MỤC LỤC

Chương 5. Lập trình cơ bản phần 1 5
5.1 Vị trí của chương trong cuốn sách 5
5.2 Mảng numpy cho âm thanh 7
5.3 Đọc và ghi tệp WAV 13
5.4 Chỉnh gain theo decibel 17
5.5 Trộn nhiều nguồn 21
5.6 Cắt và ghép 25
5.7 Chuẩn hoá 29
5.8 Thực hành và bài tập 32
5.9 Tổng kết chương 34

DANH MỤC HÌNH VẼ

Hình 5.1 Sáu thao tác của chương 5 và trình tự của chúng.....6
Hình 5.2 Bốn chương lý thuyết đọng lại thành một mảng số và sáu thao tác trên nó.....7
Hình 5.3 Dãy mẫu vẽ ra và mảng numpy trong bộ nhớ là cùng một thứ.....8
Hình 5.4 Mảng mono có shape (N,), mảng stereo có shape (N, 2).....9
Hình 5.5 Cắt lát bằng chỉ số mẫu, nhưng nghĩ bằng giây.....10
Hình 5.6 Cùng một phép sửa lát cắt: không copy thì mảng gốc bị sửa theo.....10
Hình 5.7 Cùng một phép nhân viết hai cách: vòng lặp Python và phép trên cả mảng…11
Hình 5.8 Bốn kiểu số gặp trong xử lý âm thanh và chỗ dùng của mỗi kiểu.....12
Hình 5.9 Từ tệp trên đĩa tới mảng trong RAM và ngược lại.....13
Hình 5.10 Điều xảy ra khi lấy mảng int16 mà xử lý như thể nó là float.....15
Hình 5.11 Dung lượng tệp, bộ nhớ khi mở ra, và giới hạn thực tế theo RAM của máy 16
Hình 5.12 Ba lỗi đọc ghi, dấu hiệu nhận ra và cách chặn.....17
Hình 5.13 Quan hệ giữa mức decibel và hệ số nhân, và tác dụng lên dạng sóng.....18
Hình 5.14 Cắt đỉnh nhìn trên dạng sóng và nhìn trên phổ.....19
Hình 5.15 Cùng hai phép gain, hai thứ tự, hai kết quả hoàn toàn khác nhau.....20
Hình 5.16 Mọi mẫu đều đúng toàn thang, nhưng đường liên tục thì vượt lên trên.....21
Hình 5.17 Hai nguồn đều nằm ở nửa thang, nhưng tổng thì gần chạm toàn thang.....22
Hình 5.18 Mức phải hạ mỗi nguồn theo số nguồn, ở hai giả định khác nhau.....22
Hình 5.19 Ba trường hợp trộn hai nguồn cùng mức, ba kết quả hoàn toàn khác nhau. .24
Hình 5.20 Trộn hai đoạn khác độ dài: đệm số không cho cái ngắn.....25
Hình 5.21 Ba đoạn cắt ra rồi ghép lại theo thứ tự mới.....26
Hình 5.22 Chỗ nối có bước nhảy đột ngột, và bước nhảy ấy rải năng lượng khắp dải cao 27
Hình 5.23 Cắt tuỳ tiện và cắt ở điểm cắt không: giá trị ở đầu mối khác hẳn nhau.....27
Hình 5.24 Cùng hai nguồn, hai cách chuẩn hoá, bốn kết quả khác hẳn nhau.....29
Hình 5.25 Hệ số đỉnh của năm loại nguồn âm điển hình.....30
Hình 5.26 Bốn bước của một phép chuẩn hoá an toàn.....31
Hình 5.27 Sáu thao tác nối với nhau thành một dây chuyền, và bốn quy tắc đi cùng....34
Hình 5.28 Muốn làm gì thì gõ gì — chín dòng đủ cho phần lớn công việc.....34

DANH MỤC BẢNG BIỂU

Bảng 5.1 Sáu thao tác của chương 5 và việc chúng làm....6
Bảng 5.2 Các phép hay dùng trên mảng nhiều kênh....9
Bảng 5.3 Bốn kiểu số và đặc tính của chúng....12
Bảng 5.4 Các tham số của sf.read và khi nào cần đụng tới....14
Bảng 5.5 Vài mức gain nên thuộc lòng....18
Bảng 5.6 Hai cách ước lượng và khi nào dùng cái nào....23
Bảng 5.7 Ba điều phải khớp khi ghép....28
Bảng 5.8 Hệ số đỉnh tiêu biểu và ý nghĩa thực hành....30
Bảng 5.9 Ba bài thực hành và tiêu chí hoàn thành....32
Bảng 5.10 Bốn lỗi hay gặp và cách tự phát hiện....33
Bảng 5.11 Chương trình ấy dùng lại đúng những gì của chương này....36

CHƯƠNG 5
LẬP TRÌNH CƠ BẢN PHẦN 1

Mục tiêu của chương

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

- Đọc được hình dạng và kiểu số của một mảng âm thanh, phân biệt được mono với stereo, và giải thích được vì sao tần số lấy mẫu phải giữ riêng.
- Cắt lát được một mảng theo thời gian, và biết khi nào phải gọi .copy().
- Viết được mã theo lối vector hoá, và nói được vì sao không dùng vòng lặp trên từng mẫu.
- Đọc và ghi được tệp WAV đúng quy ước, chọn được kiểu số phù hợp, và chặn được ba lỗi đọc ghi hay gặp.
- Chỉnh được gain theo decibel, đếm được số mẫu bị cắt đỉnh, và giải thích được vì sao cắt đỉnh nghe rè.
- Trộn được nhiều nguồn có kiểm mức, và tính được lượng headroom cần chừa theo số nguồn.
- Cắt và ghép được nhiều đoạn, nhận ra được nguyên nhân của tiếng "tách" ở chỗ nối.
- Chuẩn hoá được theo đỉnh và theo RMS, và chọn được cách phù hợp với từng mục đích.

> **Chương này khác bốn chương trước ở chỗ nào**
>
> Bốn chương đầu hỏi "chuyện gì đang xảy ra". Chương này hỏi "gõ cái gì vào máy" — và mỗi mục kết thúc bằng một đoạn mã chạy được chứ không phải một công thức.
>
> Không có khái niệm toán mới nào. Mọi thứ dùng ở đây đều đã học ở bốn chương trước; cái mới chỉ là cách viết chúng ra.
>
> Đổi lại, chương này đòi một thứ mà bốn chương trước không đòi: phải *gõ*. Đọc mà không gõ thì không học được mục nào của chương này.

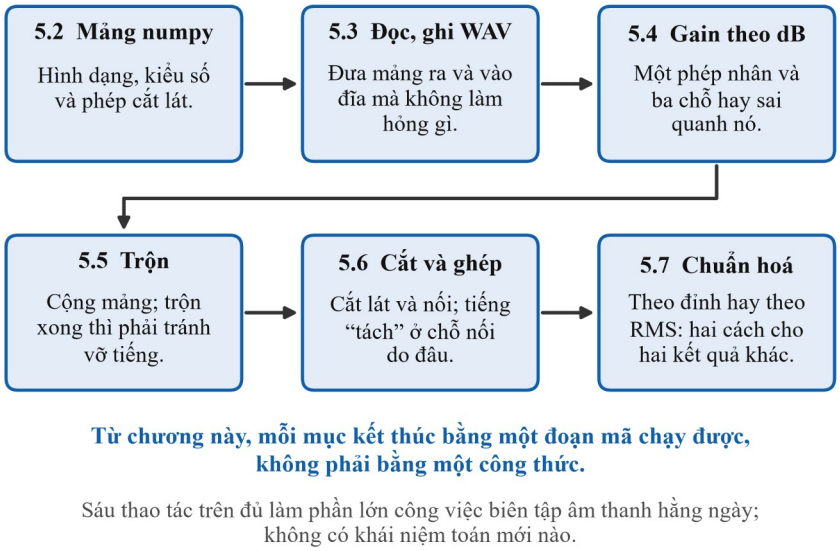
5.1 Vị trí của chương trong cuốn sách

5.1.1 Chương đầu tiên của phần lập trình cơ bản

Bốn chương đầu của cuốn sách xây nền. Chương 1 nói về âm thanh như một hiện tượng vật lý, chương 2 nói về nó khi đã thành điện áp, chương 3 nói về phép biến nó thành số, chương 4 nói về các công cụ toán để làm việc với dãy số ấy.

Từ chương này, cuốn sách đổi hẳn nhịp. Chương 5 và chương 6 làm thành phần "Lập trình cơ bản", và chữ "lập trình" cần được hiểu đúng nghĩa đen: mỗi

mục của chương kết thúc bằng một đoạn mã chạy được, không phải bằng một công thức.



Hình 5.1 Sáu thao tác của chương 5 và trình tự của chúng

Điều làm người đọc nhẹ người ở chương này

Chương này *không* có khái niệm toán mới nào. Mọi thứ dùng ở đây — decibel, dBFS, đỉnh, RMS, dải động — đều đã học ở bốn chương trước.

Cái mới là cách *viết* chúng ra thành mã. Vì thế ai còn thấy chương 4 có nhiều toán thì chương này sẽ nhẹ nhàng hơn nhiều.

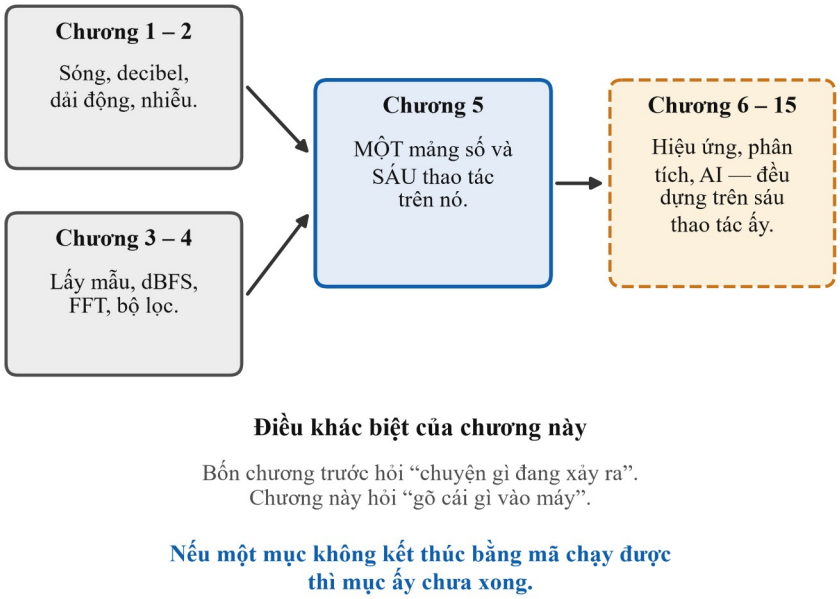
Đổi lại, chương này đòi hỏi một thứ mà bốn chương trước không đòi: phải *gõ*. Đọc mà không gõ thì không học được mục nào của chương này.

5.1.2 Sáu thao tác, và vì sao chỉ cần sáu

Nhìn vào một phần mềm biên tập âm thanh chuyên nghiệp thì thấy hàng trăm chức năng. Nhưng phần lớn công việc hằng ngày quy về sáu thao tác rất đơn giản, và cả sáu đều gói gọn trong một hai dòng numpy.

Bảng 5.1 Sáu thao tác của chương 5 và việc chúng làm

Thao tác	Việc thực tế nó làm	Mục
Đọc và ghi tệp	Đưa âm thanh vào chương trình và trả nó ra đĩa	5.3
Chỉnh gain	To lên hay nhỏ đi — theo decibel, không theo cảm tính	5.4
Trộn	Ghép nhạc nền với lời thoại, chồng nhiều lớp tiếng động	5.5
Cắt	Lấy một đoạn, bỏ phần thừa ở đầu và cuối	5.6
Ghép	Nối nhiều đoạn theo thứ tự mới	5.6
Chuẩn hoá	Đưa mức về một đích thống nhất trước khi giao bài	5.7



Hình 5.2 Bốn chương lý thuyết đọng lại thành một mảng số và sáu thao tác trên nó

Ghép sáu thao tác ấy lại là đã làm được: cắt một bản thu phỏng vấn, chèn nhạc nền dưới lời, cân mức giữa các đoạn, và xuất ra một tệp đúng chuẩn. Đó chính là công việc của một người dựng âm thanh cơ bản, và từ đây người đọc tự viết được chương trình làm việc ấy hàng loạt.

5.1.3 Cách học chương này

- **Gõ lại, đừng sao chép.** Mọi đoạn mã trong bài đều ngắn có chủ ý — đủ ngắn để gõ lại trong một hai phút. Gõ lại bằng tay thì mới phát hiện được chỗ chưa hiểu.
- **Chạy trên tệp của chính mình.** Hai tệp WAV đã dặn chuẩn bị ở cuối chương 4 là để dùng cho cả chương này. Chạy trên tệp của chính mình thì kết quả nghe được, và nghe được thì nhớ lâu.
- **Sai thì soi, đừng đoán.** Gần hết lỗi của chương này không báo lỗi khi chạy — chương trình vẫn xong, chỉ có âm thanh bị sai. Cách soi đúng: in ra hình dạng mảng, in ra đỉnh, và vẽ phổ đồ như đã học ở chương 4.

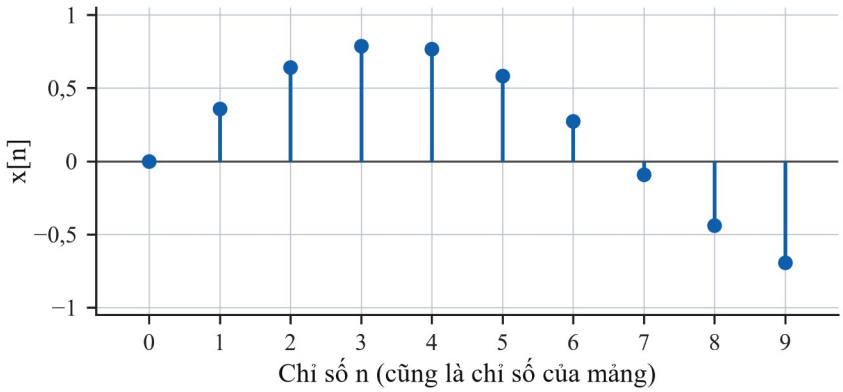
5.2 Mảng numpy cho âm thanh

5.2.1 Âm thanh là một mảng, và chỉ là một mảng

Chương 3 kết thúc bằng một câu: sau khi số hoá, âm thanh là một dãy số. Giờ nói chính xác hơn: nó là một mảng numpy — kiểu dữ liệu ndarray — cộng thêm một con số tần số lấy mẫu nằm riêng bên ngoài.

$$x:\ x[0], x[1], \ldots, x[N-1]\,; \qquad N = \lceil t \cdot f_s \rceil \tag{5.1}$$

Trong đó N là số mẫu, t là thời lượng tính bằng giây, f_s là tần số lấy mẫu. Dấu ngoặc đặc biệt nghĩa là làm tròn về số nguyên gần nhất — chi tiết nhỏ nhưng quan trọng, vì lệch một mẫu là lệch cả phép cắt về sau.



(a) Dãy mẫu vẽ ra

0,00	0,36	0,64	0,79	0,77	0,58	0,27	−0,09	−0,44	−0,69
0	1	2	3	4	5	6	7	8	9

`x.shape = (10,)` `x.dtype = float64`

(b) Cùng một thứ: mảng numpy trong bộ nhớ

Hình 5.3 Dãy mẫu vẽ ra và mảng numpy trong bộ nhớ là cùng một thứ

Ba con số phải biết về mọi mảng âm thanh

x.shape — hình dạng: bao nhiêu mẫu, bao nhiêu kênh.

x.dtype — kiểu số: float64, float32 hay một kiểu nguyên nào đó.

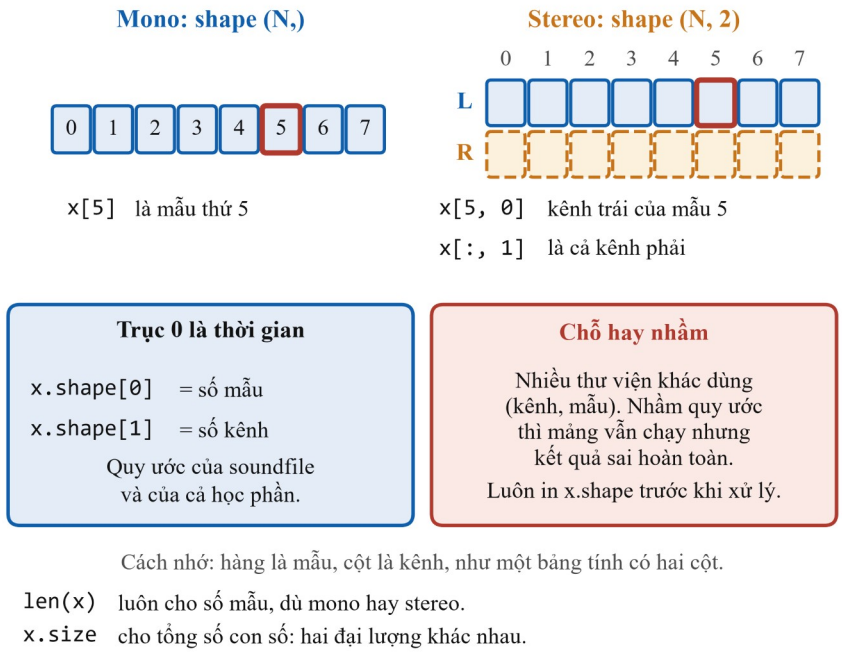
sr — tần số lấy mẫu. Con số này *không* nằm trong mảng; nó là một biến riêng, và nếu để lạc mất thì mảng trở nên vô nghĩa.

Thói quen nên có: mỗi khi nhận một mảng âm thanh từ đâu đó, in ngay ba thứ này ra trước khi làm bất cứ điều gì.

5.2.2 Mono và stereo — hình dạng của mảng

Một tệp mono cho ra mảng *một* chiều. Một tệp stereo cho ra mảng *hai* chiều, trong đó hàng là mẫu và cột là kênh.

$$\text{mono}: (N,) \qquad \text{stereo}: (N, 2), \qquad x_{\text{mono}}[n] = \frac{x_L[n] + x_R[n]}{2} \tag{5.2}$$



Hình 5.4 Mảng mono có shape (N,), mảng stereo có shape (N, 2)

Quy ước đặt trục thời gian ở trục 0 là quy ước của thư viện soundfile và của cả cuốn sách này. Nó không phải quy ước duy nhất — một số thư viện khác, trong đó có librosa ở chế độ mặc định, dùng ngược lại. Nhầm hai quy ước là lỗi âm thầm: chương trình vẫn chạy, mảng vẫn có đúng số phần tử, chỉ có kết quả là vô nghĩa.

Bảng 5.2 Các phép hay dùng trên mảng nhiều kênh

Việc cần làm	Cách viết	Chú ý
Đếm số kênh	x.shape[1] nếu x.ndim == 2, ngược lại là 1	Mảng mono *không* có shape[1]
Lấy riêng kênh trái	x[:, 0]	Cho ra mảng một chiều
Gộp về mono	x.mean(axis=1)	*Lấy trung bình*, không lấy tổng
Nhân đôi mono thành stereo	np.column_stack([x, x])	Hai kênh giống hệt nhau
Số mẫu	len(x) hoặc x.shape[0]	Luôn là số mẫu *mỗi* kênh, không phải tổng

Vì sao gộp mono phải lấy trung bình

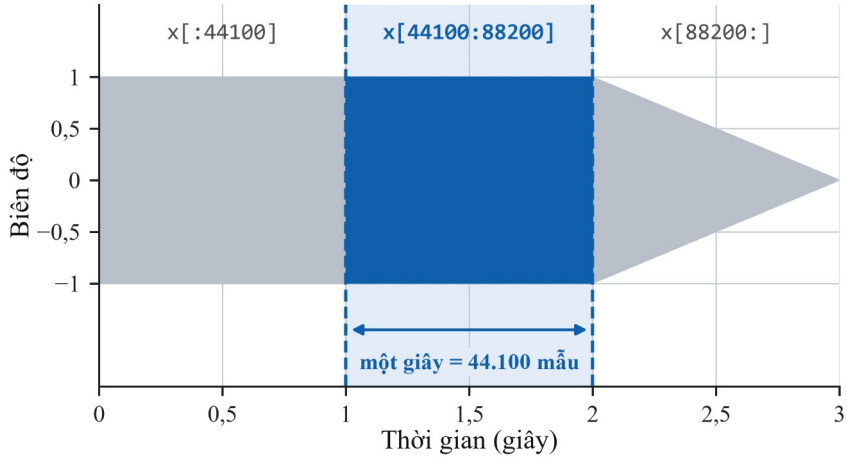
Nếu lấy *tổng* hai kênh thì với một tín hiệu stereo mà hai kênh gần giống nhau — trường hợp rất phổ biến — biên độ sẽ tăng gần gấp đôi và vượt toàn thang ngay.

Lấy trung bình thì biên độ giữ nguyên trong trường hợp ấy, và nhỏ đi trong các trường hợp khác — luôn an toàn.

Đây là ví dụ đầu tiên của một quy tắc sẽ gặp lại nhiều lần trong chương: mọi phép *cộng* tín hiệu đều có nguy cơ vượt thang, và phải nghĩ tới điều đó ngay lúc viết.

5.2.3 Cắt lát — cú pháp của mọi phép chọn đoạn

numpy cho phép lấy một đoạn liên tiếp của mảng bằng cú pháp cắt lát. Với âm thanh, phép này chính là phép chọn một khoảng thời gian.



Hình 5.5 Cắt lát bằng chỉ số mẫu, nhưng nghĩ bằng giây

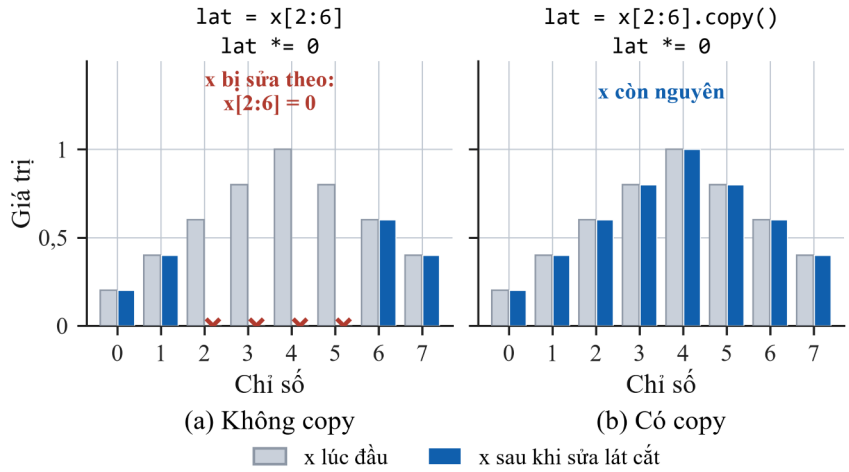
Điều cần luyện thành phản xạ: nghĩ bằng *giây*, viết bằng *chỉ số*, và đổi giữa hai thứ ấy bằng tần số lấy mẫu. Viết x[44100:88200] thì khó đọc; viết x[1*sr : 2*sr] thì đọc ra ngay là "từ giây thứ nhất tới giây thứ hai".

Một điểm dễ sai của numpy — view và copy

Phép cắt lát trong numpy *không* sao chép dữ liệu. Nó tạo ra một "cửa sổ" nhìn vào cùng vùng nhớ với mảng gốc.

Hệ quả: sửa lát cắt là sửa luôn mảng gốc. Với âm thanh, hậu quả rất khó lần ra — một đoạn giữa bản thu bỗng mất tiếng hoặc đổi mức, mà đoạn mã gây ra nó nằm ở chỗ khác hẳn.

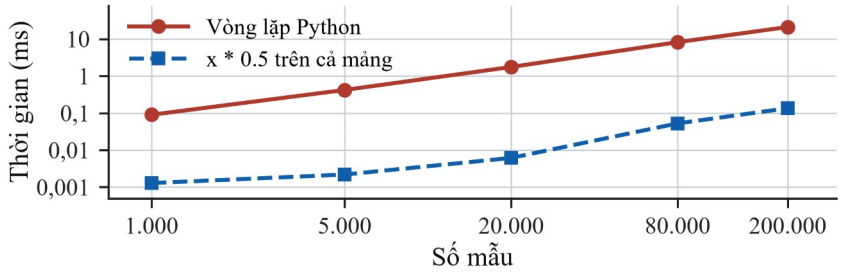
Quy tắc gọn: hễ định *sửa* một lát cắt thì gọi .copy(); chỉ *đọc* thì không cần. Cái giá của .copy() là bộ nhớ, và với âm thanh thì bộ nhớ ấy đáng bỏ ra.



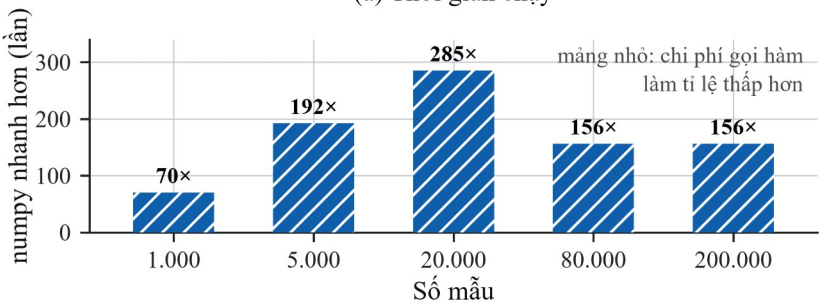
Hình 5.6 Cùng một phép sửa lát cắt: không copy thì mảng gốc bị sửa theo

5.2.4 Vector hoá — quy tắc quan trọng nhất của phần lập trình cơ bản

Người quen lập trình thông thường sẽ viết một vòng lặp để nhân từng mẫu. Cách ấy đúng về kết quả nhưng sai về phương pháp, và sai tới mức không dùng được trong thực tế.



(a) Thời gian chạy



(b) Tỉ lệ giữa hai cách

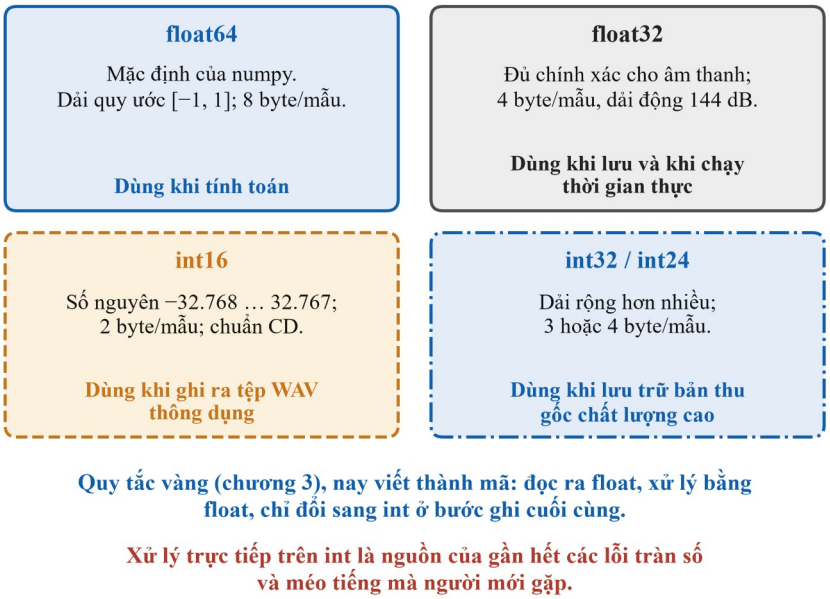
Hình 5.7 Cùng một phép nhân viết hai cách: vòng lặp Python và phép trên cả mảng

Con số ở hình trên đo trên chính máy đang dựng sách này, và tỉ lệ chênh lệch phụ thuộc máy — nhưng bậc độ lớn thì luôn như vậy: hàng trăm lần. Lý do là numpy chuyển phép tính xuống mã máy đã tối ưu sẵn, còn vòng lặp Python phải diễn giải từng bước một.

> **Quy tắc của cả phần lập trình cơ bản**
>
> *Không* viết vòng lặp trên từng mẫu. Nếu bạn đang gõ "for n in range(len(x))" thì gần như chắc chắn là có cách viết tốt hơn.
>
> Một bài hát bốn phút stereo có hơn hai mươi triệu mẫu. Với vòng lặp Python thì một phép nhân đơn giản đã mất vài giây; trên cả mảng thì vài chục mili giây.
>
> Ngoại lệ duy nhất chấp nhận được là khi thuật toán thật sự cần trạng thái từ mẫu trước — ví dụ phương trình sai phân của bộ lọc IIR ở chương 4. Ngay cả khi ấy, cách đúng vẫn là gọi hàm thư viện thay vì tự lặp.

5.2.5 Kiểu số — và quy tắc vàng viết thành mã

Chương 3 đã phát biểu quy tắc vàng về số nguyên và số thực. Giờ viết nó ra thành thao tác cụ thể.



Hình 5.8 Bốn kiểu số gặp trong xử lý âm thanh và chỗ dùng của mỗi kiểu

Bảng 5.3 Bốn kiểu số và đặc tính của chúng

Kiểu	Byte mỗi mẫu	Dải giá trị	Dùng ở đâu
float64	8	quy ước [−1, 1]	Mặc định của numpy — dùng khi tính toán
float32	4	quy ước [−1, 1]	Đủ chính xác cho âm thanh; dùng khi lưu và khi chạy thời gian thực
int16	2	−32.768 … 32.767	Chuẩn CD; định dạng WAV phổ biến nhất
int24 / int32	3 / 4	rộng hơn nhiều	Bản thu gốc chất lượng cao

Quy tắc vàng, viết thành ba bước

Đọc ra float. Thư viện soundfile làm sẵn việc này — mặc định nó trả về float64 trong dải [−1, 1] dù tệp gốc là kiểu gì.

Xử lý bằng float. Mọi phép nhân, cộng, cắt, ghép đều làm trên float. Ở đây không có chuyện tràn số, và sai số làm tròn nhỏ tới mức không nghe được.

Chỉ *đổi* sang int ở bước *ghi* cuối cùng, và để thư viện tự làm. Không bao giờ tự nhân với 32.767 rồi ép kiểu bằng tay — đó là chỗ sinh ra gần hết các lỗi méo tiếng của người mới.

Liệt kê 5.1 Xem một mảng âm thanh có gì trước khi làm bất cứ điều gì

```
import numpy as np
import soundfile as sf


def mo_ta(x, sr, ten="mang"):
    """In ba con so phai biet ve moi mang am thanh."""
```

```
    kenh = 1 if x.ndim == 1 else x.shape[1]
    print(f"{ten}: shape={x.shape}  dtype={x.dtype}")
    print(f"  {kenh} kenh, {sr} Hz, {len(x) / sr:.2f} giay")
    print(f"  dinh  = {np.abs(x).max():.4f}"
          f"  ({20 * np.log10(max(np.abs(x).max(), 1e-9)):.1f} dBFS)")
    print(f"  RMS   = {np.sqrt((x ** 2).mean()):.4f}")

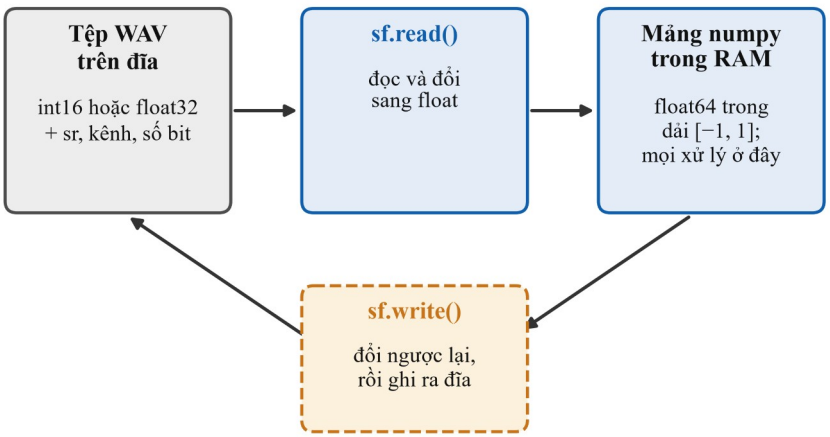
x, sr = sf.read("giong_doc.wav")
mo_ta(x, sr, "giong doc")
```

Liệt kê 5.1 nên trở thành thói quen đầu tiên của mọi chương trình âm thanh. Nó tốn ba dòng gọi và tiết kiệm hàng giờ tìm lỗi: một mảng có shape lạ, một dtype không mong đợi, hay một đỉnh bằng 0 đều lộ ra ngay ở đây thay vì lộ ra sau mười phép xử lý.

5.3 Đọc và ghi tệp WAV

5.3.1 Đường đi của một tệp

Trước khi gõ dòng nào, cần thấy rõ đường đi: tệp nằm trên đĩa ở một kiểu số nào đó, thư viện đọc nó lên và *đổi* sang float, mọi xử lý diễn ra trên float trong bộ nhớ, rồi ở bước cuối thư viện đổi ngược lại và ghi ra đĩa.



Hình 5.9 Từ tệp trên đĩa tới mảng trong RAM và ngược lại

Điểm cần nhớ nhất trong sơ đồ này nằm ở góc phải: *Mảng không tự nhớ tần số lấy mẫu*. Nó chỉ là một dãy số. Tần số lấy mẫu là một biến riêng mà chương trình phải tự giữ và tự truyền đi. Để lạc mất nó thì không có cách nào đoán lại được.

5.3.2 Đọc: ba dòng và bốn điều cần kiểm

Liệt kê 5.2 Đọc một tệp và kiểm ngay bốn thứ

```
import numpy as np
```

```
import soundfile as sf

x, sr = sf.read("nguon.wav")     # luon tra ve float64 [-1, 1]

assert x.ndim in (1, 2),      "hinh dang la"
assert sr in (44100, 48000),  f"tan so la: {sr}"
assert np.abs(x).max() <= 1.0, "vuot toan thang ngay tu tep"
assert len(x) > 0,            "tep rong"

print(f"{len(x) / sr:.2f} giay, {sr} Hz,",
      f"{1 if x.ndim == 1 else x.shape[1]} kenh")
```

Bốn dòng assert trong Liệt kê 5.2 trông thừa với một tệp tự tạo, nhưng chúng là điều bảo vệ người viết khi chương trình phải chạy trên hàng trăm tệp do người khác gửi tới. Mỗi dòng chặn một lỗi cụ thể đã từng làm sai cả một mẻ xử lý.

Bảng 5.4 Các tham số của sf.read và khi nào cần đụng tới

Tham số	Mặc định	Khi nào cần đổi
dtype	'float64'	Đặt 'float32' khi tệp rất dài và muốn tiết kiệm một nửa bộ nhớ
always_2d	False	Đặt True khi muốn mọi tệp đều ra mảng hai chiều, kể cả mono — giúp mã sau đó không phải phân nhánh
start, stop	cả tệp	Đọc một đoạn giữa tệp mà không nạp cả tệp vào bộ nhớ
frames	cả tệp	Đọc theo từng khối — cách xử lý tệp rất dài

5.3.3 Ghi: một dòng và ba quyết định

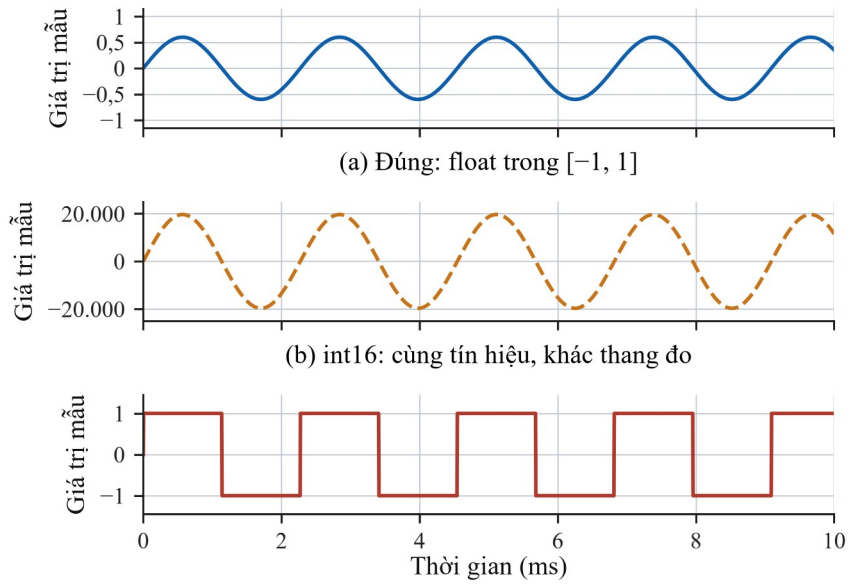
Liệt kê 5.3 Ghi ra tệp, có kiểm trước

```
dinh = np.abs(y).max()
if dinh > 1.0:
    raise ValueError(f"vuot toan thang: {dinh:.3f} — chua ghi")

sf.write("ket_qua.wav", y, sr, subtype="PCM_16")
```

Ba quyết định nằm trong đúng một dòng gọi ấy.

- **Tần số lấy mẫu.** Phải là con số đã đọc ra từ tệp gốc. Ghi sai thì tiếng nhanh hoặc chậm và cao độ lệch — lỗi nghe ra ngay nhưng người mới hay không hiểu vì sao.
- **Kiểu số.** PCM_16 cho tệp giao đi thông thường; PCM_24 cho bản gốc cần xử lý tiếp; *float* cho tệp trung gian giữa các bước trong cùng một dây chuyền.
- **Dải giá trị.** Mảng phải nằm gọn trong [−1, 1]. Thư viện không báo lỗi khi vượt — nó lặng lẽ kẹp, và tiếng ra bị rè.



Hình 5.10 Điều xảy ra khi lấy mảng int16 mà xử lý như thể nó là float

Vì sao không nên tự đổi kiểu bằng tay

Một cách viết hay gặp trên mạng: y_{int} = (y * 32767).astype(np.int16). Nó chạy được nhưng có ba vấn đề.

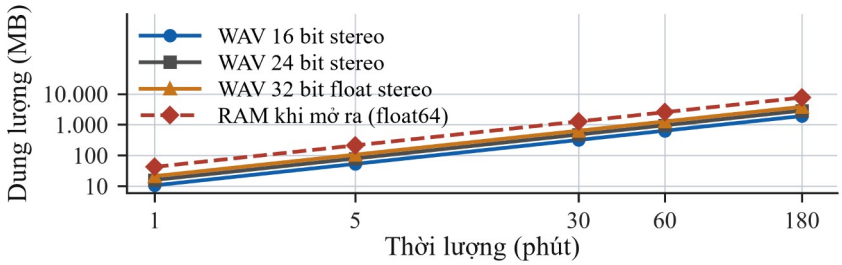
Thứ nhất, nó cắt cụt thay vì làm tròn, nên sai số gấp đôi mức cần thiết. Thứ hai, nếu y có mẫu nào vượt 1 thì phép ép kiểu tràn số và giá trị đảo dấu — tiếng ra thành nhiễu chứ không chỉ rè. Thứ ba, nó không áp dither, thứ mà chương 3 đã nói là cần khi hạ độ sâu bit.

Truyền subtype cho sf.write thì thư viện lo cả ba việc ấy. Đó là lý do quy tắc vàng nói "để thư viện tự làm".

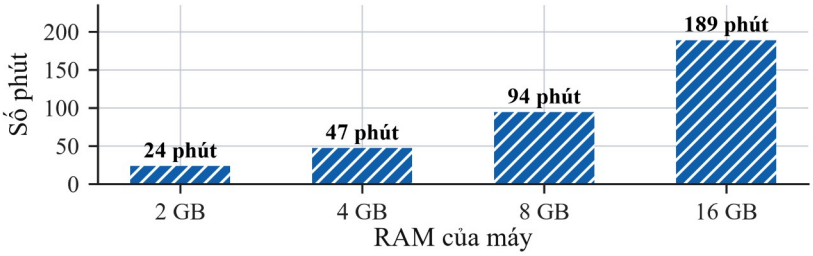
5.3.4 Dung lượng và bộ nhớ

$$S = f_s \cdot \frac{b}{8} \cdot N_{\text{kênh}} \cdot t\,; \qquad R_{\text{RAM}} = f_s \cdot 8 \cdot N_{\text{kênh}} \cdot t \quad [\text{byte}] \tag{5.3}$$

Công thức bên trái là dung lượng tệp trên đĩa, đã gặp ở chương 3. Công thức bên phải là điều mới của chương này: bộ nhớ mà mảng chiếm khi mở tệp ra. Con số 8 trong đó là 8 byte của một giá trị float64.



(a) Dung lượng trên đĩa và trong RAM



(b) Thời lượng tối đa mở cùng lúc (dùng một nửa RAM)

Hình 5.11 Dung lượng tệp, bộ nhớ khi mở ra, và giới hạn thực tế theo RAM của máy

Tỉ lệ bốn lần giữa RAM và tệp WAV 16 bit là hằng số, và nó có hệ quả rất thực tế. Một buổi thu ba tiếng chiếm 1,9 GB trên đĩa nhưng cần 7,6 GB bộ nhớ nếu mở một lần — và đó là chưa kể mỗi phép xử lý thường sinh thêm một bản sao nữa.

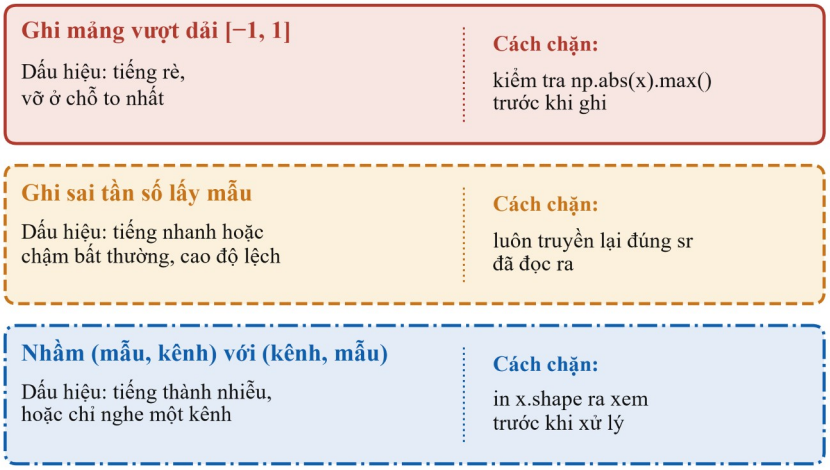
Ba cách xử lý tệp dài

Đọc một đoạn: sf.read(..., start=, stop=) chỉ nạp phần cần. Đủ cho phần lớn việc biên tập.

Đọc theo khối: sf.blocks(...) trả về từng khối một, xử lý xong khối nào thì ghi khối ấy. Đây là cách đúng cho tệp rất dài, và chương 6 sẽ dùng.

Đọc bằng float32: sf.read(..., dtype="float32") giảm một nửa bộ nhớ, và với âm thanh thì độ chính xác của float32 là thừa đủ — chương 2 đã cho thấy nó có dải động 144 dB.

5.3.5 Ba lỗi đọc ghi hay gặp



Hình 5.12 Ba lỗi đọc ghi, dấu hiệu nhận ra và cách chặn

Điểm chung của cả ba: chương trình chạy xong, không báo lỗi, và chỉ có âm thanh bị sai. Đây là đặc điểm của gần hết các lỗi trong lập trình âm thanh, và nó quyết định cách làm việc: phải *nghe* và phải *nhìn* phổ đồ sau mỗi bước, chứ không thể chỉ tin vào việc chương trình chạy trót lọt.

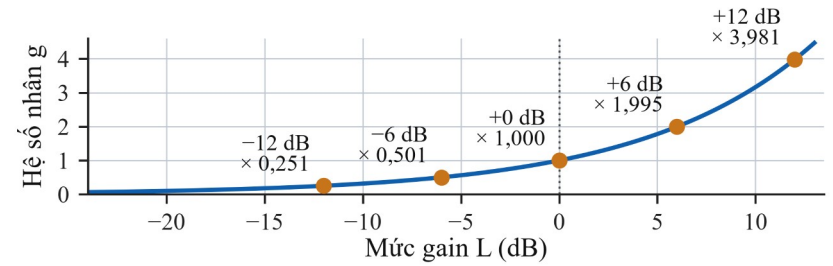
5.4 Chỉnh gain theo decibel

5.4.1 Một phép nhân duy nhất

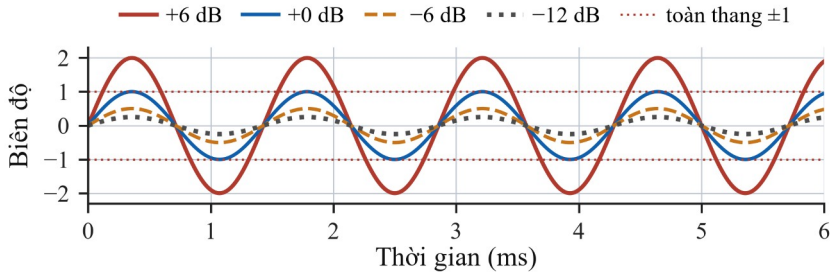
Chỉnh gain là thao tác đơn giản nhất của cả chương: nhân cả mảng với một hệ số. Cái khó không nằm ở phép nhân mà ở việc chọn con số, và ở việc biết điều gì xảy ra khi kết quả vượt thang.

$$y[n] = g \cdot x[n], \qquad g = 10^{L/20} \tag{5.4}$$

Trong đó L là mức gain tính bằng decibel và g là hệ số nhân. Đây chính là công thức decibel đã học ở chương 1, giờ dùng theo chiều ngược: từ decibel ra hệ số.



(a) Hệ số nhân theo mức gain: $g = 10^{L/20}$



(b) Cùng một tín hiệu ở bốn mức gain

Hình 5.13 Quan hệ giữa mức decibel và hệ số nhân, và tác dụng lên dạng sóng

Câu hỏi thực tế hay gặp hơn lại đi theo chiều ngược: biết đỉnh cần nằm ở đâu, hỏi phải chỉnh bao nhiêu decibel.

$$L_{\text{cần}} = L_{\text{đích}} - 20\log_{10} x_{\text{đỉnh}}\,; \qquad x_{\text{đỉnh}} = \max_n |x[n]| \tag{5.5}$$

Trong đó L_{dich} là mức đỉnh mong muốn tính bằng dBFS, và x_{dinh} là đỉnh hiện tại của tín hiệu. Đây chính là phép tính nằm bên trong mọi nút "normalize" của phần mềm biên tập, và mục 5.7 sẽ dùng lại nó.

Bảng 5.5 Vài mức gain nên thuộc lòng

Mức dB	Hệ số nhân	Nghe ra sao
-20 dB	0,100	nhỏ hẳn — như lùi ra xa
-12 dB	0,251	nhỏ rõ rệt
-6 dB	0,501	còn một nửa biên độ
-3 dB	0,708	còn một nửa công suất — thay đổi nhẹ
+0 dB	1,000	không đổi
+3 dB	1,413	to lên một chút
+6 dB	1,995	gấp đôi biên độ
+10 dB	3,162	nghe như to gấp đôi

Dòng cuối bảng đáng chú ý: gấp đôi *biên độ* là +6 dB, nhưng nghe như to gấp đôi thì cần khoảng +10 dB. Hai con số ấy khác nhau vì tai không nghe theo biên độ — đây chính là nội dung mục về độ to cảm nhận ở chương 1.

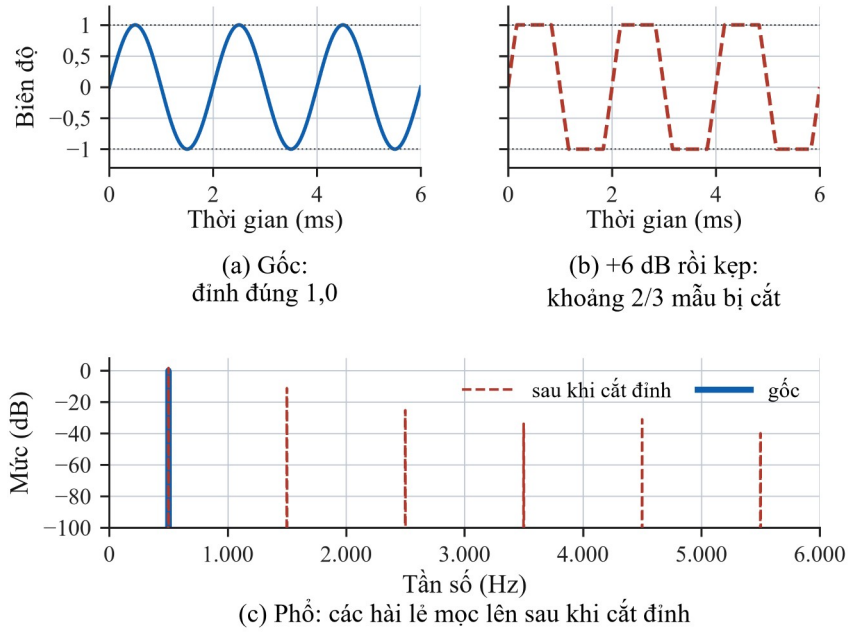
Liệt kê 5.4 Hàm chỉnh gain dùng lại được

```
def gain_db(x, db):
    """Nhan tin hieu voi he so ung voi muc dB cho truoc."""
    return x * (10.0 ** (db / 20.0))
```

```
# vi du: ha nhac nen 12 dB de nhuong cho loi thoai
nhac_nen = gain_db(nhac_nen, -12.0)
```

5.4.2 Cắt đỉnh — điều xảy ra khi vượt thang

Khi kết quả của phép nhân vượt ra ngoài [−1, 1], hai chuyện có thể xảy ra tuỳ vào việc đang làm gì. Nếu còn ở trong mảng float thì *không* có gì xảy ra cả — float chứa được giá trị lớn hơn 1 thoải mái. Chỉ tới bước ghi ra tệp hoặc phát ra loa thì giá trị mới bị *kẹp* về đúng ±1, và đó là lúc méo sinh ra.



Hình 5.14 Cắt đỉnh nhìn trên dạng sóng và nhìn trên phổ

Ô phải của hình trên là bằng chứng vì sao cắt đỉnh nghe rè. Phép kẹp biến sóng sin thành thứ gần sóng vuông, và như đã dựng ở chương 4, sóng vuông là tổng của cả một chuỗi hài lẻ. Những hài ấy không hề có trong tín hiệu gốc — chúng do chính phép kẹp sinh ra, và không cách nào gỡ bỏ về sau.

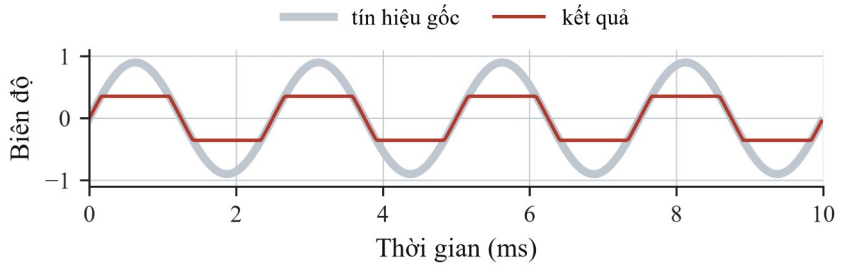
Liệt kê 5.5 Đếm số mẫu bị cắt đỉnh trước khi ghi

```
def kiem_cat_dinh(x, ten="tin hieu"):
    """Bao cao so mau vuot toan thang. Tra ve True neu sach."""
    d = int(np.sum(np.abs(x) > 1.0))
    if d == 0:
        print(f"{ten}: sach, dinh {np.abs(x).max():.4f}")
        return True
    print(f"{ten}: {d} mau vuot ({100 * d / x.size:.3f} %),",
          f"dinh {np.abs(x).max():.3f}")
    return False
```

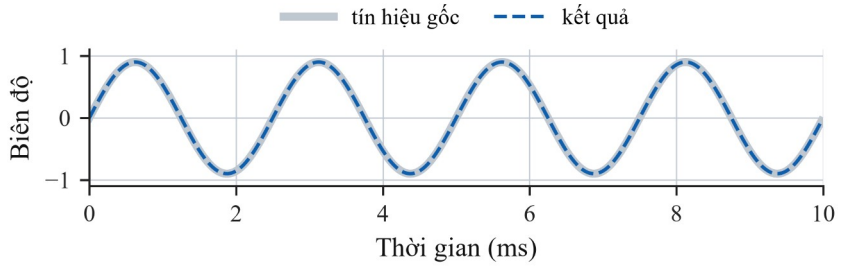
Liệt kê 5.5 nên được gọi sau *mọi* phép có thể làm tín hiệu to lên: sau chỉnh gain, sau trộn, sau lọc. Nó chạy tức thì và bắt được một loại lỗi mà nếu không kiểm thì chỉ phát hiện ra lúc nghe thấy tiếng rè trong bản đã giao.

5.4.3 Thứ tự các phép — khi nào nó quan trọng

Một câu hỏi tự nhiên: nếu hạ 9 dB rồi tăng lại 9 dB thì có về đúng như cũ không? Với số thực thì có, tới sai số máy. Nhưng chỉ khi giữa hai phép ấy không có bước nào *kẹp* giá trị.



(a) Tăng 9 dB → kẹp → hạ 9 dB (lệch tối đa 0,545)



(b) Hạ 9 dB → tăng 9 dB (lệch tối đa $\approx 10^{-16}$)

Hình 5.15 Cùng hai phép gain, hai thứ tự, hai kết quả hoàn toàn khác nhau

Quy tắc về thứ tự

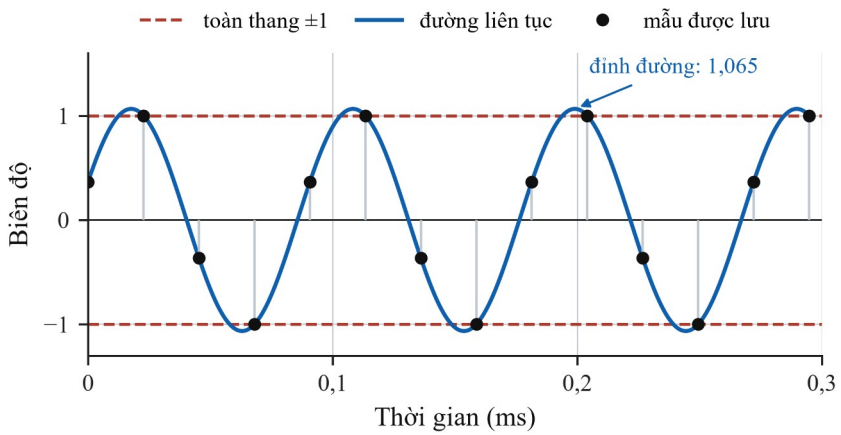
Khi làm việc bằng float và không có bước kẹp nào ở giữa, thứ tự các phép gain *không* quan trọng — chúng chỉ là các phép nhân, mà phép nhân thì giao hoán.

Nhưng ngay khi có một bước kẹp — ghi ra tệp int, phát ra loa, hay một hàm nào đó tự gọi np.clip — thì thứ tự trở nên quyết định, và phần bị kẹp mất vĩnh viễn.

Hệ quả thực hành: giữ toàn bộ dây chuyền xử lý trong float, và chỉ để nó chạm tới thang một lần duy nhất ở bước ghi cuối cùng. Đó cũng chính là lý do các phần mềm dựng chuyên nghiệp làm việc nội bộ bằng float 32 bit.

5.4.4 Đỉnh giữa hai mẫu — vì sao chuẩn hoá về 0 dBFS vẫn bị méo

Có một tình huống mà mọi phép kiểm ở trên đều báo sạch nhưng tiếng ra vẫn rè. Nó xảy ra vì một điều đã học ở chương 3 mà ít ai nối lại được với thực hành: bộ chuyển đổi ngược không phát ra dãy mẫu, nó phát ra *đường liên tục* đi qua các mẫu ấy.



Hình 5.16 Mọi mẫu đều đúng toàn thang, nhưng đường liên tục thì vượt lên trên

Ở hình trên, dãy mẫu đã được chuẩn hoá về đúng 0 dBFS nên không mẫu nào vượt thang và mọi phần mềm đều báo "không cắt đỉnh". Nhưng đường liên tục dựng lại từ chúng lên tới 1,065 — cao hơn toàn thang 0,54 dB. Phần cứng sẽ cắt đúng chỗ ấy.

Hai hệ quả thực hành

Chuẩn hoá đỉnh nên nhắm tới −1 dBFS chứ không phải 0 dBFS. Một decibel headroom ấy đủ cho phần lớn tín hiệu thật, và nó là quy ước của gần hết các chuẩn giao nộp.

Với tín hiệu đã qua nén có tổn hao — MP3, AAC — thì đỉnh liên mẫu có thể vượt tới vài decibel, vì phép giải mã dựng lại dạng sóng khác với bản gốc. Vì thế các chuẩn phát thanh thường yêu cầu −2 dBFS hoặc thấp hơn.

Chỉ số đo đúng đại lượng này gọi là "true peak", và các bộ đo chuyên nghiệp đều có nó. Cách nó làm chính là điều hình trên vẽ ra: nội suy giữa các mẫu rồi đo đỉnh của đường nội suy.

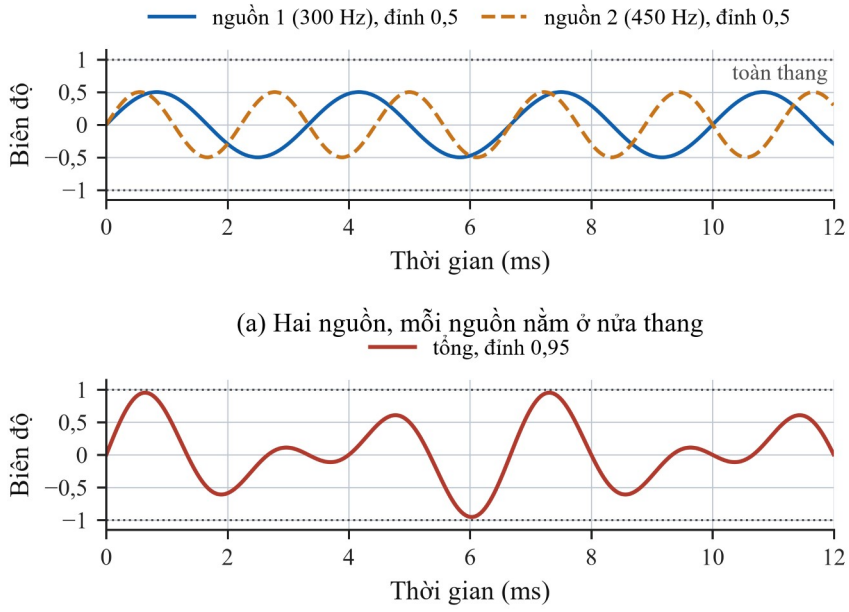
5.5 Trộn nhiều nguồn

5.5.1 Trộn là phép cộng — và đó là chỗ sinh ra vấn đề

Trộn hai nguồn âm thanh là cộng hai mảng lại. Không có gì phức tạp hơn thế, và đó cũng đúng với vật lý: hai sóng âm gặp nhau trong không khí thì áp suất của chúng cộng vào nhau.

$$y[n] = \sum_{i=1}^{K} x_i[n]\,; \qquad L_{\text{hạ}} = L_{\text{đích}} - 20\log_{10}\left(\sum_{i=1}^{K} p_i\right) \tag{5.6}$$

Vế trái là phép trộn. Vế phải là điều phải làm *trước* khi trộn: hạ mỗi nguồn xuống một mức để tổng của chúng không vượt thang. Trong đó p_i là đỉnh của nguồn thứ i, và L_{dich} là mức đích mong muốn cho bản trộn.



(a) Hai nguồn, mỗi nguồn nằm ở nửa thang

(b) Tổng của hai nguồn, gần chạm toàn thang

Hình 5.17 Hai nguồn đều nằm ở nửa thang, nhưng tổng thì gần chạm toàn thang

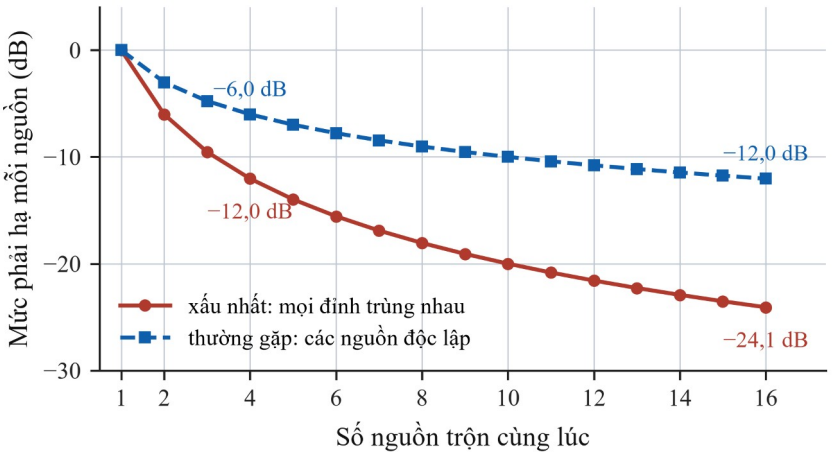
Điều làm người mới bất ngờ nhất

Trộn *không* lấy trung bình. Nó *cộng*.

Bốn nguồn cùng đỉnh 0,5 trộn lại thì đỉnh có thể lên tới 2,0 — gấp bốn lần mỗi nguồn, và gấp đôi toàn thang.

Bảng trộn của phần mềm dựng ẩn chuyện này đi bằng cách tự hạ mức, nên người dùng phần mềm ít khi gặp. Nhưng khi tự viết mã thì không có ai hạ hộ.

5.5.2 Cần hạ bao nhiêu trước khi trộn



Hình 5.18 Mức phải hạ mỗi nguồn theo số nguồn, ở hai giả định khác nhau

Bảng 5.6 Hai cách ước lượng và khi nào dùng cái nào

Cách ước lượng	Công thức	Khi nào dùng
Xấu nhất — cộng biên độ	hạ $20 \cdot \log_{10}(K)$ dB	Khi các nguồn có thể trùng đỉnh: nhiều bản sao của một tiếng, nhiều lớp cùng một nhịp
Thường gặp — cộng công suất	hạ $10 \cdot \log_{10}(K)$ dB	Khi các nguồn độc lập nhau: lời thoại, nhạc nền, tiếng động môi trường

Với bốn nguồn, cách xấu nhất bảo hạ −12,0 dB còn cách thường gặp chỉ bảo hạ −6,0 dB. Khoảng cách sáu decibel ấy chính là phần headroom mà người làm nghề chấp nhận rủi ro để đổi lấy độ to. Cách làm đúng nằm ở giữa: hạ theo ước lượng thường gặp, rồi *đo* đỉnh thật của bản trộn và chỉnh lại lần cuối.

Liệt kê 5.6 Trộn nhiều nguồn có kiểm mức

```
def tron_an_toan(nguon, dinh_dich_db=-1.0):
    """Tron nhieu mang, tu ha muc de tong khong vuot thang."""
    n = max(len(x) for x in nguon)
    tong = np.zeros(n)
    for x in nguon:
        tong[:len(x)] += x              # tu dem zero cho ngan hon

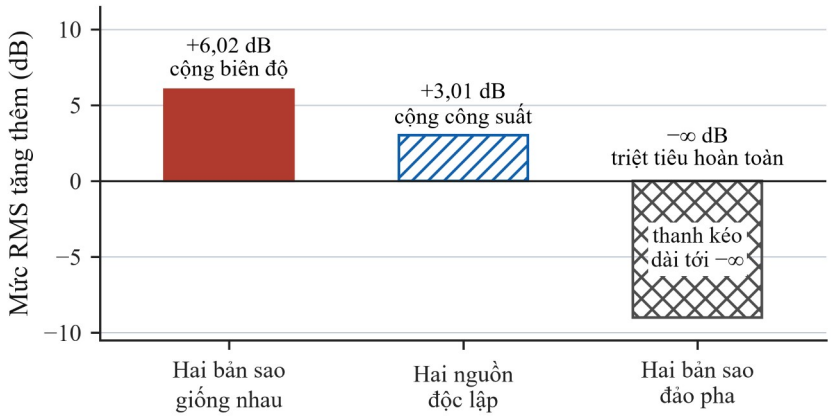
    dinh = np.abs(tong).max()
    if dinh <= 0:
        return tong
    can = dinh_dich_db - 20 * np.log10(dinh)
    print(f"dinh sau tron {20 * np.log10(dinh):.1f} dBFS",
          f"-> chinh {can:+.1f} dB")
    return tong * (10 ** (can / 20))
```

Cách viết trong Liệt kê 5.6 khác cách ở công thức (5.6) một chỗ đáng chú ý: nó trộn *trước* rồi mới đo và chỉnh, thay vì ước lượng rồi hạ từng nguồn. Làm được như thế vì đang ở trong float, nơi giá trị vượt 1 không gây hại gì. Đây là ví dụ rõ nhất cho lợi ích của quy tắc vàng.

5.5.3 Hai nguồn cùng mức trộn lại thì to lên bao nhiêu

Câu hỏi tưởng đơn giản nhưng không có đáp số duy nhất, và biết vì sao là một phần của nghề.

$$L_{\text{tươngquan}} = 20\log_{10}(a_1 + a_2)\,; \qquad L_{\text{độclập}} = 20\log_{10}\sqrt{a_1^2 + a_2^2} \tag{5.7}$$



Hình 5.19 Ba trường hợp trộn hai nguồn cùng mức, ba kết quả hoàn toàn khác nhau

- **Hai bản sao y nhau** — biên độ cộng thẳng, được đúng +6,02 dB. Đây là trường hợp nhân đôi một track.
- **Hai nguồn độc lập** — công suất cộng, chỉ được +3,01 dB. Đây là trường hợp thường gặp nhất: lời thoại và nhạc nền, hai micro khác nhau.
- **Hai bản sao đảo pha** — triệt tiêu sạch, còn đúng số không. Nghe có vẻ nhân tạo, nhưng nó là nguyên nhân thật của một lỗi rất phổ biến.

Trường hợp thứ ba không nhân tạo như chúng ta tưởng

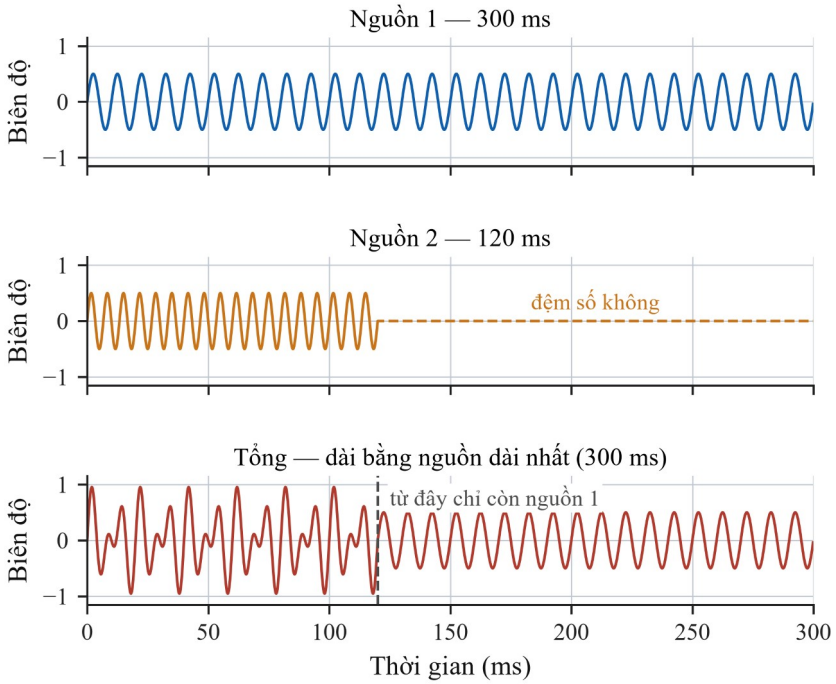
Nếu hai kênh của một bản stereo bị đảo pha so với nhau — do đấu sai dây, do đặt hai micro đối diện nhau, hay do một hiệu ứng nới rộng không gian làm quá tay — thì bản ấy nghe rất hay trên tai nghe stereo.

Nhưng khi phát trên một loa mono, hoặc khi ai đó gộp nó về mono để đưa lên một nền tảng nào đó, hai kênh cộng vào nhau và triệt tiêu. Tiếng biến mất hoặc mỏng hẳn đi.

Đây gọi là vấn đề tương thích mono, và cách kiểm rất gọn: gộp về mono rồi nghe. Chương 8 sẽ quay lại khi bàn về không gian stereo.

5.5.4 Trộn hai đoạn khác độ dài

numpy không tự cộng được hai mảng khác độ dài — nó báo lỗi ngay. Người viết phải tự quyết định làm gì, và có đúng hai lựa chọn.



Hình 5.20 Trộn hai đoạn khác độ dài: đệm số không cho cái ngắn

– **Đệm số không cho cái ngắn.** Giữ trọn cả hai nguồn; bản trộn dài bằng cái dài nhất. Gần như luôn chọn cách này.
– **Cắt cụt cái dài.** Bản trộn dài bằng cái ngắn nhất, và phần đuôi của nguồn dài mất hẳn. Chỉ chọn khi thật sự muốn thế — ví dụ khi nhạc nền cố ý dừng cùng lúc với lời.

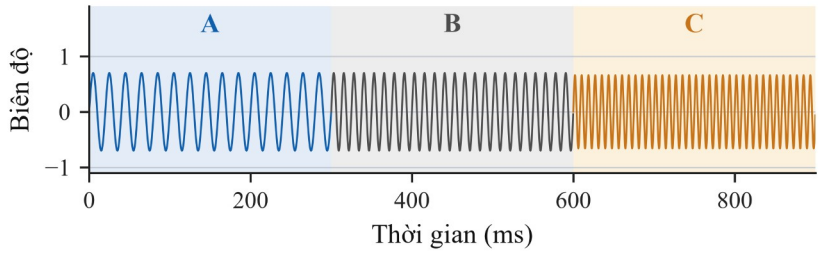
Nếu một nguồn cần bắt đầu *muộn* hơn thay vì cùng lúc, cách làm là đệm số không vào *đầu* nó chứ không phải vào cuối. Đây chính là phép trễ, và chương 6 sẽ nói kỹ.

5.6 Cắt và ghép

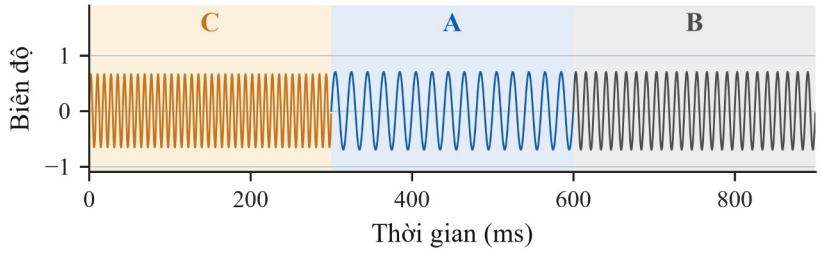
5.6.1 Hai phép, và cả nghề dựng phi tuyến

Toàn bộ việc dựng phi tuyến — cắt cảnh, đổi thứ tự, bỏ đoạn thừa, lặp một đoạn — quy về đúng hai phép: cắt lát một mảng và nối nhiều mảng lại.

$$i = \lceil t_0 \cdot f_s \rceil , \quad j = \lceil t_1 \cdot f_s \rceil ; \qquad N_{\text{ghép}} = \sum_{i=1}^{K} N_i \tag{5.8}$$



(a) Bản gốc, cắt thành ba đoạn A, B, C



(b) Sau khi ghép lại theo thứ tự C – A – B

Hình 5.21 Ba đoạn cắt ra rồi ghép lại theo thứ tự mới

Liệt kê 5.7 Cắt theo giây và ghép lại theo thứ tự mới

```
def cat(x, t0, t1, sr):
    """Lay doan [t0, t1) tinh bang giay."""
    i = max(0, round(t0 * sr))
    j = min(len(x), round(t1 * sr))
    return x[i:j].copy() if j > i else x[:0]

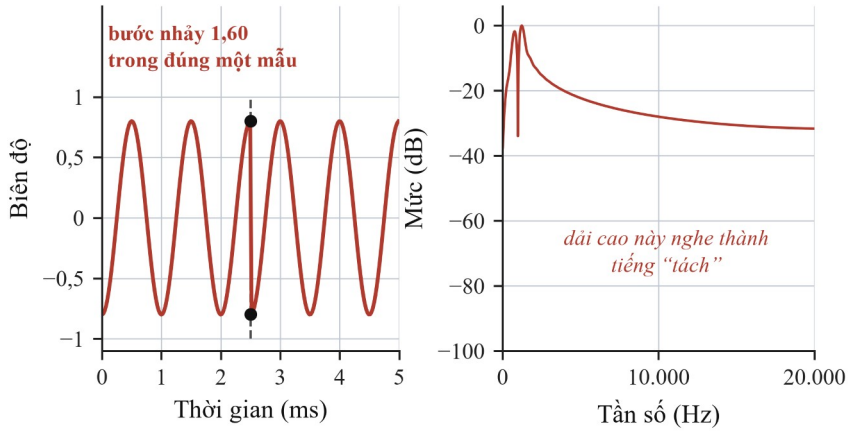
A = cat(x, 0.0, 0.3, sr)
B = cat(x, 0.3, 0.6, sr)
C = cat(x, 0.6, 0.9, sr)

moi = np.concatenate([C, A, B])       # doi thu tu
print(f"{len(moi) / sr:.2f} giay")
```

Hai chi tiết nhỏ trong Liệt kê 5.7 đáng chú ý. Dòng max và min chặn trường hợp t0 âm hoặc t1 vượt quá cuối tệp — không có chúng thì numpy trả về âm thầm mảng rỗng hoặc mảng ngắn hơn mong đợi. Lời gọi .copy() thì tách hẳn ba đoạn khỏi mảng gốc, đúng theo quy tắc ở mục 5.2.3.

5.6.2 Tiếng "tách" ở chỗ nối

Chạy đoạn mã trên rồi nghe kết quả thì sẽ nhận ra ngay một vấn đề: ở mỗi chỗ nối có một tiếng "tách" nhỏ nhưng rất khó chịu. Nguyên nhân của nó nhìn thấy được trên dạng sóng.



(a) Dạng sóng quanh chỗ nối (b) Phổ biên độ

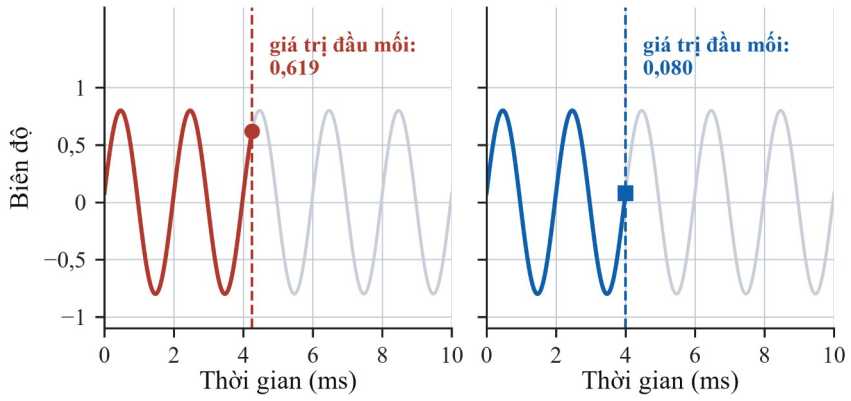
Hình 5.22 Chỗ nối có bước nhảy đột ngột, và bước nhảy ấy rải năng lượng khắp dải cao

Đoạn trước kết thúc ở một giá trị nào đó, đoạn sau bắt đầu ở một giá trị khác hẳn. Giữa hai mẫu liền nhau, tín hiệu nhảy một bước lớn. Như đã học ở chương 4, một bước nhảy đột ngột là một xung, và xung chứa mọi tần số — tai nghe chuỗi tần số cao ấy thành tiếng "tách".

> **Vì sao phải nói chuyện này ở chương 5 mà chữa ở chương 6**
>
> Cách chữa dứt điểm là fade rất ngắn ở hai đầu mỗi đoạn — chỉ vài mili giây, không nghe thấy nhưng đủ để không còn bước nhảy nào.
>
> Fade là một phép nhân với một đường bao, và nó thuộc về nhóm thao tác của chương 6 cùng với trễ và đảo thời gian.
>
> Nêu vấn đề ở đây mà không chữa ngay là có chủ ý: người đọc cần nghe thấy tiếng tách trên chính bản dựng của mình trước, thì lời giải ở chương sau mới có ý nghĩa.

5.6.3 Một giải pháp tạm: cắt ở điểm cắt không

Có một cách giảm nhẹ tiếng tách mà không cần fade: chọn chỗ cắt rơi đúng vào lúc sóng đi qua giá trị không.



(a) Cắt tuỳ tiện (b) Cắt ở điểm cắt không

Hình 5.23 Cắt tuỳ tiện và cắt ở điểm cắt không: giá trị ở đầu mối khác hẳn nhau

Liệt kê 5.8 Tìm điểm cắt không gần nhất

```
def cat_khong_gan_nhat(x, i, huong=1):
```

```
    """Tim chi so gan i nhat ma song di qua khong (chieu len)."""
    x = x if x.ndim == 1 else x.mean(axis=1)
    len_qua = np.where((x[:-1] <= 0) & (x[1:] > 0))[0] + 1
    if len(len_qua) == 0:
        return i
    return int(len_qua[np.argmin(np.abs(len_qua - i))])

i = cat_khong_gan_nhat(x, round(2.5 * sr))
print(f"cat o mau {i} thay vi {round(2.5 * sr)},",
      f"lech {abs(i - round(2.5 * sr)) / sr * 1000:.2f} ms")
```

Giải pháp này chỉ khắc phục được một nửa

Với một tín hiệu đơn giản — một nốt đàn, một sóng sin — cắt ở điểm cắt không làm bước nhảy gần bằng không và tiếng tách biến mất hẳn.

Nhưng âm thanh thật gồm rất nhiều tần số cùng lúc, và không có thời điểm nào *tất cả* chúng cùng đi qua không. Chọn được điểm tốt cho thành phần trầm thì thành phần cao vẫn nhảy.

Vì thế nhiều phần mềm biên tập có nút "snap to zero crossing" nhưng vẫn áp thêm một fade rất ngắn. Giải pháp này đáng biết, nhưng nó không thay được lời giải thật.

5.6.4 Ghép nhiều mảng — ba điều phải khớp

Phép np.concatenate nối các mảng theo trục thời gian. Nó đòi ba điều phải khớp giữa các mảng, và không khớp thì hoặc báo lỗi hoặc cho kết quả sai.

Bảng 5.7 Ba điều phải khớp khi ghép

Điều phải khớp	Nếu không khớp	Cách chặn
Số kênh	numpy báo lỗi ngay — đây là trường hợp may mắn nhất	Đưa mọi đoạn về cùng số kênh trước khi ghép
Tần số lấy mẫu	numpy *không* biết gì về sr nên không báo. Kết quả: một đoạn phát nhanh hoặc chậm bất thường	Kiểm sr của mọi nguồn ngay lúc đọc
Kiểu số	numpy tự nâng kiểu, thường vô hại — trừ khi trộn float với int	Đọc mọi thứ ra float ngay từ đầu

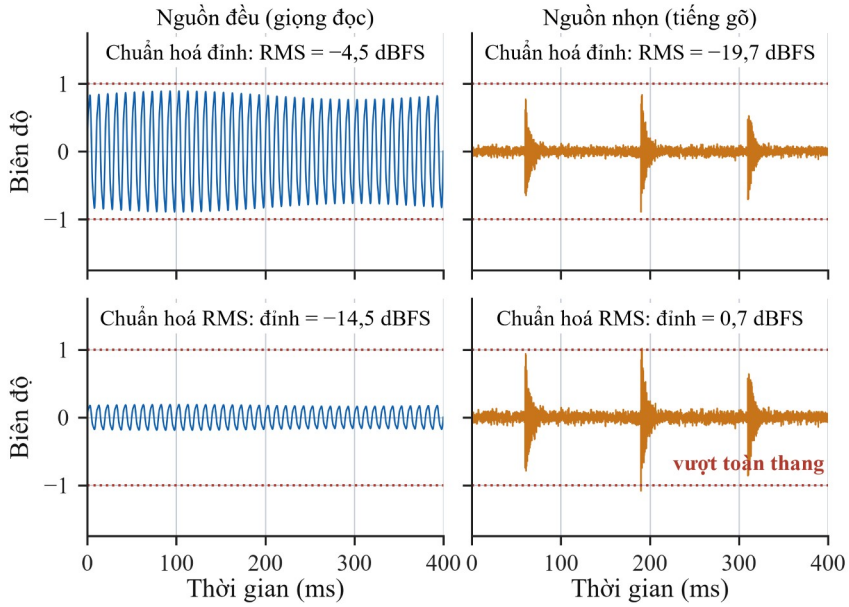
Điều thứ hai là nguy hiểm nhất vì nó không báo lỗi. Một đoạn thu ở 48.000 Hz ghép vào một bản 44.100 Hz sẽ phát nhanh hơn khoảng chín phần trăm và cao độ lệch gần một nửa cung — nghe ra được nhưng dễ đổ lỗi cho chuyện khác. Muốn ghép hai tệp khác tần số lấy mẫu thì phải lấy mẫu lại một cái, và đó là việc của chương 6.

5.7 Chuẩn hoá

5.7.1 Hai câu hỏi khác nhau, hai cách chuẩn hoá

Chuẩn hoá là đưa mức của một bản thu về một đích thống nhất. Nhưng "mức" có ít nhất hai nghĩa, và chọn nghĩa nào quyết định kết quả.

- **Chuẩn hoá đỉnh** trả lời câu: "dùng hết thang mà không vượt". Nó nhân cả tín hiệu để mẫu lớn nhất rơi đúng vào mức đích.
- **Chuẩn hoá RMS** trả lời câu: "nghe to bằng nhau". Nó nhân cả tín hiệu để mức trung bình bình phương rơi đúng vào mức đích.



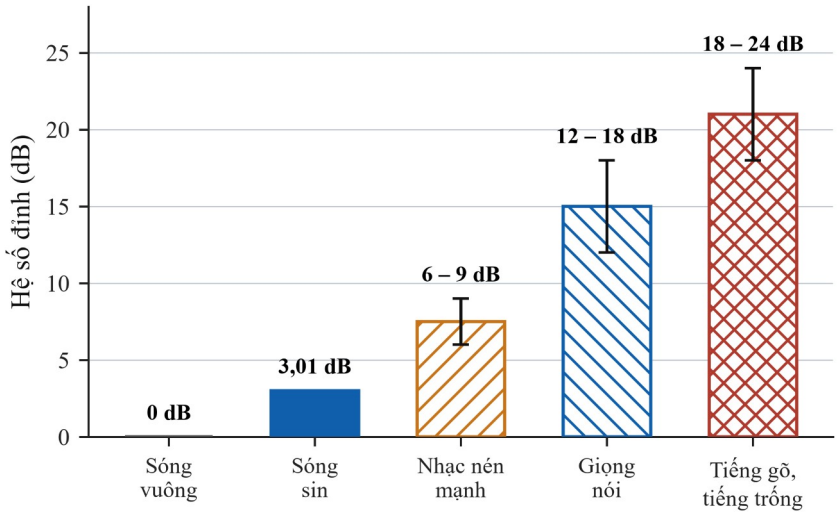
Hình 5.24 Cùng hai nguồn, hai cách chuẩn hoá, bốn kết quả khác hẳn nhau

Hàng trên của hình cho thấy vấn đề của chuẩn hoá đỉnh: hai nguồn cùng đỉnh −1 dBFS nhưng mức RMS lệch nhau tới mười lăm decibel — nghe ra thì cái này to hơn hẳn cái kia. Hàng dưới cho thấy vấn đề của chuẩn hoá RMS: hai nguồn nghe cân nhau, nhưng nguồn có nhiều đỉnh nhọn thì đỉnh đã vượt toàn thang.

5.7.2 Hệ số đỉnh — con số giải thích tất cả

$$C = 20\log_{10}\frac{x_{\text{đỉnh}}}{x_{\text{rms}}}; \qquad x_{\text{rms}} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x[n]^2} \tag{5.9}$$

Đại lượng C gọi là hệ số đỉnh, tính bằng decibel. Nó cho biết đỉnh cao hơn mức trung bình bao nhiêu — tức tín hiệu "nhọn" đến mức nào.



Hình 5.25 Hệ số đỉnh của năm loại nguồn âm điển hình

Bảng 5.8 Hệ số đỉnh tiêu biểu và ý nghĩa thực hành

Loại nguồn	Hệ số đỉnh	Nghĩa là
Sóng vuông	0 dB	Đỉnh bằng RMS. Chuẩn hoá đỉnh là nó to hết cỡ
Sóng sin	3,01 dB	Mốc lý thuyết, tính ra được
Nhạc đã nén mạnh	6 – 9 dB	Nghe to nhưng không còn động lực — đây là hệ quả của "cuộc chiến độ to" đã nhắc ở chương 2
Giọng nói	12 – 18 dB	Chuẩn hoá đỉnh thì phần lớn thời gian vẫn nhỏ
Tiếng gõ, tiếng trống	18 – 24 dB	Rất nhọn. Chuẩn hoá đỉnh gần như không làm nó to lên chút nào

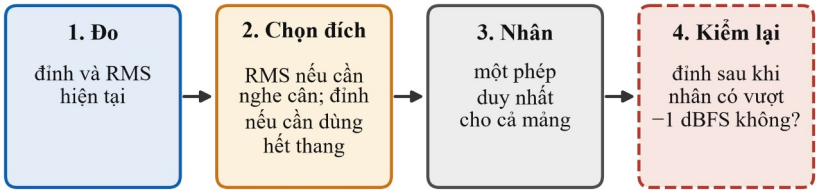
Cách chọn giữa hai kiểu chuẩn hoá

Nếu mục đích là *dùng hết thang* — chuẩn bị một tệp để xử lý tiếp, hoặc để đưa vào một hệ thống có dải động hạn chế — thì chuẩn hoá *đỉnh*, nhắm tới −1 dBFS.

Nếu mục đích là *nghe cân nhau* — cân mức giữa nhiều đoạn của một bản dựng, hoặc giữa nhiều tệp trong một tuyển tập — thì chuẩn hoá RMS, rồi kiểm lại đỉnh.

Nếu bản thu sẽ lên một nền tảng phát trực tuyến thì chuẩn phù hợp không phải cả hai, mà là LUFS — một phép đo độ to cảm nhận có trọng số tần số theo tai người. Chương 9 sẽ nói tới nó cùng với bộ nén.

5.7.3 Quy trình chuẩn hoá an toàn



Bước 4 là bước hay bị bỏ nhất — và là bước duy nhất bắt được lỗi.

Nếu chuẩn hoá RMS mà đỉnh vượt: hoặc hạ đích RMS xuống, hoặc dùng bộ nén (chương 9).

Không bao giờ chữa bằng cách kẹp: kẹp là méo, và méo thì không gỡ lại được.

Hình 5.26 Bốn bước của một phép chuẩn hoá an toàn

Liệt kê 5.9 Chuẩn hoá theo đỉnh hoặc theo RMS, có kiểm lại

```
def chuan_hoa(x, dich_db=-1.0, theo="dinh"):
    """Chuan hoa ve dich_db. theo = "dinh" hoac "rms"."""
    if theo == "dinh":
        muc = np.abs(x).max()
    else:
        muc = np.sqrt((x ** 2).mean())
    if muc <= 0:
        return x

    can = dich_db - 20 * np.log10(muc)
    y = x * (10 ** (can / 20))

    dinh_moi = 20 * np.log10(max(np.abs(y).max(), 1e-9))
    if dinh_moi > -0.1:
        print(f"CANH BAO: dinh sau chuan hoa {dinh_moi:.2f} dBFS")
    return y
```

Bước kiểm lại ở cuối Liệt kê 5.9 là bước hay bị bỏ nhất, và là bước duy nhất bắt được lỗi. Nó chỉ cần thiết khi chuẩn hoá theo RMS — chuẩn hoá theo đỉnh thì theo định nghĩa là không thể vượt. Nhưng để nó ở đó cho cả hai nhánh thì không tốn gì, và tránh được việc bỏ sót khi đổi tham số.

Ba cách sai khi đỉnh vượt sau chuẩn hoá RMS

Sai: kẹp bằng np.clip. Kẹp là méo, và méo thì không gỡ được — mục 5.4.2 đã cho thấy nó sinh ra cả một chuỗi hài lạ.

Sai: hạ toàn bộ xuống cho đỉnh vừa. Làm thế là quay lại chuẩn hoá đỉnh, và mất luôn mục đích ban đầu là nghe cân nhau.

Đúng: hạ đích RMS xuống một chút và thử lại; hoặc dùng bộ nén để kéo riêng các đỉnh nhọn xuống trước rồi mới chuẩn hoá. Bộ nén là nội dung chương 9.

5.7.4 Chuẩn hoá là bước cuối, không phải bước đầu

Một thói quen sai rất phổ biến: chuẩn hoá từng nguồn ngay sau khi đọc, rồi mới trộn và cắt ghép. Cách ấy sai ở hai chỗ.

– Chuẩn hoá từng nguồn về cùng một đỉnh sẽ xoá mất quan hệ mức giữa chúng — mà quan hệ ấy thường chính là thứ người thu đã cố ý tạo ra. Một tiếng thì thầm và một tiếng hét sau khi chuẩn hoá sẽ to bằng nhau.
– Sau khi trộn thì mức lại thay đổi hết, nên công chuẩn hoá lúc đầu thành công cốc, và vẫn phải chuẩn hoá lại lần nữa ở cuối.

Trình tự đúng: đọc — cân mức tương đối giữa các nguồn bằng gain — trộn, cắt, ghép — rồi *chuẩn hoá một lần* ở cuối, ngay trước khi ghi ra. Đó chính là dây chuyền mà hình tổng kết ở mục 5.9 vẽ ra.

5.8 Thực hành và bài tập

5.8.1 Ba bài thực hành

Bảng 5.9 Ba bài thực hành và tiêu chí hoàn thành

Bài	Việc phải làm	Coi là xong khi
TH1 — Đọc và mô tả	Đọc hai tệp WAV của chính mình, in ra shape, dtype, thời lượng, số kênh, đỉnh và RMS của mỗi tệp	Đọc được kết quả ra và nói được tệp nào to hơn, tệp nào nhọn hơn
TH2 — Ghép có nhạc nền	Cắt ba đoạn từ tệp giọng đọc, ghép lại theo thứ tự mới, trộn nhạc nền đã hạ 12 dB xuống dưới	Nghe được bản kết quả, và *chỉ ra* được chỗ nào có tiếng tách
TH3 — Chuẩn hoá và ghi	Chuẩn hoá bản vừa dựng về −1 dBFS theo đỉnh, rồi lại theo RMS −20 dBFS, ghi ra hai tệp	So được hai tệp và giải thích vì sao chúng nghe khác nhau

TH2 là bài quan trọng nhất

Nó ghép năm trong sáu thao tác của chương vào một việc có nghĩa, và nó kết thúc bằng một vấn đề chưa giải được.

Người đọc phải nghe thấy tiếng tách trên chính bản dựng của mình. Đó là điều kiện để chương 6 có ý nghĩa.

Ai không nghe ra thì thử ghép ở chỗ tiếng đang to, và thử nghe bằng tai nghe thay vì loa máy tính.

5.8.2 Bài tập

Bảy bài dưới đây. Bài 1 tới 5 cần cả mã lẫn kết quả in ra; bài 6 và 7 là bài viết.

Bài 1. Một tệp WAV stereo 44.100 Hz dài 3 phút 20 giây, 24 bit. Tính dung lượng tệp trên đĩa và bộ nhớ cần khi mở ra float64. Viết mã kiểm lại hai con số ấy bằng một tệp tự tạo.

Bài 2. Viết hàm nhận vào một mảng và trả về một chuỗi mô tả gồm: số kênh, thời lượng, đỉnh tính bằng dBFS, RMS tính bằng dBFS, và hệ số đỉnh. Chạy nó trên hai tệp của mình và giải thích chênh lệch hệ số đỉnh giữa chúng.

Bài 3. Một tín hiệu có đỉnh 0,32. Cần chỉnh bao nhiêu dB để đỉnh về −1 dBFS? Nếu sau đó trộn thêm ba nguồn khác cùng đỉnh ấy thì phải hạ mỗi nguồn bao nhiêu dB để bản trộn vẫn ở −1 dBFS, theo cả hai cách ước lượng ở mục 5.5.2?

Bài 4. Viết chương trình cắt ba đoạn bất kỳ từ một tệp, ghép lại theo thứ tự ngược, rồi *đo* độ lớn bước nhảy tại mỗi chỗ nối. In ra ba con số ấy và sắp xếp chúng theo độ lớn.

Bài 5. Lấy một tệp giọng đọc, tạo hai bản: một chuẩn hoá đỉnh về −1 dBFS, một chuẩn hoá RMS về −20 dBFS. In ra đỉnh và RMS của cả hai, và nói bản nào nghe to hơn. Sau đó thử chuẩn hoá RMS về −12 dBFS: đỉnh lúc này bằng bao nhiêu, có vượt thang không, và con số ấy liên quan thế nào tới hệ số đỉnh của tệp?

Bài 6. Một người mới học viết: đọc tệp, nhân 32.767, ép kiểu int16, nhân lại 2 để "to gấp đôi", rồi ghi ra. Chỉ ra *ba* lỗi trong quy trình ấy, giải thích hậu quả nghe được của mỗi lỗi, và viết lại quy trình cho đúng.

Bài 7. (tổng hợp) Nhận nhiệm vụ dựng một đoạn giới thiệu dài 30 giây gồm: lời dẫn thu sẵn, nhạc nền, và hai tiếng động chuyển cảnh. Trình bày trình tự các thao tác theo đúng thứ tự, nêu ở mỗi bước cần *đo* gì để biết bước ấy làm đúng, và nêu một rủi ro của mỗi bước.

> **Cách trình bày bài làm**
>
> Bài 1 tới 5 nên có một tệp .py chạy được và kết quả in ra. Mã phải chạy được trên máy khác mà không cần sửa đường dẫn — dùng đường dẫn tương đối.
>
> Mọi con số trong bài viết phải kèm đơn vị. Số thập phân dùng dấu phẩy; nghìn dùng dấu chấm.
>
> Bài 6 và 7 không có đáp số duy nhất. Hãy tự đánh giá theo tính hợp lý của lập luận và theo việc có nêu được rủi ro hay không.

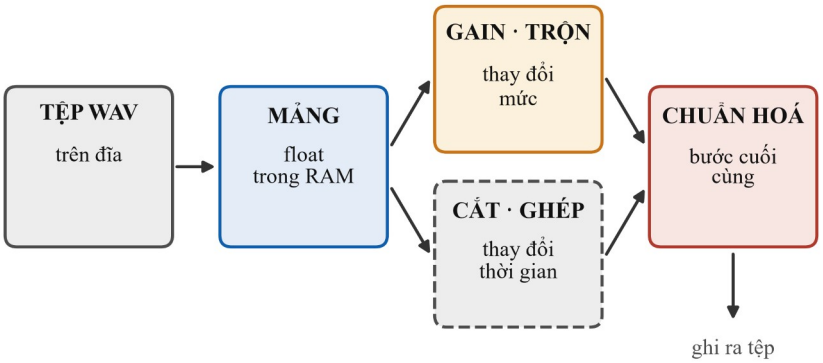
5.8.3 Bốn lỗi hay gặp nhất khi làm bài chương này

Bảng 5.10 Bốn lỗi hay gặp và cách tự phát hiện

Lỗi	Dấu hiệu	Cách chữa
Sửa lát cắt mà bỏ sót .copy()	Một đoạn giữa bản thu bỗng đổi mức hoặc mất tiếng	Gọi .copy() mỗi khi định sửa một lát cắt
Trộn xong không kiểm đỉnh	Tiếng rè ở chỗ to nhất của bản trộn	Gọi hàm đếm cắt đỉnh sau mọi phép có thể làm to lên
Chuẩn hoá từng nguồn trước khi trộn	Mất hết quan hệ mức; tiếng thì thầm to bằng tiếng hét	Chuẩn hoá *một lần* ở cuối
Ghép hai tệp khác tần số lấy mẫu	Một đoạn phát nhanh hoặc chậm bất thường	Kiểm sr của mọi nguồn ngay lúc đọc

5.9 Tổng kết chương

5.9.1 Dây chuyền sáu thao tác



Bốn quy tắc đi cùng dây chuyền này

1. Đọc ra float, xử lý bằng float, chỉ đổi sang int ở bước ghi.
2. Không viết vòng lặp trên từng mẫu; làm trên cả mảng.
3. Sau mỗi phép có thể làm to lên, phải đo lại đỉnh.
4. Chuẩn hoá là bước cuối, không phải bước đầu.

Hình 5.27 Sáu thao tác nối với nhau thành một dây chuyền, và bốn quy tắc đi cùng

Thứ tự trong sơ đồ này không tuỳ tiện. Đọc ra float ở đầu, xử lý toàn bộ trong float, chuẩn hoá một lần ở cuối, rồi mới đổi sang int lúc ghi. Mọi cách sắp xếp khác đều dẫn tới một trong bốn lỗi ở bảng trên.

5.9.2 Bảng tra nhanh

Đọc một tệp	`x, sr = sf.read("a.wav")`
Xem mảng có gì	`x.shape, x.dtype, len(x) / sr`
Gộp stereo về mono	`x.mean(axis=1)`
Hạ 6 dB	`x * 10 ** (-6 / 20)`
Trộn hai nguồn	`a + b  # phải cùng độ dài`
Lấy đoạn 1 – 2 giây	`x[1 * sr : 2 * sr]`
Nối hai đoạn	`np.concatenate([a, b])`
Chuẩn hoá đỉnh −1 dBFS	`x * 10 ** (-1 / 20) / np.abs(x).max()`
Ghi ra tệp 16 bit	`sf.write("b.wav", x, sr, subtype="PCM_16")`

Hình 5.28 Muốn làm gì thì gõ gì — chín dòng đủ cho phần lớn công việc

5.9.3 Năm điều phải nhớ

– **Âm thanh là một mảng, cộng một con số sr nằm riêng.** Mất sr là mất tất cả. In shape, dtype và sr ra trước khi làm bất cứ điều gì.

- **Đọc ra float, xử lý bằng float, chỉ đổi sang int ở bước ghi.** Và để thư viện tự đổi, đừng tự nhân với 32.767.
- **Không viết vòng lặp trên từng mẫu.** Làm trên cả mảng — nhanh hơn hàng trăm lần và ngắn hơn.
- **Trộn là cộng, không phải lấy trung bình.** Sau mỗi phép có thể làm to lên, đo lại đỉnh.
- **Chuẩn hoá là bước cuối.** Đỉnh nếu cần dùng hết thang, RMS nếu cần nghe cân — và luôn kiểm lại đỉnh sau khi chuẩn hoá RMS.

5.9.4 Tự kiểm trước khi sang chương 6

Tám câu dưới đây nên trả lời được không cần mở tài liệu.

- Một tệp stereo 44.100 Hz dài 10 giây cho mảng shape bằng bao nhiêu? (5.2.2)
- Vì sao gộp stereo về mono phải lấy trung bình chứ không lấy tổng? (5.2.2)
- Khi nào phải gọi .copy() trên một lát cắt? (5.2.3)
- Hạ 6 dB thì nhân với bao nhiêu? Còn hạ 20 dB? (5.4.1)
- Vì sao cắt đỉnh nghe rè — nói bằng ngôn ngữ của chương 4? (5.4.2)
- Trộn bốn nguồn cùng mức thì phải hạ mỗi nguồn bao nhiêu dB? (5.5.2)
- Vì sao chỗ nối khi ghép lại kêu "tách"? (5.6.2)
- Hai bản thu cùng đỉnh −1 dBFS có nghe to bằng nhau không, và vì sao? (5.7.2)

5.9.5 Chương sau

Chương 6 hoàn tất phần lập trình cơ bản và giải nốt vấn đề mà chương này để lại. Nội dung gồm: phép trễ, đảo thời gian, fade và đường bao chống click, xử lý theo khối cho tệp dài, và cách đóng gói những gì đã viết thành một mô-đun dùng lại được.

Chương 6 cũng là mốc đầu tiên của dự án thực hành. Người đọc nên bắt đầu nghĩ từ bây giờ về việc cần làm gì với những công cụ này.

Chuẩn bị trước chương sau gồm ba việc.

- Làm xong bài TH2 và *giữ lại* tệp kết quả có tiếng tách. Chương 6 sẽ chữa chính tệp ấy, nên có nó trong tay thì bài học có ý nghĩa hơn hẳn.
- Đọc lại mục 5.2.4 về vector hoá. Chương 6 dùng nó nhiều hơn chương này, và phần đường bao fade là ví dụ đẹp nhất của lối viết ấy.
- Chuẩn bị thêm một tệp WAV dài trên năm phút để thử phần xử lý theo khối. Một buổi thu phỏng vấn hay một bản ghi buổi họp đều dùng được.

5.9.6 Một việc nên tự làm trước chương sau

Sáu thao tác của chương này đã đủ để làm một việc hoàn chỉnh, và tự làm nó một lần là cách củng cố tốt hơn mọi bài tập lẻ.

Đề bài tự đặt: viết một chương trình nhận vào một thư mục chứa nhiều tệp WAV thu rời rạc, rồi tự động ghép chúng thành một tệp duy nhất, cân mức giữa các đoạn cho nghe đều nhau, và xuất ra một tệp 16 bit đã chuẩn hoá.

Bảng 5.11 Chương trình ấy dùng lại đúng những gì của chương này

Bước	Dùng mục nào
Duyệt thư mục, đọc từng tệp, kiểm sr và số kênh giống nhau	5.3.2
Đo RMS của từng tệp để biết tệp nào to tệp nào nhỏ	5.7.2
Chỉnh gain từng tệp về cùng một mức RMS	5.4.1 và 5.7.1
Ghép tất cả lại theo thứ tự tên tệp	5.6.1
Chuẩn hoá bản ghép về −1 dBFS theo đỉnh	5.7.3
Ghi ra tệp PCM_16, có kiểm đỉnh trước khi ghi	5.3.3

Chương trình ấy dài khoảng ba mươi dòng. Nó sẽ còn một khiếm khuyết — chỗ nối giữa các tệp kêu tách — và đó chính là thứ chương 6 chữa bằng đúng ba dòng nữa.

Tài liệu tham khảo của chương

Tài liệu của thư viện

- Tài liệu numpy — phần "NumPy: the absolute basics for beginners" và phần "Indexing on ndarrays". Hai phần này đủ cho toàn bộ mục 5.2, và nên đọc trước chương.
- Tài liệu python-soundfile — nguồn của mục 5.3. Đặc biệt chú ý phần về subtype và về sf.blocks.
- Tài liệu libsndfile — thư viện C nằm dưới soundfile, nơi liệt kê đầy đủ các định dạng và subtype được hỗ trợ.

Sách và giáo trình

- Smith, J. O. — Introduction to Digital Filters with Audio Applications. Nguồn nền về quy ước biên độ và dải [−1, 1].
- McFee, B. và cộng sự — tài liệu librosa. Không dùng trực tiếp ở chương này nhưng nên biết, vì nhiều tài liệu trên mạng dùng nó và nó có quy ước trục *ngược* với soundfile.
- Zölzer, U. (chủ biên) — DAFX: Digital Audio Effects, ấn bản 2, chương 1. Nguồn của phần quy ước xử lý và dây chuyền float.

Chuẩn và quy ước

- EBU R 128 và ITU-R BS.1770 — chuẩn đo độ to LUFS. Chưa dùng ở chương này nhưng là đích đến của phần chuẩn hoá; chương 9 sẽ nói tới.
- ITU-R BS.1770 phụ lục về true peak — nguồn của phần đỉnh liên mẫu ở mục 5.4.4.

Ghi chú về phiên bản thư viện

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

- np.trapz đổi tên thành np.trapezoid từ numpy 2.0; bài này không dùng nhưng tài liệu trên mạng thì có.
- soundfile phiên bản rất cũ không có tham số always_2d. Nếu thiếu thì tự xử lý bằng np.atleast_2d rồi chuyển vị.
- Trên một số bản Linux, soundfile cần cài thêm gói hệ thống libsndfile1. Nếu lệnh import báo lỗi thư viện C thì đó là nguyên nhân, không phải lỗi Python.

Một lưu ý về các con số trong bài

> **Cách đọc các con số**
>
> Các con số về hệ số nhân theo decibel, về dung lượng tệp và về bộ nhớ là kết quả tính chính xác, kiểm được bằng máy tính bỏ túi.
>
> Các con số đo thời gian chạy — phần vector hoá ở mục 5.2.4 — phụ thuộc vào máy. Bậc độ lớn thì luôn như trong bài, nhưng con số cụ thể trên máy của người đọc sẽ khác.
>
> Các con số về hệ số đỉnh của từng loại nguồn là giá trị *tiêu biểu* dùng để so sánh, không phải quy định. Mỗi bản thu một khác.
>
> Mọi phép tính và mọi đáp số bài tập trong bài đều sinh ra từ mô-đun sigtools.py, đã qua 201 phép tự kiểm.