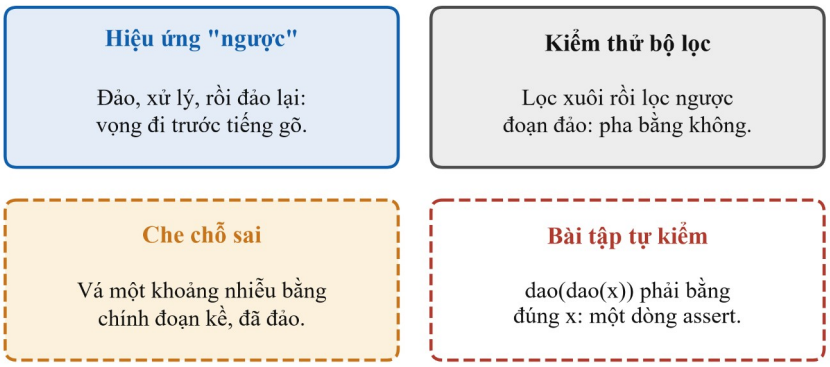
Một câu hỏi đáng suy nghĩ

Nếu phổ biên độ giống hệt nhau mà tai nghe khác hẳn nhau, thì những công cụ chỉ nhìn phổ biên độ — phổ đồ, bộ phân tích phổ trong phần mềm — bỏ sót cái gì?

Trả lời: chúng bỏ sót toàn bộ thông tin về *thời điểm*. Phổ đồ khắc phục được một phần vì nó chia tín hiệu thành nhiều khung ngắn, nhưng trong mỗi khung thì vẫn mất pha.

Đây là lý do trong chương 4 đã dặn: soát lỗi bằng phổ đồ thì tốt, nhưng không bao giờ được thay cho việc *nghe* lại.

6.5.3 Đảo dùng vào việc gì



Hình 6.19 Bốn chỗ dùng thật của phép đảo thời gian

Ứng dụng thứ hai trên hình đáng nói thêm, vì nó là một kỹ thuật thật sự dùng trong ngành. Lọc một tín hiệu rồi đảo nó, lọc lần nữa bằng cùng bộ lọc, rồi đảo lại — kết quả là một bộ lọc có *pha bằng không*, nghĩa là không làm lệch thời

điểm của bất cứ thành phần tần số nào. Trong scipy nó có sẵn với tên filtfilt, và chương 8 sẽ dùng đến.

Liệt kê 6.6 Đảo thời gian — và điểm dễ sai .copy()

```
def dao_thoi_gian(x):
    """Đảo trục thời gian. Mảng nhiều kênh chỉ đảo trục 0."""
    a = np.asarray(x, dtype=float)
    return a[::-1].copy() if a.ndim == 1 else a[::-1, :].copy()


# phép kiểm bắt buộc — đảo hai lần phải về đúng bản gốc:
assert np.array_equal(dao_thoi_gian(dao_thoi_gian(x)), x)
```

Vì sao phải gọi .copy()

x[::-1] không tạo mảng mới. Nó trả về một "view" — một cách nhìn khác vào cùng vùng nhớ, với bước nhảy âm.

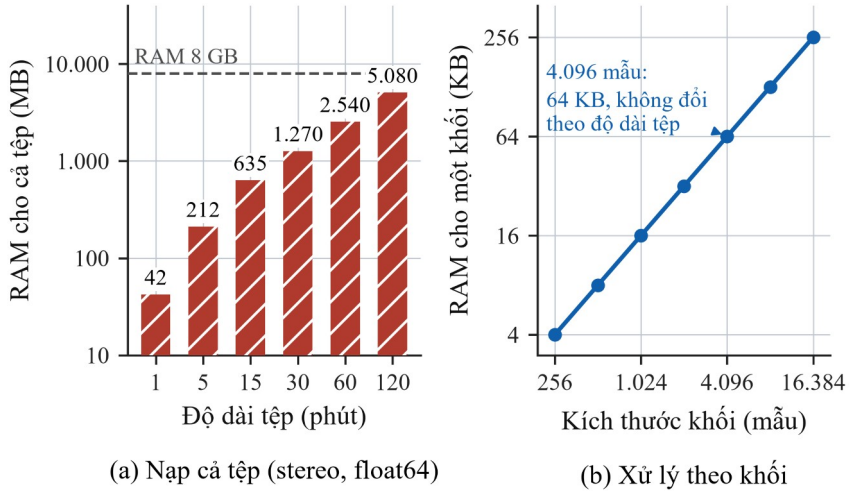
View ấy dùng để tính toán thì không sao, nhưng một số thư viện ghi tệp và một số hàm C bên dưới đòi mảng liên tục trong bộ nhớ. Gặp view bước âm thì chúng báo lỗi, hoặc nghiêm trọng hơn, ghi ra dữ liệu sai.

Đây chính là điểm dễ sai view-hay-copy đã gặp ở mục 5.2.3, lần này xuất hiện dưới một hình dạng khác.

6.6 Xử lý theo khối cho tệp dài

6.6.1 Vì sao không nạp cả tệp

Đến giờ mọi đoạn mã trong hai chương đều bắt đầu bằng việc đọc cả tệp vào một mảng. Với tư liệu ba mươi giây thì không vấn đề gì. Với một bản ghi buổi hoà nhạc hai tiếng thì có.



(a) Nạp cả tệp (stereo, float64)

(b) Xử lý theo khối

Hình 6.20 RAM cần cho cả tệp so với RAM cần cho một khối

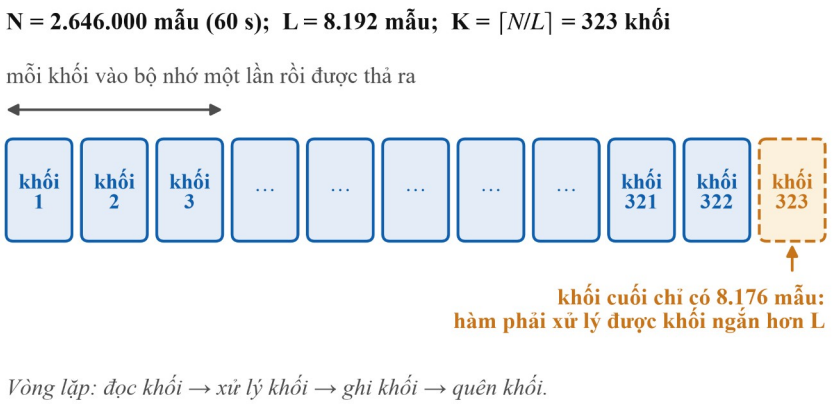
Một bản ghi hai tiếng, stereo, ở dạng số thực 64 bit chiếm 5,1 gi-ga-byte. Trên một máy 8 gi-ga-byte, chỉ riêng việc đọc nó đã đủ làm treo máy — và đó là mới đọc, chưa xử lý gì. Mọi phép tính trung gian còn cần thêm một bản sao nữa.

Cách làm đúng là chia tệp thành từng khối, đọc một khối, xử lý nó, ghi kết quả ra, rồi thả khối ấy đi. Khi ấy bộ nhớ cần dùng không phụ thuộc vào độ dài tệp nữa: khối 4 096 mẫu stereo chiếm 64 ki-lô-byte, dù tệp dài ba phút hay ba tiếng.

6.6.2 Chia khối

$$K = \left\lceil \frac{N}{L} \right\rceil ; \qquad R_{\text{khối}} = L \cdot N_{\text{kênh}} \cdot 8 \quad [\text{byte}] \tag{6.8}$$

K là số khối, N là tổng số mẫu, L là kích thước khối. Dấu ngoặc là phép làm tròn *lên*, vì khối cuối cùng thường không đầy. R_{khoi} là bộ nhớ cho một khối, với N_{ch} là số kênh và 8 byte cho mỗi mẫu số thực 64 bit.



Hình 6.21 Một tệp 60 giây chia thành 323 khối 8 192 mẫu

Khối cuối cùng — chỗ dễ bỏ sót nhất

Với N = 2 646 000 và L = 8 192, khối thứ 323 chỉ có 8 176 mẫu chứ không phải 8 192.

Mọi hàm xử lý khối phải chạy đúng với khối ngắn hơn L. Viết hàm giả định mọi khối đều dài đúng L là lỗi hay gặp nhất khi mới học phần này — và nó chỉ lộ ra ở cuối tệp, sau khi chương trình đã chạy vài phút.

Cách kiểm nhanh: chạy thử với một tệp có độ dài *không* chia hết cho L. Đây là một trong bốn loại phép kiểm ở mục 6.7.3.

Liệt kê 6.7 Chia khối và xử lý từng khối

```
def chia_khoi(x, khoi):
    """Cắt mảng thành danh sách khối; khối cuối có thể ngắn hơn."""
    a = np.asarray(x, dtype=float)
    k = int(khoi)
    return [a[i:i + k] for i in range(0, a.shape[0], k)]


def xu_ly_theo_khoi(x, khoi, ham):
    """ĐÚNG cho phép KHÔNG NHỚ. SAI cho phép có trạng thái."""
    return np.concatenate([ham(k) for k in chia_khoi(x, khoi)])


# phép kiểm: chia rồi ghép lại phải bằng đúng bản gốc
assert np.array_equal(
```

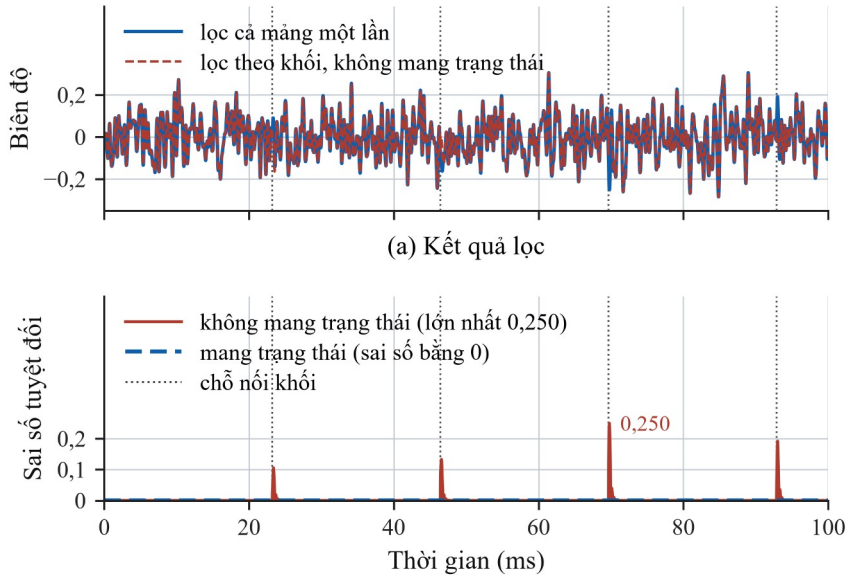
```
    np.concatenate(chia_khoi(x, 8192)), x)
```

6.6.3 Trạng thái — chỗ khác nhau giữa "chạy được" và "chạy đúng"

Sơ đồ "đọc khối, xử lý, ghi, quên" ở trên đúng cho mọi phép *không nhớ* — những phép mà kết quả của một mẫu chỉ phụ thuộc vào chính mẫu ấy. Nhân hệ số là một ví dụ: nhân mẫu thứ một triệu với 0,5 cho cùng kết quả dù xử lý cả mảng hay xử lý theo khối.

Bộ lọc thì không như thế. Mục 4.6 đã viết phương trình sai phân: giá trị đầu ra tại mẫu n phụ thuộc vào các mẫu đầu vào *trước* nó và các mẫu đầu ra trước nó. Thông tin về "trước nó" gọi là *trạng thái* của bộ lọc.

Cắt tín hiệu thành khối rồi gọi bộ lọc lại từ đầu ở mỗi khối tức là bắt bộ lọc khởi động lại từ số không ở mỗi chỗ nối khối. Mã vẫn chạy, vẫn ghi ra tệp — nhưng tệp ấy sai.



Hình 6.22 Lọc theo khối có mang và không mang trạng thái

Trên ví dụ của hình — nhiễu trắng qua bộ lọc thông thấp Butterworth bậc bốn, khối 1 024 mẫu — sai số lớn nhất khi không mang trạng thái là 0,250 trên thang biên độ đầy là 1,0. Nghe ra thì đó là một tiếng lách tách đều đặn, cứ mỗi 23 mili giây một lần. Khi mang trạng thái đúng cách, sai số bằng *đúng không*.

$$(y_k, z_k) = \text{lọc}(b, a, x_k, z_{k-1}) \tag{6.9}$$

Cách viết đúng chỉ là ghi nhớ trạng thái sau mỗi khối và truyền nó vào lần gọi sau. Trong scipy, hàm lfilter nhận và trả trạng thái qua tham số zi khi được yêu cầu; ở công thức trên, z_k là trạng thái sau khối thứ k.

Liệt kê 6.8 Lọc theo khối, có mang trạng thái

```
from scipy import signal as sg

def xu_ly_theo_khoi_co_trang_thai(x, khoi, b, a):
```

```
    """Lọc theo khối, MANG trạng thái từ khối trước sang khối sau."""
    zi = sg.lfilter_zi(b, a) * 0.0     # bắt đầu từ trạng thái nghỉ
    ra = []
    for k in chia_khoi(x, khoi):
        y, zi = sg.lfilter(b, a, k, zi=zi)   # <- nhận và trả zi
        ra.append(y)
    return np.concatenate(ra)


# phép kiểm bắt buộc: phải bằng ĐÚNG kết quả lọc cả mảng
dung = sg.lfilter(b, a, x)
mang = xu_ly_theo_khoi_co_trang_thai(x, 1024, b, a)
assert np.max(np.abs(dung - mang)) == 0.0
```

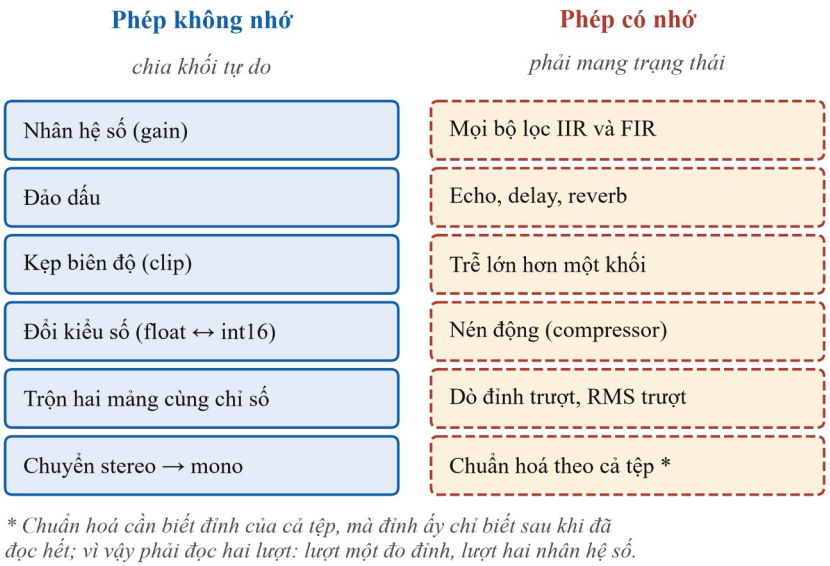
Phép kiểm đáng giá nhất của cả chương

Ba dòng cuối của đoạn mã trên là mẫu mực của loại phép kiểm "so với cách chậm": làm cùng một việc bằng hai cách, một cách chậm nhưng chắc chắn đúng, một cách nhanh nhưng dễ sai, rồi so hai kết quả.

Nó bắt được lỗi trạng thái ngay lập tức, và bắt được nó bằng một dấu bằng tuyệt đối chứ không phải một ngưỡng dung sai.

Mỗi khi viết một hàm xử lý theo khối mới, hãy viết ba dòng này trước, rồi mới viết hàm.

6.6.4 Phép nào có nhớ, phép nào không



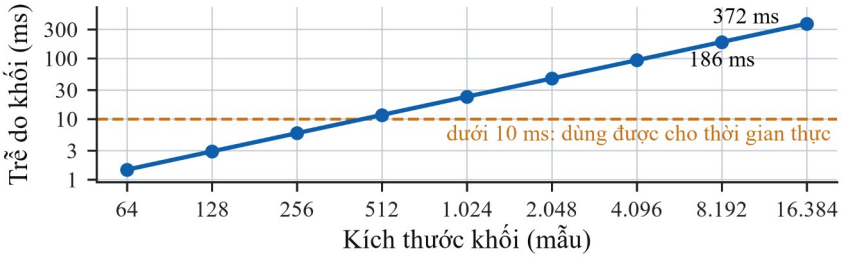
Hình 6.23 Phân loại các phép theo có trạng thái hay không

Cách nhận biết nhanh: tự hỏi "kết quả của mẫu này có phụ thuộc vào mẫu nào khác không". Nếu không thì phép ấy không nhớ và chia khối thoải mái. Nếu có thì phải mang trạng thái.

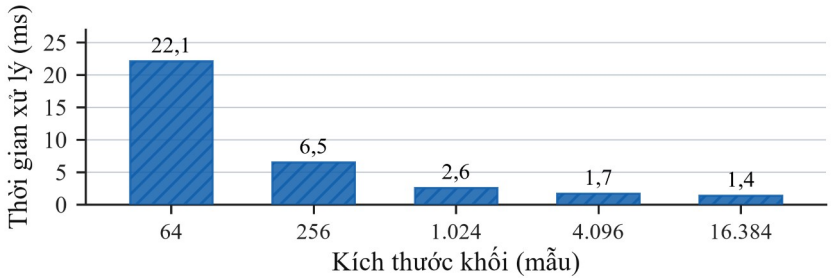
Trường hợp khó chịu nhất nằm ở dòng cuối bảng bên phải: chuẩn hoá theo cả tệp. Muốn đưa đỉnh về −1 dBFS thì phải biết đỉnh của cả tệp, mà đỉnh ấy chỉ biết được sau khi đã đọc hết. Cách duy nhất là đọc *hai lượt*: lượt một chỉ đo đỉnh,

lượt hai nhân hệ số. Chương 7 sẽ gặp lại đúng vấn đề này khi làm chuẩn hoá theo LUFS.

6.6.5 Chọn kích thước khối



(a) Khối càng lớn, trễ càng nhiều



(b) Khối càng nhỏ, chi phí vòng lặp càng lớn

Hình 6.24 Trễ và chi phí vòng lặp theo kích thước khối

Lại là hai áp lực ngược chiều. Khối càng lớn thì càng ít lần lặp, mỗi lần lặp làm nhiều việc hơn, nên tổng thời gian chạy giảm — từ khối 64 xuống khối 4 096, thời gian xử lý giảm khoảng mười lần. Nhưng khối càng lớn thì độ trễ càng nhiều: phải đợi đủ L mẫu mới xử lý được, nên khối 8 192 mẫu đã là 186 mili giây.

Bảng 6.7 Chọn kích thước khối theo mục đích

Mục đích	Kích thước nên dùng	Vì sao
Xử lý tệp ngoại tuyến	4 096 – 16 384	Không ai đợi trực tiếp, nên tối ưu cho tốc độ
Phát lại có xử lý	512 – 2 048	Trễ 12 – 46 ms, người dùng không nhận ra khi chỉ nghe
Thời gian thực có nhạc cụ	64 – 256	Trễ dưới 6 ms — người chơi không cảm thấy nhạc cụ "chậm tay"
Phân tích phổ theo khung	Bằng kích thước FFT	Để khối trùng với khung phân tích của chương 4

Một lưu ý về con số ở cột phải của hình F05: chúng phụ thuộc vào máy. Bậc độ lớn thì luôn như vậy, nhưng con số cụ thể trên máy của người đọc sẽ khác. Hãy tự chạy lại đoạn đo trên máy của mình — đó cũng là một bài thực hành nhỏ.

6.6.6 Đọc và ghi tệp theo khối

Phần trên chia khối trên một mảng đã nạp sẵn, chỉ để minh hoạ. Trong thực tế, cái làm cho xử lý theo khối có ý nghĩa là đọc từ *đĩa* theo khối — không bao giờ nạp cả tệp. Thư viện soundfile có sẵn cho việc ấy.

Liệt kê 6.9 Xử lý một tệp dài mà không nạp cả tệp

```
import soundfile as sf

L = 8192
with sf.SoundFile("vao.wav") as f_vao:
    with sf.SoundFile("ra.wav", "w",
                      samplerate=f_vao.samplerate,
                      channels=f_vao.channels,
                      subtype="PCM_24") as f_ra:
        zi = None
        for khoi in f_vao.blocks(blocksize=L, dtype="float64"):
            if zi is None:
                zi = np.zeros((max(len(a), len(b)) - 1,
                               f_vao.channels))
            y, zi = sg.lfilter(b, a, khoi, axis=0, zi=zi)
            f_ra.write(y)

# Bộ nhớ dùng: một khối, bất kể tệp dài bao nhiêu.
```

Ba điểm cần chú ý trong đoạn mã trên. Thứ nhất, sf.blocks trả về khối cuối cùng ngắn hơn L, đúng như đã nói ở mục 6.6.2 — vòng lặp không được giả định độ dài. Thứ hai, trạng thái zi phải có một cột cho mỗi kênh, nên hình dạng của nó phụ thuộc số kênh. Thứ ba, tệp ra được mở với subtype PCM_24 chứ không phải PCM_16, vì đây là tệp trung gian và còn đi qua các bước sau — quy tắc đã nêu ở mục 5.3.

6.7 Đóng gói thành mô-đun dùng lại được

6.7.1 Vì sao mục này nằm trong một cuốn sách về xử lý tín hiệu

Hai chương vừa qua đã sinh ra khoảng mười tám hàm. Bốn chương tới sẽ sinh thêm chừng ấy nữa. Nếu chúng nằm rải rác trong các tệp sao chép lẫn nhau thì đến chương 12 sẽ không ai còn biết bản nào là bản đúng, và dự án thực hành không hoàn thành được.

Trước: bốn bản sao của cùng một đoạn mã

```
lam_bai/
    thu1.py          ← sao chép 40 dòng
    thu2.py          ← sao chép, sửa 1 số
    thu2_final.py    ← sao chép lần nữa
    thu2_final_v2.py ← khác nhau ở đâu?

Sửa một lỗi → phải sửa ở BỐN chỗ. Bỏ sót một chỗ
→ hai kết quả khác nhau mà không ai biết vì sao.
```

Sau: một mô-đun, nhiều kịch bản gọi nó

```
am/
    __init__.py
    io.py         doc_wav, ghi_wav
    bien_do.py    gain_db, chuan_hoa_dinh
    thoi_gian.py  fade, crossfade, tre
    khoi.py       chia_khoi, xu_ly_theo_khoi
    test_am.py    các phép kiểm
lam_bai/thu1.py: import am  ← 6 dòng
Sửa một lỗi → sửa ĐÚNG MỘT chỗ.
```

Hình 6.25 Bốn bản sao của một đoạn mã, và một mô-đun được nhiều kịch bản gọi

Vấn đề của cột trái không phải là "trông lộn xộn". Nó là: sửa một lỗi thì phải sửa ở bốn chỗ, bỏ sót một chỗ thì hai tệp kết quả khác nhau mà không ai biết vì sao. Đây là tình huống thật, ai mới viết mã cũng gặp, và nó xuất hiện rất sớm.

6.7.2 Bốn quy tắc cho một hàm dùng lại được

Hàm khó dùng lại

```
def xu_ly():
    x, sr = doc_wav("D:/bai/giong.wav")
    x = x * 0.5
    x[:220] *= np.linspace(0, 1, 220)
    ghi_wav("D:/bai/ra.wav", x, 44100)
    print("xong")
· Đường dẫn cứng → máy khác chạy sai
· 220 và 44100 rời nhau → đổi sr là sai
· Vừa đọc, tính, ghi, in → khó kiểm thử
```

Hàm dùng lại được

```
def fade_vao(x, ms=5.0, sr=44100):
    """Nhân đường bao đi lên vào đầu đoạn."""
    a = np.array(x, dtype=float, copy=True)
    n = min(int(ms / 1000 * sr), len(a))
    a[:n] *= duong_bao_fade(n)
    return a
· Nhận mảng, trả mảng, không sửa tại chỗ
· ms và sr là tham số, không phải hằng
· Kiểm thử được bằng một dòng assert
```

Hình 6.26 Một hàm khó dùng lại và một hàm dùng lại được

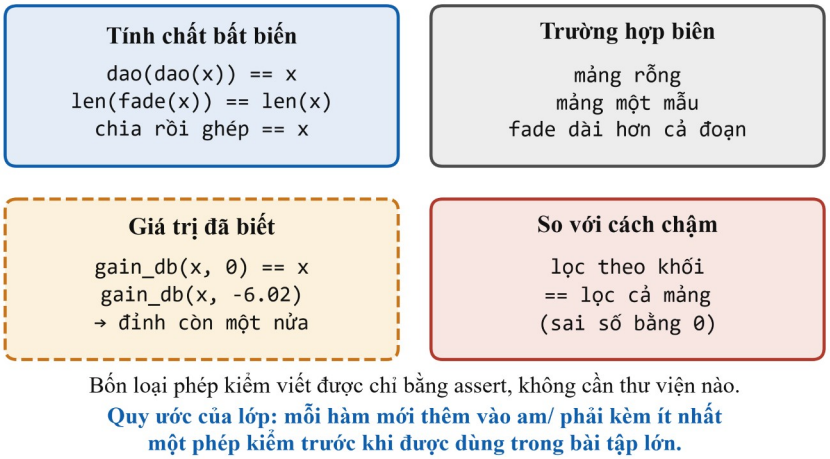
– **Nhận mảng, trả mảng.** Hàm tính toán không đọc đĩa và không ghi đĩa. Đọc ghi là việc của mô-đun io. Hàm nào vừa tính vừa ghi tệp thì không kiểm thử được, vì muốn kiểm phải có tệp thật.

– **Mọi con số là tham số, có giá trị mặc định.** Số 220 và số 44100 phải đi cùng nhau; viết cứng số 220 thì đổi tần số lấy mẫu là sai ngay mà không báo lỗi.

– **Không sửa mảng đầu vào tại chỗ.** Dùng np.array(x, copy=True) ở đầu hàm. Không có nó thì gọi hàm hai lần cho hai kết quả khác nhau — loại lỗi mất nhiều giờ nhất để tìm ra.
– **Có một dòng docstring nói hàm làm gì.** Một dòng thôi. Nó là thứ hiện ra khi gõ help(ham) ở tháng sau, khi người viết đã không còn nhớ gì.

6.7.3 Bốn loại phép kiểm

Không cần thư viện kiểm thử nào. Câu lệnh assert của Python là đủ cho toàn bộ cuốn sách này.



Hình 6.27 Bốn loại phép kiểm dùng cho mô-đun am

Bảng 6.8 Bốn loại phép kiểm và ví dụ trong chương này

Loại	Ý tưởng	Ví dụ đã gặp trong chương 6
Tính chất bất biến	Một quan hệ phải đúng với mọi đầu vào	dao(dao(x)) bằng x; fade không đổi độ dài; chia khối rồi ghép lại bằng bản gốc
Trường hợp biên	Đầu vào ở rìa miền xác định	Mảng rỗng; mảng một mẫu; fade dài hơn cả đoạn; tệp có độ dài không chia hết cho L
Giá trị đã biết	Một đầu vào mà đáp số tính được bằng tay	gain 0 dB không đổi gì; đường bao nửa cosin có giá trị đầu bằng 0 và giá trị cuối bằng 1
So với cách chậm	Hai cách làm cùng một việc, so kết quả	Lọc theo khối có mang trạng thái phải bằng *đúng* lọc cả mảng

Liệt kê 6.10 Một tệp kiểm thử tối giản cho mô-đun am

```
# am/test_am.py — chạy bằng: python -m am.test_am
import numpy as np
from . import thoi_gian as tg
```

```
def kiem():
    n = 0
    x = np.random.default_rng(0).normal(size=4410) * 0.3

    # 1. tính chất bất biến
    assert np.array_equal(tg.dao(tg.dao(x)), x);            n += 1
    assert len(tg.fade_hai_dau(x)) == len(x);               n += 1

    # 2. trường hợp biên
    assert len(tg.fade_vao(np.zeros(0))) == 0;              n += 1
    assert len(tg.fade_vao(x[:3], ms=50.0)) == 3;           n += 1

    # 3. giá trị đã biết
    w = tg.duong_bao_fade(220)
    assert w[0] == 0.0 and w[-1] == 1.0;                    n += 1

    # 4. so với cách chậm  (xem mục B6.6.3)
    # ...

    print(f"{n}/{n} phép kiểm đạt")

if __name__ == "__main__":
    kiem()
```

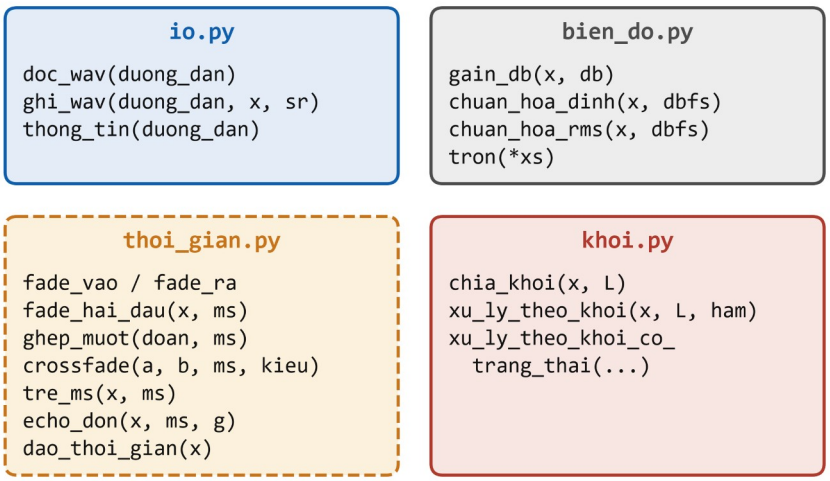
Quy ước từ chương này trở đi

Mỗi hàm mới thêm vào mô-đun am phải kèm ít nhất một phép kiểm, viết *trước* khi hàm ấy được gọi trong dự án thực hành.

Tệp test_am.py phải chạy sạch trước khi coi một phần việc là xong. Một phép kiểm không đạt là một lỗi phải sửa, không phải một cảnh báo bỏ qua được.

Bộ tự kiểm của mô-đun sigtools dùng để sinh mọi con số trong sách này chạy theo đúng bốn loại trên, hiện có 241 phép kiểm.

6.7.4 Mô-đun am sau chương 6



Hình 6.28 Bốn tệp và mười tám hàm của mô-đun am

Liệt kê 6.11 Bố cục thư mục và tệp __init__.py

```
am/
    __init__.py
    io.py
    bien_do.py
    thoi_gian.py
    khoi.py
    test_am.py

# am/__init__.py – gom các tên hay dùng lên một tầng
from .io        import doc_wav, ghi_wav, thong_tin
from .bien_do   import gain_db, chuan_hoa_dinh, chuan_hoa_rms, tron
from .thoi_gian import (duong_bao_fade, fade_vao, fade_ra,
                        fade_hai_dau, ghep_muot, crossfade,
                        tre_ms, echo_don, dao_thoi_gian)
from .khoi      import chia_khoi, xu_ly_theo_khoi

# nhờ đó kịch bản chỉ cần:
#     import am
#     y = am.ghep_muot([a, b, c], ms=5.0)
```

Một kịch bản dùng mô-đun này chỉ còn vài dòng, và mọi con số kỹ thuật đều nằm ở chỗ dễ đọc.

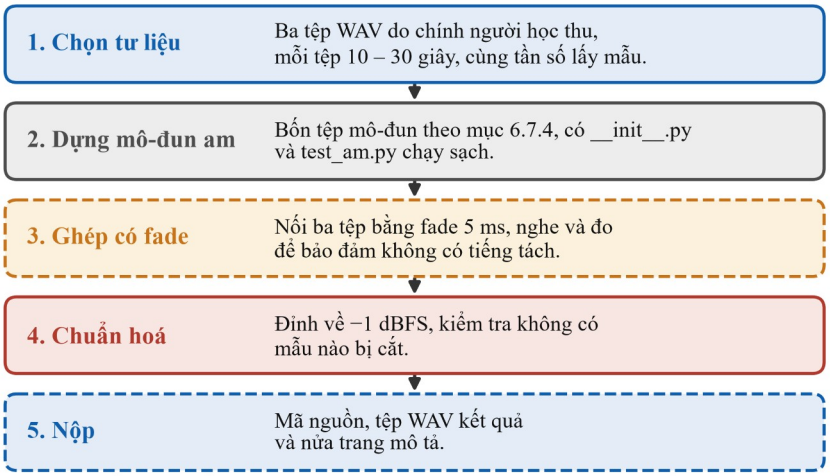
Liệt kê 6.12 Một kịch bản hoàn chỉnh sau khi đã có mô-đun

```
import am

doan = [am.doc_wav(f"tu_lieu/{t}.wav")[0] for t in ("a", "b", "c")]
y = am.ghep_muot(doan, ms=5.0)
y = am.chuan_hoa_dinh(y, dinh_dbfs=-1.0)
am.ghi_wav("ket_qua.wav", y, 44100)
```

6.8 Dự án thực hành — giai đoạn 1

6.8.1 Yêu cầu



Toàn bộ giai đoạn 1 dùng lại các hàm đã viết ở chương 5 và chương 6.

Hình 6.29 Năm bước của giai đoạn 1

Giai đoạn 1 nên làm xong trước khi sang chương 8. Toàn bộ giai đoạn này dùng lại đúng những hàm đã viết trong chương 5 và chương 6 — không cần học thêm kiến thức mới nào.

Bảng 6.9 Kết quả cần có

Hạng mục	Yêu cầu cụ thể
Tư liệu	Ba tệp WAV tự thu, mỗi tệp 10 – 30 giây, cùng tần số lấy mẫu và cùng số kênh. Không dùng tệp tải trên mạng không rõ nguồn
Mô-đun am	Bốn tệp io.py, bien_do.py, thoi_gian.py, khoi.py và một __init__.py. Kèm test_am.py chạy sạch
Kịch bản	Một tệp lam_bai.py không quá 30 dòng, chỉ gọi mô-đun, không chứa dòng xử lý tín hiệu nào
Kết quả	Một tệp WAV: ba đoạn nối bằng ghep_muot với fade 5 ms, chuẩn hoá đỉnh về −1 dBFS
Mô tả	Nửa trang: nguồn tư liệu, các tham số đã chọn, và một câu cho biết đã kiểm chỗ nối bằng cách nào

6.8.2 Cách tự kiểm

Bảng 6.10 Bốn tiêu chí tự kiểm giai đoạn 1

Tiêu chí	Trọng số	Cách kiểm
Chạy được trên máy khác	3	Giải nén vào thư mục mới, chạy lam_bai.py. Không được có đường dẫn tuyệt đối nào trong mã
Không có tiếng tách ở chỗ nối	3	Nghe; và đo năng lượng trên 8 kHz quanh hai chỗ nối
Đỉnh đúng −1 dBFS	2	Đo đỉnh tệp kết quả, sai lệch cho phép 0,1 dB. Không có mẫu nào chạm hoặc vượt 1,0
Mô-đun tách khỏi kịch bản	2	lam_bai.py không được chứa numpy hay scipy; test_am.py chạy sạch

Ba lỗi hay gặp nhất ở giai đoạn này

Đường dẫn tuyệt đối kiểu D:/bai/... trong mã. Máy khác không có thư mục ấy nên chương trình dừng ở dòng đầu tiên.

Chuẩn hoá *trước* khi ghép thay vì sau. Ghép ba đoạn đã chuẩn hoá riêng thì tổng có thể vượt quá 1,0, và tệp kết quả bị cắt đỉnh.

Ghi tệp ra ở PCM_16 rồi lại đọc vào để xử lý tiếp. Mỗi vòng như thế lại thêm một lần lượng tử hoá — quy tắc ở mục 5.3 là tệp trung gian dùng PCM_24 hoặc số thực.

6.9 Thực hành, bài tập và tổng kết

6.9.1 Ba bài thực hành

Ba bài dưới đây làm ngay sau khi đọc xong chương, mỗi bài khoảng mười phút, trên tư liệu của chính mình.

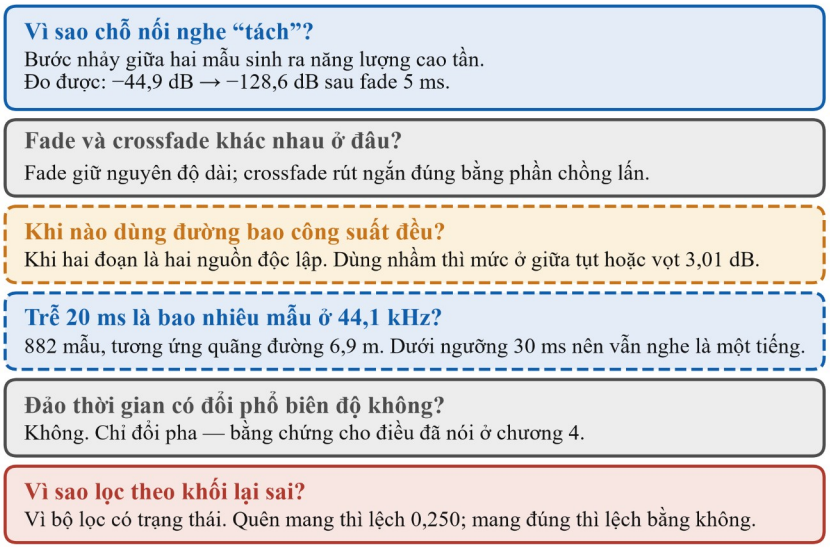
- **Bài 1 — nghe cho được tiếng tách.** Cắt một tệp giọng đọc ở hai chỗ bất kỳ giữa câu, ghép ba mảnh lại theo thứ tự đảo. Nghe. Rồi làm lại bằng ghep_muot với fade 5 ms và nghe lại. In ra bước nhảy tại hai chỗ nối, trước và sau. Đáp số phải là hai con số dương và hai số không.
- **Bài 2 — đo tiếng tách bằng số.** Với đúng hai bản ghép của bài 1, tính năng lượng trên 8 kHz của mỗi bản. So sánh. Sau đó thử các độ dài fade 0,25 · 1 · 5 · 20 mili giây và vẽ bốn con số ấy thành một đồ thị cột.
- **Bài 3 — chứng minh lỗi trạng thái.** Lọc thông thấp một đoạn nhiễu trắng bằng hai cách: cả mảng một lần, và theo khối 1 024 mẫu không mang trạng thái. In sai số lớn nhất. Rồi sửa lại thành có mang trạng thái và in lại. Nghe cả hai bản.

6.9.2 Bảy bài tập

Bảng 6.11 Bài tập của chương 6

Bài	Nội dung	Đáp số hoặc tiêu chí
1	Ở 48 kHz, fade 5 ms là bao nhiêu mẫu? Và ở 96 kHz?	240 mẫu; 480 mẫu
2	Nối 20 đoạn 0,25 s bằng crossfade 50 ms. Tổng độ dài là bao nhiêu? Muốn tổng đúng 5,000 s thì mỗi đoạn phải cắt dài bao nhiêu?	4,050 s; mỗi đoạn 0,2975 s
3	Một loa cách người nghe 12 m, loa kia cách 4 m. Chênh lệch trễ là bao nhiêu? Nghe ra một tiếng hay hai tiếng?	23,3 ms; vẫn là một tiếng — dưới ngưỡng 30 ms
4	Viết hàm som(x, ms) làm ngược lại phép trễ: cắt bỏ phần đầu. Kèm một phép kiểm cho quan hệ som(tre(x, ms), ms) bằng x.	Hàm chạy và phép kiểm đạt
5	Một tệp 44,1 kHz stereo dài 47 phút. Chia khối 4 096 thì được bao nhiêu khối, và khối cuối có bao nhiêu mẫu?	30 362 khối; khối cuối 3 344 mẫu
6	Dùng crossfade nối một đoạn với chính *bản đảo* của nó. Nghe. Giải thích vì sao chỗ nối nghe êm hơn hẳn so với ghép thẳng hai đoạn khác nhau.	Vì tại chỗ nối hai bản có cùng giá trị mẫu — đường bao dán hai giá trị giống nhau
7	Gói toàn bộ hàm đã viết ở chương 5 và chương 6 thành mô-đun am theo bố cục ở mục 6.7.4, kèm test_am.py có ít nhất tám phép kiểm.	Đây là bước 2 của dự án thực hành giai đoạn 1

6.9.3 Sáu câu phải trả lời được



Hình 6.30 Sáu câu tổng kết chương 6

Nếu còn câu nào chưa trả lời được, hãy quay lại đúng mục tương ứng trước khi sang chương 7. Cả sáu câu đều có đáp số bằng số, và người đọc tự kiểm được trên máy của mình.

6.9.4 Ba điều đọng lại từ chương này

- **Mọi hiện tượng nghe được đều đo được.** Tiếng tách là năng lượng trên 8 kHz. Lỗ im lặng là mức RMS tụt. Lỗi trạng thái là sai số so với cách chậm. Khi đã đo được thì không còn phải tranh cãi "tôi nghe thấy" hay "tôi không nghe thấy".
- **Ba trong bốn lỗi của chương này không báo lỗi khi chạy.** Đây là đặc trưng của lập trình âm thanh. Cách phòng duy nhất là phép kiểm viết sẵn và thói quen nghe lại kết quả.
- **Hàm viết cẩu thả hôm nay thì bốn chương sau phải sửa.** Chương 8, 10 và 11 đều đứng trên phép trễ và phép xử lý theo khối của chương này. Bốn quy tắc ở mục 6.7.2 không phải là chuyện hình thức.

6.9.5 Chuẩn bị cho chương 7

Chương 7 — Tăng cường chất lượng — bắt đầu từ đúng chỗ giai đoạn 1 của dự án thực hành dừng lại: đỉnh đã về −1 dBFS, vậy đã đủ chưa.

- Chuẩn bị ba tệp: một đoạn nhạc pop hiện đại, một đoạn nhạc cổ điển, và một đoạn giọng đọc. Ba tệp này có *độ to* cảm nhận rất khác nhau dù đỉnh có thể như nhau — đó là chất liệu của chương 7.
- Một tệp có tiếng ồn nền rõ ràng: quạt, điều hoà, tiếng đường phố. Chương 7 sẽ dùng nó cho phần khử nhiễu.
- Đọc lại mục 1.5 về đường đẳng âm và mục 5.7 về đỉnh so với RMS. Chương 7 nối trực tiếp vào hai mục ấy.

6.9.6 Một câu hỏi để tự trả lời trước chương sau

Giai đoạn 1 của dự án thực hành yêu cầu chuẩn hoá đỉnh về −1 dBFS. Hãy làm thử việc sau trên ba tệp đã chuẩn bị ở trên: chuẩn hoá cả ba về đúng −1 dBFS, rồi nghe lần lượt ba tệp ở cùng một mức âm lượng loa.

Ba tệp có *đỉnh* bằng nhau chính xác. Câu hỏi: nghe có thấy chúng to bằng nhau không? Nếu không thì tệp nào nghe to nhất, tệp nào nhỏ nhất, và chênh nhau chừng bao nhiêu decibel theo cảm nhận?

Không cần trả lời đúng. Chỉ cần có một con số ước lượng của riêng mình — chương 7 sẽ đo lại con số ấy bằng thước LUFS.

Tài liệu tham khảo của chương

Tài liệu của thư viện

- Tài liệu numpy — phần "Indexing on ndarrays" (lát cắt bước âm và view) và phần "Broadcasting" (nhân đường bao một chiều vào tín hiệu hai kênh). Hai phần này đủ cho mục 6.2 và 6.5.
- Tài liệu scipy.signal — mục lfilter và lfilter_zi. Đây là nguồn trực tiếp của mục 6.6.3 về trạng thái bộ lọc; phần mô tả tham số zi nên đọc kỹ.
- Tài liệu python-soundfile — lớp SoundFile và phương thức blocks. Nguồn của mục 6.6.6 về đọc ghi tệp theo khối.
- Hướng dẫn Python — "Modules" và "Packages" trong Python Tutorial. Đủ cho toàn bộ mục 6.7 về bố cục mô-đun.

Sách và giáo trình

- Zölzer, U. (chủ biên) — DAFX: Digital Audio Effects, ấn bản 2. Chương về delay-based effects là nguồn của mục 6.4, đặc biệt phần về nội suy trong bộ trễ biến thiên.
- Smith, J. O. — Physical Audio Signal Processing. Phần về fractional delay filters đi sâu hơn hẳn mục 6.4.4, dành cho ai muốn hiểu vì sao nội suy tuyến tính lại là bộ lọc thông thấp.
- Blauert, J. — Spatial Hearing. Nguồn về hiệu ứng Haas và ngưỡng 30 mili giây nêu ở mục 6.4.3.
- Pirkle, W. — Designing Audio Effect Plugins in C++. Chương đầu về kiến trúc xử lý theo khối và trạng thái, viết cho người làm phần mềm thời gian thực.

Chuẩn và quy ước

- AES17 — quy ước đo lường cho thiết bị âm thanh số. Phần về đo trong dải cao là nền của phép đo năng lượng trên 8 kHz dùng ở mục 6.2.
- EBU R 128 — chưa dùng ở chương này nhưng là đích của chương 7; nên đọc lướt trước.

Đọc thêm theo hướng

- **Muốn đi sâu về chỗ chuyển tiếp:** tìm đọc về overlap-add và overlap-save. Hai kỹ thuật ấy dùng đúng ý tưởng chồng lấn của mục 6.3 nhưng cho mục đích khác — nối các khung tích chập lại với nhau.
- **Muốn đi sâu về xử lý theo khối:** tìm đọc mã nguồn của thư viện sounddevice, phần callback. Nó cho thấy vì sao trong xử lý thời gian thực, hàm xử lý một khối *không* được phép cấp phát bộ nhớ hay đọc đĩa.
- **Muốn đi sâu về tổ chức mã:** tìm đọc tài liệu của pytest và của thư viện hypothesis. Hypothesis tự sinh đầu vào để kiểm các tính chất bất biến ở mục 6.7.3 — đúng loại phép kiểm đầu tiên trong bốn loại.

Ghi chú về phiên bản thư viện

Mã trong bài viết theo numpy 1.26 trở lên, scipy 1.11 trở lên và soundfile 0.12 trở lên. Hai chỗ có thể khác nếu dùng phiên bản cũ hơn.

- scipy.signal.lfilter với tham số zi trả về một bộ hai phần tử. Ở phiên bản rất cũ, thứ tự trục của zi cho tín hiệu nhiều kênh khác với bây giờ — nếu kết quả sai ở kênh thứ hai thì đó là nguyên nhân.
- Phương thức SoundFile.blocks có tham số overlap; bài này không dùng nhưng nó cần cho overlap-add ở chương 9.

Một lưu ý về các con số trong bài

> **Cách đọc các con số**
>
> Các con số về số mẫu, độ dài, dung lượng bộ nhớ và độ trễ theo khoảng cách là kết quả tính chính xác, kiểm được bằng máy tính bỏ túi.
>
> Các con số về năng lượng cao tần và về sai số trạng thái là kết quả *đo* trên tư liệu dựng sẵn mô tả trong chú thích từng hình. Trên tư liệu khác, con số sẽ khác; điều không đổi là bậc độ lớn và dấu của chênh lệch.
>
> Các con số đo thời gian chạy — mục 6.6.5 — phụ thuộc vào máy. Hãy tự chạy lại trên máy của mình.
>
> Mọi phép tính và mọi đáp số bài tập trong bài đều sinh ra từ mô-đun sigtools.py, đã qua 241 phép tự kiểm.