TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
ĐẠI HỌC QUỐC GIA TP.HCM

BÁO CÁO ĐỒ ÁN CUỐI KÌ

PHƯƠNG PHÁP SỐ CHO KHOA HỌC DỮ LIỆU

ĐỀ TÀI: PHƯƠNG PHÁP FOURIER RỜI RẠC (DFT) ĐỂ KHỬ NHIỄU

Giảng viên hướng dẫn: T.S. Nguyễn Thị Hoài Thương

Nhóm thực hiện: Nhóm 4

Phan Thị Ngọc Linh	MSSV: 22280052
Phạm Bá Hoàng Anh	MSSV: 22280003
Trương Bình Ba	MSSV: 22280004
Quách Vũ Luân	MSSV: 22110110
Phạm Lê Hồng Đức	MSSV: 22280013
Bạch Ngọc Lê Duy	MSSV: 22280016
Ngô Thị Mỹ Duyên	MSSV: 22280017

TP. Hồ Chí Minh, tháng 6 năm 2025

Mục lục

1 Mở đầu

Trong đời sống hiện đại, việc trao đổi và truyền đạt thông tin không chỉ diễn ra qua văn bản hay hình ảnh, mà còn thông qua âm thanh - một phương tiện quan trọng trong giao tiếp, truyền thông, giáo dục, y tế, cũng như trong các lĩnh vực công nghệ như trợ lý ảo, nhận dạng giọng nói và điều khiển thiết bị bằng giọng nói.

Tuy nhiên, trong thực tế thì tín hiệu âm thanh thu được thường xuyên bị ảnh hưởng bởi nhiều loại nhiễu từ môi trường. Chẳng hạn, khi đang thưởng thức một bản nhạc yêu thích được ghi âm ngoài trời, nhưng bản ghi lại bị lẫn tiếng xe chạy, tiếng nói chuyện xung quanh hoặc tiếng quạt máy hoạt động. Những yếu tố nhiễu này không chỉ làm giảm chất lượng trải nghiệm âm thanh mà còn gây trở ngại lớn trong các hệ thống xử lý tín hiệu số. Đây cũng chính là tình huống mà nhóm chúng em đã tiến hành nghiên cứu: một file âm thanh bị nhiễu bởi các tạp âm môi trường, cần được làm sạch để tái tạo lại chất lượng gốc.

Trước những thách thức đã đặt ra trên, việc phát triển một giải pháp có khả năng tách biệt tín hiệu mong muốn khỏi các thành phần nhiễu trở nên vô cùng cần thiết. Đặc biệt trong các trường hợp nhiễu có đặc trưng tần số khác biệt rõ rệt so với tín hiệu chính thì việc khai thác đặc tính tần số của tín hiệu là một hướng tiếp cận rất tiềm năng. Một trong những công cụ toán học hiệu quả nhất trong trường hợp này là **Biến đổi Fourier Rời rạc (Discrete Fourier Transform - DFT)**. DFT cho phép chuyển đổi tín hiệu từ miền thời gian sang miền tần số, nơi mà các thành phần nhiễu hiện rõ dưới dạng những dải tần đặc trưng. Từ đó, ta có thể xác định và loại bỏ những tần số không mong muốn, góp phần khôi phục lại tín hiệu âm thanh sạch và dễ xử lý hơn.

Xuất phát từ những quan sát trên, nhóm chúng em đề xuất và hiện thực hóa ý tưởng xây dựng một phần mềm khử nhiễu âm thanh sử dụng DFT. Phần mềm được phát triển bằng ngôn ngữ Python, cho phép người dùng xử lý các tệp âm thanh bất kỳ với mục tiêu loại bỏ thành phần nhiễu không mong muốn và nâng cao chất lượng tín hiệu phục vụ cho các ứng dụng thực tiễn như ghi âm, truyền thông, và xử lý giọng nói.

Phần tiếp theo của báo cáo sẽ trình bày chi tiết cơ sở lý thuyết của DFT, quy trình thực hiện khử nhiễu từng bước, và mô hình thực nghiệm sử dụng dữ liệu ghi âm để minh họa khả năng ứng dụng của phương pháp.

2 Lý thuyết

2.1 Căn bậc n của đơn vị

Với số nguyên dương n, các số phức:

$$\{1, \omega, \omega^2, \ldots, \omega^{n-1}\}$$

trong đó:

$$\omega = e^{2\pi i/n} = \cos\frac{2\pi}{n} + i\sin\frac{2\pi}{n}$$

được gọi là **căn bậc n của đơn vị** vì chúng biểu diễn tất cả các nghiệm của phương trình $z^n = 1$.

Về mặt hình học, chúng là các đỉnh của đa giác đều n cạnh trên đường tròn đơn vị. Các căn này có tính chất tuần hoàn: nếu $k \geq n$, thì $\omega^k = \omega^{k \bmod n}$.

Trong biến đổi Fourier rời rạc (DFT), ta thường làm việc với các hàm mũ phức có dấu âm, do đó ta định nghĩa biến thể:

$$\xi = e^{-2\pi i/n} = \omega^{-1} = \cos\left(\frac{2\pi}{n}\right) - i\sin\left(\frac{2\pi}{n}\right) = \overline{\omega}$$

Tập hợp các lũy thừa của ξ:

$$\{1, \xi, \xi^2, \ldots, \xi^{n-1}\}$$

cũng tạo thành các căn bậc n của đơn vị, nhưng khác với trường hợp của ω, chúng được sắp xếp theo chiều kim đồng hồ trên đường tròn đơn vị trong mặt phẳng phức.

Tính chất quan trọng

Nếu k là số nguyên, thì từ $1 = |\xi^k|^2 = \xi^k\overline{\xi^k}$ suy ra:

$$\boxed{\xi^{-k} = \overline{\xi^k} = \omega^k.} \tag{2.1a}$$

Hơn nữa, từ đẳng thức:

$$\xi^k\left(1 + \xi^k + \xi^{2k} + \cdots + \xi^{(n-2)k} + \xi^{(n-1)k}\right) = \xi^k + \xi^{2k} + \cdots + \xi^{(n-1)k} + 1,$$

ta có:

$$\left(1 + \xi^k + \xi^{2k} + \cdots + \xi^{(n-1)k}\right)\left(1 - \xi^k\right) = 0.$$

Do đó:

$$\boxed{1 + \xi^k + \xi^{2k} + \cdots + \xi^{(n-1)k} = 0 \quad \text{khi} \quad \xi^k \neq 1.} \tag{2.1b}$$

(2.1a), (2.1b) Trích từ Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000, Chương 5, Mục 8, trang 358.

2.2 Ma trận Fourier

Ma trận Fourier cấp n là ma trận $n \times n$ có phần tử (j,k) là $\xi^{jk} = \omega^{-jk}$ với $0 \le j,k \le n-1$:

$$F_n = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \xi & \xi^2 & \cdots & \xi^{n-1} \\ 1 & \xi^2 & \xi^4 & \cdots & \xi^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \xi^{n-1} & \xi^{2(n-1)} & \cdots & \xi^{(n-1)^2} \end{pmatrix}$$

Tính chất của ma trận Fourier:

- Các cột của F_n trực giao với nhau
- Mỗi cột có độ dài là $\sqrt{n}$
- $\frac{1}{\sqrt{n}} F_n$ là ma trận unitary
- Ma trận nghịch đảo: $F_n^{-1} = \frac{1}{n}\overline{F_n}$

Chứng minh

Sử dụng (1) và (2), tích vô hướng của hai cột bất kỳ trong F_n (ví dụ cột thứ r và s) là:

$$\mathbf{F}_{*r}^* \mathbf{F}_{*s} = \sum_{j=0}^{n-1} \xi^{jr}\overline{\xi^{js}} = \sum_{j=0}^{n-1} \xi^{j(s-r)} = 0.$$

Do đó, **các cột của F_n trực giao với nhau**. Hơn nữa, mỗi cột **có chuẩn** $\sqrt{n}$ vì:

$$\|\mathbf{F}_{*k}\|_2^2 = \sum_{j=0}^{n-1} |\xi^{jk}|^2 = \sum_{j=0}^{n-1} 1 = n.$$

Vì vậy, $(1/\sqrt{n})F_n$ là **ma trận unitary**. Kết hợp với $F_n^T = \overline{F_n}$, ta có:

$$\left(\frac{1}{\sqrt{n}} F_n\right)^{-1} = \left(\frac{1}{\sqrt{n}} F_n\right)^* = \frac{1}{\sqrt{n}}\overline{F_n},$$

Suy ra **ma trận nghịch đảo**:

$$F_n^{-1} = \frac{1}{n}\overline{F_n} = \frac{1}{n}\begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{n-1} \\ 1 & \omega^2 & \omega^4 & \cdots & \omega^{n-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{n-1} & \omega^{n-2} & \cdots & \omega \end{pmatrix}_{n\times n} = \frac{1}{n}\overline{F_n}.$$

Ví dụ 1: Ma trận Fourier bậc 2 và bậc 4 được cho bởi:

$$F_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad F_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{pmatrix}$$

và các ma trận nghịch đảo của chúng là:

$$F_2^{-1} = \frac{1}{2}F_2 = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad \text{và} \quad F_4^{-1} = \frac{1}{4}F_4 = \frac{1}{4}\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix}$$

2.3 Biến đổi Fourier rời rạc (DFT)

2.3.1 Cơ sở của biến đổi Fourier rời rạc*

Biến đổi Fourier rời rạc (DFT) có thể được xem là ánh xạ tuyến tính từ không gian vector tín hiệu rời rạc vào không gian tần số. Về mặt ý tưởng, DFT là sự rời rạc hóa của biến đổi Fourier liên tục, vốn có công thức:

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}dt$$

Khi chỉ có N mẫu tín hiệu rời rạc được lấy cách đều nhau tại các thời điểm $t_k = kT$, ta thay tích phân bằng tổng rời rạc, và thu được phép biến đổi gần đúng:

$$F(\omega) \approx \sum_{k=0}^{N-1} f[k]\, e^{-j\omega kT}.$$

Nếu chỉ xét các giá trị rời rạc của tần số $\omega_n = \frac{2\pi n}{NT}$ với $n = 0, 1, ..., N-1$, thì ta thu được dạng rời rạc chuẩn hóa của biến đổi Fourier, tức chính là DFT:

$$F[n] = \sum_{k=0}^{N-1} f[k]\, e^{-j\frac{2\pi}{N}nk}, \quad n = 0, 1, ..., N-1.$$

2.3.2 DFT dưới góc nhìn đại số tuyến tính

Xét một tín hiệu rời rạc hữu hạn gồm n phần tử, được biểu diễn dưới dạng vector cột $\mathbf{x} \in \mathbb{C}^{n\times 1}$. Trong khuôn khổ đại số tuyến tính, Biến đổi Fourier Rời Rạc (DFT)

*Các công thức ở mục **2.3.1** tham khảo từ Isaac Amidror, *Mastering the Discrete Fourier Transform in One, Two or Several Dimensions: Pitfalls and Artifacts*, Springer, 2013, Chương 2, Mục 3, trang 15, 19.

được hiểu là một ánh xạ tuyến tính từ không gian tín hiệu $\mathbb{C}^n$ sang miền tần số, thông qua phép nhân với ma trận **ma trận Fourier**:

$$\mathbf{y} = F_n \mathbf{x}$$

Trong đó, F_n là ma trận Fourier chuẩn cấp n, có kích thước $n \times n$, được xác định bởi:

$$[F_n]_{k,j} = \xi^{jk}, \quad \text{với } \xi = e^{-2\pi i/n}, \quad 0 \leq j, k < n.$$

Mỗi phần tử hàng k, cột j của ma trận F_n là một lũy thừa của căn bậc n của đơn vị ξ. Khi nhân F_n với vector tín hiệu $\mathbf{x}$, ta thu được vector kết quả $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})^T$, biểu diễn tín hiệu trong miền tần số.

Phần tử thứ k trong kết quả biến đổi $\mathbf{y}$, tức là $y_k = [F_n \mathbf{x}]_k$, chính là giá trị của DFT tại chỉ số k, và được tính theo công thức:

$$\boxed{[F_n \mathbf{x}]_k = \sum_{j=0}^{n-1} x_j \, \xi^{jk}} \tag{2.3.2}$$

Đây chính là **định nghĩa đại số tuyến tính của DFT**, cho thấy mỗi hệ số DFT là một tổ hợp tuyến tính của các giá trị đầu vào, với trọng số là các lũy thừa của căn bậc n của đơn vị.

Ví dụ 2: Tính DFT bằng định nghĩa

Xét tín hiệu rời rạc độ dài $n = 4$ được biểu diễn bởi vector:

$$\mathbf{x} = \begin{pmatrix} 1 & 2 & 1 & 0 \end{pmatrix}^T$$

Ta cần tính DFT của $\mathbf{x}$ bằng công thức:

$$[F_4 \mathbf{x}]_k = \sum_{j=0}^{3} x_j \cdot \xi^{jk}, \quad \text{với } \xi = e^{-2\pi i/4} = -i$$

Tính lần lượt các hệ số DFT:

$$\begin{aligned} y_0 &= x_0 \cdot \xi^{0 \cdot 0} + x_1 \cdot \xi^{1 \cdot 0} + x_2 \cdot \xi^{2 \cdot 0} + x_3 \cdot \xi^{3 \cdot 0} \\ &= 1 + 2 + 1 + 0 = 4 \end{aligned}$$

$$\begin{aligned} y_1 &= x_0 \cdot \xi^0 + x_1 \cdot \xi^1 + x_2 \cdot \xi^2 + x_3 \cdot \xi^3 \\ &= 1 + 2(-i) + 1(-1) + 0(i) = 1 - 2i - 1 = -2i \end{aligned}$$

(2.3.2)Trích từ Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000, Chương 5, Mục 8, trang 358.

$$
\begin{aligned}
y_2 &= x_0 \cdot \xi^0 + x_1 \cdot \xi^2 + x_2 \cdot \xi^4 + x_3 \cdot \xi^6 \\
&= 1 + 2(-1) + 1(1) + 0(-1) = 1 - 2 + 1 = 0 \\
y_3 &= x_0 \cdot \xi^0 + x_1 \cdot \xi^3 + x_2 \cdot \xi^6 + x_3 \cdot \xi^9 \\
&= 1 + 2(i) + 1(-1) + 0(-i) = 1 + 2i - 1 = 2i
\end{aligned}
$$

Vậy: Phổ DFT thu được là:

$$\mathbf{y} = F_4\mathbf{x} = \begin{pmatrix} 4 & -2i & 0 & 2i \end{pmatrix}^T$$

Vector biên độ phổ:

$$|\mathbf{y}| = (|y_0|, |y_1|, |y_2|, |y_3|)^T = \begin{pmatrix} 4 & 2 & 0 & 2 \end{pmatrix}^T$$

2.4 Biến đổi Fourier rời rạc ngược (IDFT)

2.4.1 Định nghĩa

Sau khi chuyển tín hiệu từ miền thời gian sang miền tần số bằng DFT, ta hoàn toàn có thể khôi phục lại tín hiệu gốc thông qua phép biến đổi Fourier rời rạc ngược (Inverse Discrete Fourier Transform – IDFT). Về mặt đại số tuyến tính, IDFT được định nghĩa bởi:

$$\mathbf{x} = F_n^{-1}\mathbf{y} = \frac{1}{n}\overline{F_n}\mathbf{y} \tag{2.4.1a}$$

Trong đó, $\overline{F_n}$ là liên hợp phức của ma trận Fourier F_n và $\mathbf{y}$ là đầu ra của phép DFT ở trên. Từ đó, phần tử thứ k của tín hiệu gốc được tính theo công thức:

$$\boxed{x_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j \omega^{jk}, \quad \text{với } \omega = e^{2\pi i/n}} \tag{2.4.1b}$$

Ví dụ 3: Tính IDFT bằng định nghĩa

Ta đã có vector DFT (phổ tần số) thu được từ ví dụ trước là:

$$\mathbf{y} = \begin{pmatrix} 4 & -2i & 0 & 2i \end{pmatrix}^T$$

Ta sẽ khôi phục lại tín hiệu ban đầu $\mathbf{x}$ bằng định nghĩa IDFT:

$$x_k = \frac{1}{n}\sum_{j=0}^{n-1} y_j \cdot \omega^{jk}, \quad \text{với } \omega = e^{2\pi i/n}, \quad n = 4$$

Trong trường hợp này, $\omega = e^{2\pi i/4} = i$.

(2.4.1a), (2.4.1b)Tham khảo từ Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000, Chương 5, Mục 8, trang 358.

Tính từng phần tử x_k:

$$\begin{aligned}
x_0 &= \frac{1}{4}(y_0 \cdot \omega^{0\cdot 0} + y_1 \cdot \omega^{1\cdot 0} + y_2 \cdot \omega^{2\cdot 0} + y_3 \cdot \omega^{3\cdot 0}) \\
&= \frac{1}{4}(4 + (-2i) + 0 + 2i) = \frac{1}{4}(4) = 1 \\
x_1 &= \frac{1}{4}(y_0 \cdot \omega^{0\cdot 1} + y_1 \cdot \omega^{1\cdot 1} + y_2 \cdot \omega^{2\cdot 1} + y_3 \cdot \omega^{3\cdot 1}) \\
&= \frac{1}{4}(4 + (-2i)(i) + 0(-1) + 2i(-i)) = \frac{1}{4}(4 + 2 + 0 + 2) = \frac{8}{4} = 2 \\
x_2 &= \frac{1}{4}(y_0 + y_1 \cdot \omega^2 + y_2 \cdot \omega^4 + y_3 \cdot \omega^6) \\
&= \frac{1}{4}(4 + (-2i)(-1) + 0(1) + 2i(-1)) = \frac{1}{4}(4 + 2i + 0 - 2i) = \frac{4}{4} = 1 \\
x_3 &= \frac{1}{4}(4 + (-2i)(-i) + 0(-1) + 2i(i)) \\
&= \frac{1}{4}(4 - 2 - 0 - 2) = \frac{0}{4} = 0
\end{aligned}$$

Kết luận: Ta đã khôi phục lại tín hiệu ban đầu:

$$\mathbf{x} = \begin{pmatrix} 1 & 2 & 1 & 0 \end{pmatrix}^T$$

Kết quả này trùng khớp với tín hiệu đầu vào ban đầu, xác nhận rằng phép biến đổi IDFT đã hoạt động chính xác.

2.4.2 Tính IDFT bằng liên hợp phức

Bất kỳ thuật toán hoặc chương trình nào tính DFT cũng có thể được sử dụng để tính IDFT. Điều này xuất phát từ mối liên hệ đại số:

$$F_n^{-1}\mathbf{y} = \frac{1}{n} \cdot \overline{F_n \cdot \overline{\mathbf{y}}}$$

Nhờ đó, ta tính IDFT qua ba bước sau:

1. $\mathbf{y} \leftarrow \overline{\mathbf{y}}$ (liên hợp phức đầu vào)
2. $\mathbf{y} \leftarrow F_n\overline{\mathbf{y}}$ (thực hiện DFT)
3. $\mathbf{y} \leftarrow \frac{1}{n} \cdot \overline{\mathbf{y}}$ (liên hợp kết quả và chuẩn hóa)

Ví dụ 4: Tính biến đổi ngược của $\mathbf{x} = \begin{pmatrix} i & 0 & -i & 0 \end{pmatrix}^T$ được thực hiện như sau với F_4 đã được cho trong Ví dụ 1:

$$\overline{\mathbf{x}} = \begin{pmatrix} -i & 0 & i & 0 \end{pmatrix}^T$$

$$F_4\overline{\mathbf{x}} = \begin{pmatrix} 0 & -2i & 0 & -2i \end{pmatrix}^T$$

$$\frac{1}{4}\overline{F_4\overline{\mathbf{x}}} = \frac{1}{4}\begin{pmatrix} 0 & 2i & 0 & 2i \end{pmatrix}^T = F_4^{-1}\mathbf{x}$$

2.5 Định lý tích chập (Convolution theorem)

2.5.1 Nhân đa thức

Cho hai đa thức:

$$p(x) = \sum_{k=0}^{n-1} \alpha_k x^k, \quad q(x) = \sum_{k=0}^{n-1} \beta_k x^k$$

Khi đó, tích của chúng là một đa thức có bậc tối đa $2n-2$:

$$p(x)q(x) = \sum_{k=0}^{2n-2} \gamma_k x^k, \quad \text{với } \gamma_k = \sum_{j=0}^{k} \alpha_j \beta_{k-j}$$

Tức là, mỗi hệ số γ_k chính là kết quả của **phép chập rời rạc** (discrete convolution) giữa hai dãy hệ số:

$$\mathbf{a} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{pmatrix}, \quad \mathbf{b} = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix}, \quad \gamma_k = [\mathbf{a} \odot \mathbf{b}]_k$$

Như đã thấy, nhân hai đa thức tương đương với thực hiện phép chập rời rạc giữa hai vector hệ số. Do đó, nếu ta có cách thực hiện phép chập hiệu quả, thì cũng có thể nhân đa thức một cách nhanh chóng.

Có hai ý tưởng chính làm nền tảng cho phương pháp này:

- **(1)** Biến đổi Fourier rời rạc (DFT) có thể biến phép chập trong miền thời gian thành phép nhân đơn giản trong miền tần số. Đây chính là nội dung của **định lý chập**.
- **(2)** Biến đổi Fourier có thể được tính nhanh bằng thuật toán FFT, giúp giảm đáng kể chi phí tính toán.

Từ đó, Nhân đa thức có thể thực hiện hiệu quả bằng DFT/IDFT, ta chỉ cần biến đổi Fourier hai vector hệ số, nhân từng phần tử, rồi biến đổi ngược.

2.5.2 Nội dung định lý tích chập

Cho $\mathbf{a} \times \mathbf{b}$ biểu thị tích từng phần tử:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_{n-1} \end{pmatrix} \times \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix} = \begin{pmatrix} \alpha_0\beta_0 \\ \alpha_1\beta_1 \\ \vdots \\ \alpha_{n-1}\beta_{n-1} \end{pmatrix}_{n\times 1}$$

và cho $\hat{\mathbf{a}}$ và $\hat{\mathbf{b}}$ là các vector được mở rộng:

$$\hat{\mathbf{a}} = \begin{pmatrix} \alpha_0 \\ \vdots \\ \alpha_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}_{2n\times 1} \quad \text{và} \quad \hat{\mathbf{b}} = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_{n-1} \\ 0 \\ \vdots \\ 0 \end{pmatrix}_{2n\times 1}$$

Nếu $F = F_{2n}$ là ma trận Fourier cấp $2n$, thì:

$$\boxed{F(\mathbf{a} \odot \mathbf{b}) = (F\hat{\mathbf{a}}) \times (F\hat{\mathbf{b}}) \quad \text{và} \quad \mathbf{a} \odot \mathbf{b} = F^{-1}[(F\hat{\mathbf{a}}) \times (F\hat{\mathbf{b}})]} \tag{2.5.2}$$

Chứng minh:

Quan sát rằng thành phần thứ t trong $F_*^j \times F_*^k$ là:

$$[F_*^j \times F_*^k]_t = \xi^{tj}\xi^{tk} = \xi^{t(j+k)} = [F_*^{j+k}]_t$$

do đó các cột của F có tính chất:

$$F_*^j \times F_*^k = F_*^{j+k} \quad \text{với mỗi } j, k = 0, 1, \ldots, (n-1)$$

Điều này có nghĩa là nếu $F\hat{\mathbf{a}}$, $F\hat{\mathbf{b}}$, và $F(\mathbf{a} \odot \mathbf{b})$ được biểu diễn như các tổ hợp của các cột của F như sau:

$$F\hat{\mathbf{a}} = \sum_{k=0}^{n-1} \alpha_k F_*^k$$
$$F\hat{\mathbf{b}} = \sum_{k=0}^{n-1} \beta_k F_*^k$$

(2.5.2)Trích từ Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000, Chương 5, Mục 8, trang 367.

$$F(\mathbf{a} \odot \mathbf{b}) = \sum_{k=0}^{2n-2} [\mathbf{a} \odot \mathbf{b}]_k F_*^k$$

thì việc tính toán $(F\hat{\mathbf{a}}) \times (F\hat{\mathbf{b}})$ hoàn toàn giống như việc tạo tích của hai đa thức theo nghĩa:

$$\begin{aligned}(F\hat{\mathbf{a}}) \times (F\hat{\mathbf{b}}) &= \left(\sum_{k=0}^{n-1} \alpha_k F_*^k\right) \times \left(\sum_{k=0}^{n-1} \beta_k F_*^k\right) \\ &= \sum_{k=0}^{2n-2} \left(\sum_{j=0}^{k} \alpha_j \beta_{k-j}\right) F_*^k \\ &= \sum_{k=0}^{2n-2} [\mathbf{a} \odot \mathbf{b}]_k F_*^k = F(\mathbf{a} \odot \mathbf{b})\end{aligned}$$

Theo định lý tích chập, tích chập của hai vector $n \times 1$ có thể được tính bằng cách thực hiện ba phép biến đổi Fourier rời rạc cấp $2n$:

$$\mathbf{a}_{n\times 1} \odot \mathbf{b}_{n\times 1} = F_{2n}^{-1}[(F_{2n}\hat{\mathbf{a}}) \times (F_{2n}\hat{\mathbf{b}})]$$

Việc thực hiện phép tích chập theo định nghĩa cần khoảng n^2 phép nhân vô hướng. Nếu sử dụng DFT cấp $2n$ và thực hiện trực tiếp bằng phép nhân ma trận–vector, số phép nhân tăng lên đến $4n^2$, tức gấp ít nhất 12 lần. Do đó, để khai thác định lý chập một cách hiệu quả, cần một phương pháp tính DFT hiệu quả hơn.

Thuật toán Fourier nhanh (Fast Fourier Transform - FFT) cho phép tính $F_n\mathbf{x}$ với chỉ khoảng $(n/2)\log_2 n$ phép nhân vô hướng. Khi dùng FFT để thực hiện tích chập, tổng số phép nhân chỉ còn khoảng khoảng $3n\log_2 n$, nhỏ hơn rất nhiều so với n^2 khi n lớn.

2.6 Hạn chế của biến đổi Fourier rời rạc

Aliasing (chồng phổ)

Aliasing (hay chồng phổ) là một hiện tượng xảy ra trong quá trình lấy mẫu tín hiệu tương tự để chuyển đổi thành tín hiệu số, tức là bước tiền xử lý trước khi có được vector $\mathbf{x}$.

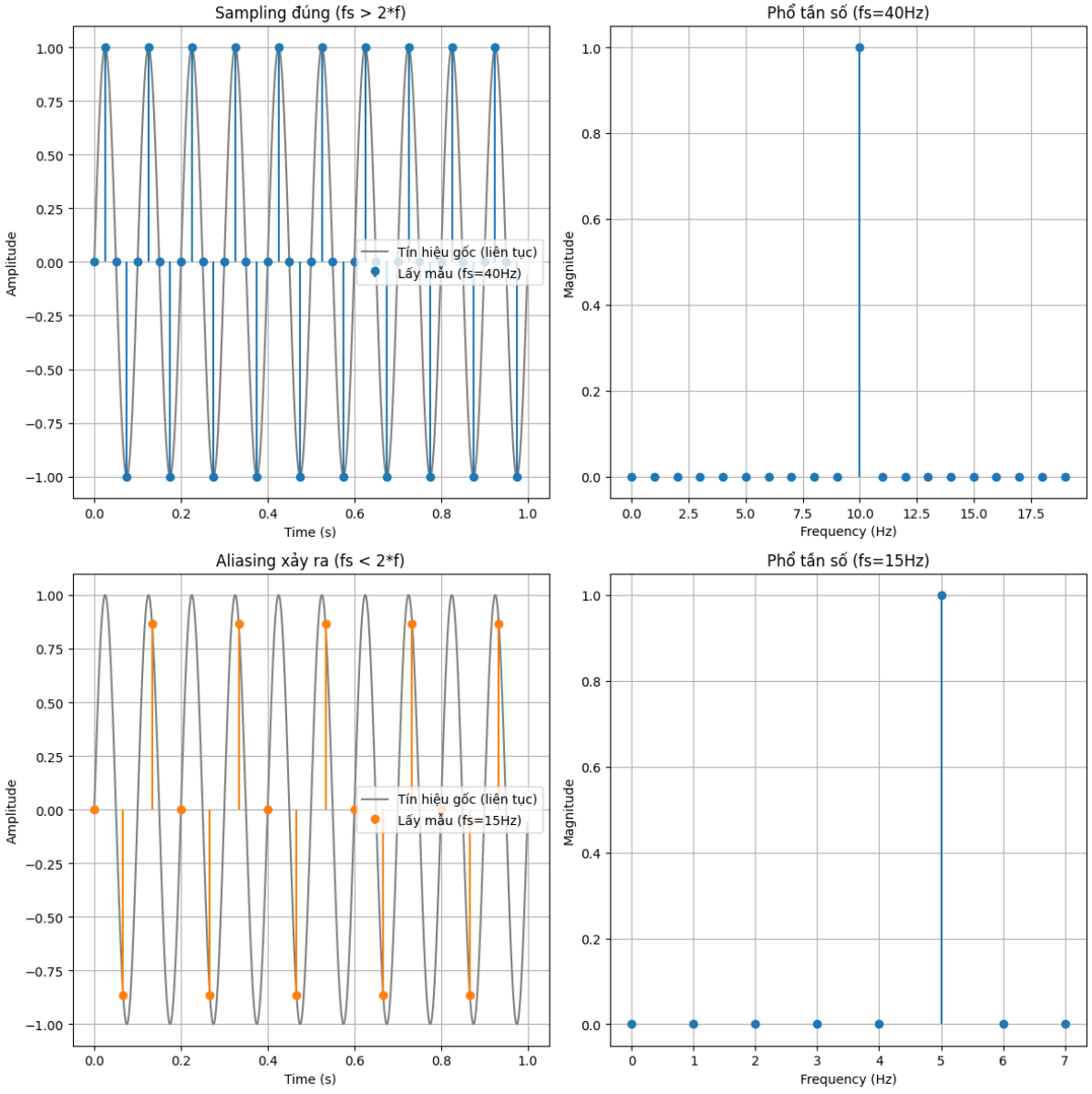
Giải thích

Theo định lý lấy mẫu Nyquist-Shannon, để tái tạo chính xác một tín hiệu tương tự, tần số lấy mẫu f_s phải lớn hơn ít nhất hai lần tần số cao nhất f_{max} có trong tín hiệu đó:

$$f_s > 2f_{max}$$

Nếu điều kiện này không được thỏa mãn, các thành phần tần số cao hơn $f_s/2$ (tần số Nyquist) sẽ bị (folded back) và xuất hiện dưới dạng các thành phần tần số thấp hơn

trong phổ của tín hiệu đã lấy mẫu. Điều này tạo ra các thành phần tần số , làm sai lệch hoàn toàn kết quả DFT. Trong tài liệu, ta bắt đầu với một vector dữ liệu rời rạc $\mathbf{x}_{n\times 1}$ và giả định rằng tín hiệu đã được lấy mẫu đúng cách để tránh aliasing.



Hình 1: Hiện tượng Aliasing.

Nhận xét: Từ Hình 1 ta nhận thấy rằng:

- Tín hiệu gốc có tần số 10 Hz.
- Trường hợp 1: Chúng ta lấy mẫu với tần số $fs1 = 40$ Hz, nghĩa là tần số Nyquist là $40/2 = 20$ Hz.
  - Vì 10 Hz $<$ 20 Hz, hiện tượng aliasing KHÔNG xảy ra.
  - Trong miền thời gian (biểu đồ trên cùng bên trái), các điểm lấy mẫu biểu

diễn chính xác tín hiệu gốc 10 Hz.

- Trong miền tần số (biểu đồ trên cùng bên phải), đỉnh phổ xuất hiện tại 10 Hz, đúng với tần số tín hiệu gốc.

- Trường hợp 2: Chúng ta lấy mẫu với tần số $fs2 = 15$ Hz, nghĩa là tần số Nyquist là $15/2 = 7.5$ Hz.
  - Vì 10 Hz $>$ 7.5 Hz, hiện tượng aliasing xảy ra.
  - Trong miền thời gian (biểu đồ dưới cùng bên trái), tín hiệu lấy mẫu 10 Hz trông giống như một sóng sin có tần số thấp hơn ($15 - 10 = 5$ Hz).
  - Trong miền tần số (biểu đồ dưới cùng bên phải), đỉnh phổ xuất hiện tại 5 Hz, chứ không phải 10 Hz.
  - Để tránh aliasing, tần số lấy mẫu phải ít nhất là $2 \times 10 = 20$ Hz.

Kết luận:

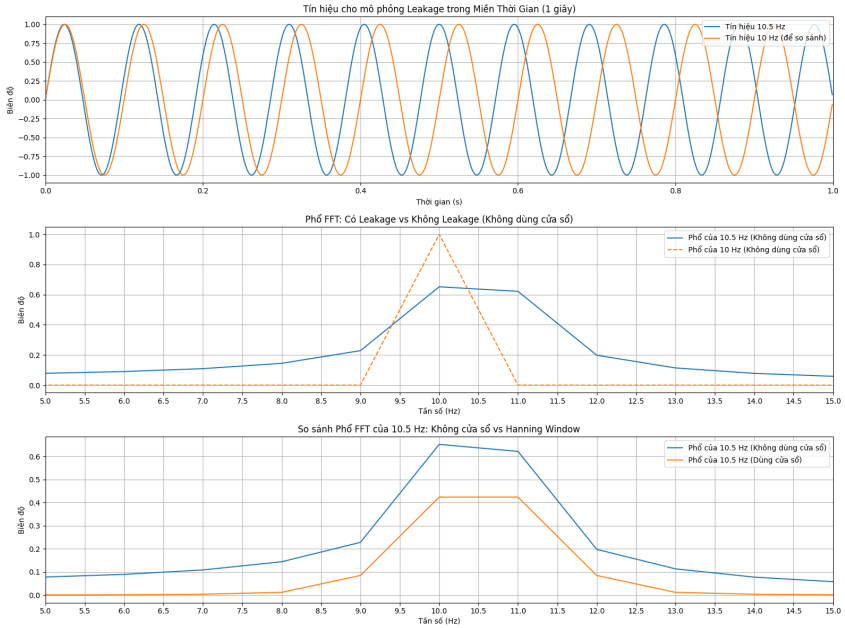
Aliasing làm mất thông tin tần số gốc và tạo ra các tần số giả trong phổ, không thể phục hồi. Để tránh aliasing, tần số lấy mẫu f_s phải lớn hơn gấp đôi tần số cao nhất f_{max} của tín hiệu, tức là $f_s > 2 \times f_{max}$ (định lý Nyquist-Shannon).

Leakage (rò rỉ phổ)

Leakage (hay rò rỉ phổ) là hiện tượng năng lượng của một thành phần tần số sin thuần túy bị lan tỏa ra nhiều ô tần số (frequency bins) lân cận trong kết quả DFT, thay vì tập trung tại một điểm duy nhất.

Giải thích

DFT xử lý một đoạn tín hiệu có độ dài hữu hạn n. Phương pháp này ngầm định rằng đoạn tín hiệu này chứa một chu kỳ của một tín hiệu tuần hoàn vô hạn. Nếu số chu kỳ của của các thành phần trong đoạn tín hiệu đó không phải là số nguyên, sự liên tục tại điểm nối giữa các chu kỳ tuần hoàn sẽ bị mất đi. Sự mất liên tục này chính là nguyên nhân gây ra leakage. Về mặt toán học, việc cắt một đoạn tín hiệu hữu hạn tương đương với việc nhân tín hiệu vô hạn với một hàm cửa sổ hình chữ nhật. Phép biến đổi Fourier của hàm cửa sổ này có các thùy phụ (sidelobes) khác 0, gây ra sự rò rỉ năng lượng. Hậu quả là biên độ của các thành phần tần số bị ước tính thấp hơn thực tế và có thể che khuất các thành phần tần số yếu hơn ở gần đó.



Hình 2: Hiện tượng Leakage.

Nhận xét: Từ Hình 2

- Tín hiệu 10 Hz tạo ra đỉnh phổ sắc nét tại 10 Hz vì có số nguyên chu kỳ trong 1 giây (10 chu kỳ).
- Tín hiệu 10.5 Hz tạo ra đỉnh phổ 'rộng' hơn với các thùy bên rõ rệt khi không dùng cửa sổ (hiện tượng leakage).
- Khi áp dụng cửa sổ Hanning làm giảm biên độ ở hai đầu khung tín hiệu về gần 0, các thùy bên của tín hiệu 10.5 Hz được giảm đáng kể.

Kết luận:

Rò rỉ phổ làm mờ các đỉnh tần số và có thể che khuất các thành phần tín hiệu yếu. Cửa sổ hóa là giải pháp hiệu quả để giảm thiểu nó. Cửa sổ hóa giúp giảm ảnh hưởng của các discontinuities (điểm gián đoạn) ở biên của khung phân tích.

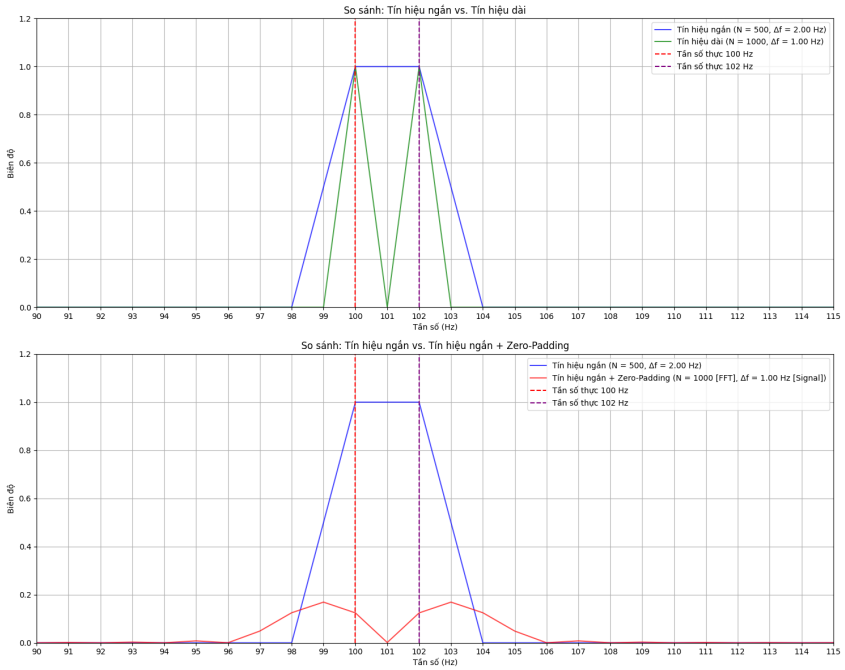
Thông qua áp dụng cửa sổ Hann để làm giảm biên độ ở hai đầu khung tín hiệu về gần 0, giúp loại bỏ các bước nhảy đột ngột mà FFT giả định là tuần hoàn, từ đó ngăn chặn sự rò rỉ năng lượng tần số sang các tần số lân cận và làm cho biểu đồ phổ trở nên sắc nét và đáng tin cậy hơn.

Resolution Limit (Giới hạn Độ phân giải)

Độ phân giải tần số trong DFT phụ thuộc vào độ dài cửa sổ quan sát, tức là số lượng mẫu ban đầu n, chứ không phải số điểm trong biến đổi DFT. Việc tăng số điểm DFT,

chẳng hạn thông qua kỹ thuật đệm thêm số 0 (*zero-padding*) vào vector tín hiệu $\mathbf{x}$, không làm tăng độ phân giải tần số thực sự mà chỉ làm mịn biểu đồ phổ.

Như đã đề cập trong phần **Định lý tích chập**, việc đệm 0 có tác dụng làm tăng kích thước ma trận Fourier (ví dụ từ F_n thành F_{2n}), từ đó cho phép ta thu được phổ với nhiều điểm hơn. Tuy nhiên, điều này chỉ đơn thuần giúp làm mịn (interpolate) phổ tần số, tức là tăng độ phân giải biểu diễn, chứ **không làm tăng khả năng phân biệt** giữa các tần số gần nhau.



Hình 3: Hiện tượng Resolution.

Nhận xét: Từ Hình 3

- Với thời gian quan sát ngắn 0.5s, độ phân giải lý thuyết là 2.00 Hz. Hai đỉnh 100.0 Hz và 102.0 Hz bị trộn lẫn, xuất hiện như một đỉnh duy nhất.
- Với thời gian quan sát dài 1s, độ phân giải lý thuyết là 1.00 Hz. Hai đỉnh 100.0 Hz và 102.0 Hz được phân biệt rõ ràng.
- Dùng Zero-padding thêm M số 0 vào cuối một chuỗi dữ liệu (khung tín hiệu) trong miền thời gian, độ dài của chuỗi tín hiệu đầu vào cho FFT tăng lên (từ N lên $N + M$) làm cho số lượng điểm tần số đầu ra của FFT cũng tăng lên tương ứng, các điểm tần số này sẽ được đặt gần nhau hơn trên trục tần số, làm cho phổ mượt mà hơn và có nhiều bin tần số hơn.

Kết luận:

Độ phân giải tần số phụ thuộc trực tiếp vào thời gian quan sát tín hiệu. Để phân biệt các tần số gần nhau, cần thu thập dữ liệu trong một khoảng thời gian đủ dài. Zero-padding có thể giúp việc biểu diễn phổ mượt mà hơn và tạo ra nhiều bin tần số hơn nhưng không cải thiện độ phân giải thực tế hay khả năng phân biệt các tần số nếu dữ liệu gốc không đủ dài. Nó chỉ nội suy giữa các điểm phổ đã có, không cung cấp thông tin mới về tần số.

Nếu hai thành phần tần số đã bị "hoà lẫn"do hiện tượng rò rỉ phổ (leakage) gây ra bởi cửa sổ ngắn, thì việc đệm 0 không thể khắc phục điều đó. Muốn cải thiện độ phân giải thực sự, cần tăng thời gian quan sát tín hiệu, tức là phải lấy nhiều mẫu hơn (tăng n), từ đó làm thu hẹp khoảng cách giữa các tần số rời rạc trong miền DFT.

2.7 Biến đổi Fourier nhanh (FFT)

2.7.1 Phân rã ma trận Fourier - Tiền đề của FFT

Thuật toán biến đổi Fourier nhanh xuất phát từ thực tế rằng nếu n là lũy thừa của 2, thì biến đổi Fourier rời rạc cấp n có thể được thực hiện bằng cách thực hiện hai phép biến đổi cấp $n/2$. Để hiểu chính xác điều này xảy ra như thế nào, quan sát rằng khi $n = 2^r$ ta có $(\xi^j)^n = (\xi^{2j})^{n/2}$, do đó:

$$\{1, \xi, \xi^2, \xi^3, \ldots, \xi^{n-1}\} = \text{căn bậc } n \text{ của đơn vị}$$

khi và chỉ khi:

$$\{1, \xi^2, \xi^4, \xi^6, \ldots, \xi^{n-2}\} = \text{căn bậc } (n/2) \text{ của đơn vị}$$

Điều này có nghĩa là các phần tử (j, k) trong ma trận Fourier F_n và $F_{n/2}$ là:

$$[F_n]_{jk} = \xi^{jk} \quad \text{và} \quad [F_{n/2}]_{jk} = (\xi^2)^{jk} = \xi^{2jk} \tag{*}$$

Nếu các cột của F_n được hoán vị sao cho các cột có chỉ số chẵn được liệt kê trước các cột có chỉ số lẻ, và nếu P_n^T là ma trận hoán vị tương ứng, thì ta có thể phân chia $F_n P_n^T$ như sau:

$$F_n P_n^T = [F_*^0 \; F_*^2 \; \cdots \; F_*^{n-2} \mid F_*^1 \; F_*^3 \; \cdots \; F_*^{n-1}] = \begin{pmatrix} A_{n/2\times n/2} & B_{n/2\times n/2} \\ C_{n/2\times n/2} & G_{n/2\times n/2} \end{pmatrix}$$

Bằng cách sử dụng (*) cùng với $\xi^{nk} = 1$ và:

$$\xi^{n/2} = \cos\frac{2\pi(n/2)}{n} - i\sin\frac{2\pi(n/2)}{n} = -1$$

ta thấy rằng các phần tử trong A, B, C, và G là:

$$\begin{aligned}
A_{jk} &= F_{j,2k} = \xi^{2jk} = [F_{n/2}]_{jk} \\
B_{jk} &= F_{j,2k+1} = \xi^{j(2k+1)} = \xi^j \xi^{2jk} = \xi^j [F_{n/2}]_{jk} \\
C_{jk} &= F_{n/2+j,2k} = \xi^{(n/2+j)2k} = \xi^{nk} \xi^{2jk} = \xi^{2jk} = [F_{n/2}]_{jk} \\
G_{jk} &= F_{n/2+j,2k+1} = \xi^{(n/2+j)(2k+1)} = \xi^{nk} \xi^{n/2} \xi^j \xi^{2jk} = -\xi^j \xi^{2jt} = -\xi^j [F_{n/2}]_{jk}
\end{aligned}$$

Nói cách khác, nếu $D_{n/2}$ là ma trận đường chéo:

$$D_{n/2} = \begin{pmatrix} 1 & & & & \\ & \xi & & & \\ & & \xi^2 & & \\ & & & \ddots & \\ & & & & \xi^{n/2-1} \end{pmatrix}$$

thì:

$$F_n P_n^T = \begin{pmatrix} A_{(n/2)\times(n/2)} & B_{(n/2)\times(n/2)} \\ C_{(n/2)\times(n/2)} & G_{(n/2)\times(n/2)} \end{pmatrix} = \begin{pmatrix} F_{n/2} & D_{n/2}F_{n/2} \\ F_{n/2} & -D_{n/2}F_{n/2} \end{pmatrix} \tag{2.7.1}$$

2.7.2 Triển khai thuật toán FFT

Đối với một vector đầu vào x chứa $n = 2^r$ thành phần, biến đổi Fourier rời rạc $F_n x$ là kết quả của việc tạo liên tiếp các mảng sau đây:

1. $X_{1\times n} \leftarrow \text{rev}(x)$ (đảo bit các chỉ số)
2. Đối với $j = 0, 1, 2, \ldots, r-1$
   - $D \leftarrow \begin{pmatrix} 1 \\ e^{-\pi i/2^j} \\ e^{-2\pi i/2^j} \\ \vdots \\ e^{-(2^j-1)\pi i/2^j} \end{pmatrix}_{2^j\times 1}$ (Một nửa số căn bậc 2^{j+1} của 1)
   - $X^{(0)} \leftarrow \begin{bmatrix} X_{*0} & X_{*2} & \cdots & X_{*2^{r-j}-2} \end{bmatrix}_{2^j\times 2^{r-j-1}}$
   - $X^{(1)} \leftarrow \begin{bmatrix} X_{*1} & X_{*3} & \cdots & X_{*2^{r-j}-1} \end{bmatrix}_{2^j\times 2^{r-j-1}}$
   - $X \leftarrow \begin{pmatrix} X^{(0)} + D \times X^{(1)} \\ X^{(0)} - D \times X^{(1)} \end{pmatrix}_{2^{j+1}\times 2^{r-j-1}}$
     (Phép nhân $\times$ được định nghĩa là $[D \times M]_{ik} = d_i m_{ik}$)

(2.7.1)Trích từ Carl D. Meyer, *Matrix Analysis and Applied Linear Algebra*, SIAM, 2000, Chương 5, Mục 8, trang 369.

Ví dụ 5: Thực hiện FFT trên $x = (x_0, x_1, x_2, x_3)^T$.
Lời giải:
Bắt đầu với $X \leftarrow \text{rev}(x) = (x_0, x_2, x_1, x_3)^T$.

- **Với j = 0:**
  - $D \leftarrow (1)$
  - $X^{(0)} \leftarrow \begin{pmatrix} x_0 & x_1 \end{pmatrix}$ và $X^{(1)} \leftarrow \begin{pmatrix} x_2 & x_3 \end{pmatrix}$
  - $D \times X^{(1)} \leftarrow \begin{pmatrix} x_2 & x_3 \end{pmatrix}$
  - $X \leftarrow \begin{pmatrix} X^{(0)} + D \times X^{(1)} \\ X^{(0)} - D \times X^{(1)} \end{pmatrix} = \begin{pmatrix} x_0 + x_2 & x_1 + x_3 \\ x_0 - x_2 & x_1 - x_3 \end{pmatrix}$
- **Với j = 1:**
  - $D \leftarrow \begin{pmatrix} 1 \\ -i \end{pmatrix}$
  - $X^{(0)} \leftarrow \begin{pmatrix} x_0 + x_2 \\ x_0 - x_2 \end{pmatrix}$ và $X^{(1)} \leftarrow \begin{pmatrix} x_1 + x_3 \\ x_1 - x_3 \end{pmatrix}$
  - $D \times X^{(1)} \leftarrow \begin{pmatrix} x_1 + x_3 \\ -i(x_1 - x_3) \end{pmatrix} = \begin{pmatrix} x_1 + x_3 \\ -ix_1 + ix_3 \end{pmatrix}$
  - $X \leftarrow \begin{pmatrix} X^{(0)} + D \times X^{(1)} \\ X^{(0)} - D \times X^{(1)} \end{pmatrix} = \begin{pmatrix} x_0 + x_2 + x_1 + x_3 \\ x_0 - x_2 - ix_1 + ix_3 \\ x_0 + x_2 - x_1 - x_3 \\ x_0 - x_2 + ix_1 - ix_3 \end{pmatrix} = F_4 x$

Kết quả này khớp với kết quả thu được bằng cách nhân ma trận-vector trực tiếp với F_4.

Giảm độ phức tạp tính toán bằng FFT:

Thuật toán được gọi là "nhanh"vì số lượng phép nhân nó sử dụng ít hơn rất nhiều so với phương pháp tính DFT trực tiếp. Ở mỗi vòng lặp thứ j, thuật toán thực hiện 2^j phép nhân cho mỗi trong số 2^{r-j-1} nhóm, dẫn đến tổng cộng $2^j \cdot 2^{r-j-1} = 2^{r-1}$ phép nhân ở mỗi vòng.

Vì toàn bộ thuật toán gồm $r = \log_2 n$ vòng lặp, nên tổng số phép nhân cần thiết là:

$$r \cdot 2^{r-1} = \frac{n}{2} \log_2 n$$

Tổng quát: Nếu n là lũy thừa của 2, thuật toán FFT yêu cầu tối đa $(n/2) \log_2 n$ phép nhân để tính DFT cho một véc-tơ $n \times 1$.

Lợi ích này trở nên rõ rệt khi n lớn. Ví dụ, với $n = 512$, số phép nhân trong phép nhân ma trận–vector trực tiếp là $n^2 = 262{,}144$, trong khi FFT chỉ cần:

$$\frac{512}{2} \cdot \log_2 512 = 256 \cdot 9 = 2{,}304$$

Tức là FFT nhanh hơn khoảng 100 lần so với phương pháp thông thường.

2.8 Biến đổi Fourier Nhanh (FFT) - Thuật toán Cooley-Tukey Radix-2 Decimation-In-Time (DIT)

2.8.1 Cơ sở lý thuyết

Thuật toán Cooley-Tukey, đặc biệt là phiên bản Radix-2 Decimation-In-Time (DIT), là một trong những thuật toán FFT phổ biến nhất. "Decimation-In-Time"có nghĩa là tín hiệu đầu vào được chia thành các phần nhỏ hơn theo thời gian (chẵn/lẻ), và "Radix-2"có nghĩa là kích thước của DFT được chia đôi ở mỗi bước. Thuật toán này yêu cầu kích thước tín hiệu N phải là lũy thừa của 2.

Thuật toán FFT DIT hoạt động bằng cách chia một DFT N điểm thành hai DFT $N/2$ điểm: một cho các mẫu đầu vào có chỉ số chẵn và một cho các mẫu đầu vào có chỉ số lẻ. Kết quả của hai DFT con này sau đó được kết hợp bằng cách sử dụng các "hệ số xoắn"(twiddle factor) $W_N^k = e^{-j\frac{2\pi}{N}k}$ để tạo ra kết quả DFT N điểm cuối cùng.

Cụ thể, nếu $X[k]$ là DFT của $x[n]$, ta có thể chia $x[n]$ thành phần chẵn $x_{\text{even}}[n] = x[2n]$ và phần lẻ $x_{\text{odd}}[n] = x[2n+1]$. Khi đó, $X[k]$ có thể được viết lại thành:

$$\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} \\
&= \sum_{n=0}^{N/2-1} x[2n] e^{-j\frac{2\pi}{N}k(2n)} + \sum_{n=0}^{N/2-1} x[2n+1] e^{-j\frac{2\pi}{N}k(2n+1)} \\
&= \sum_{n=0}^{N/2-1} x_{\text{even}}[n] e^{-j\frac{2\pi}{N/2}kn} + \sum_{n=0}^{N/2-1} x_{\text{odd}}[n] e^{-j\frac{2\pi}{N}k(2n+1)} \\
&= \sum_{n=0}^{N/2-1} x_{\text{even}}[n] e^{-j\frac{2\pi}{N/2}kn} + e^{-j\frac{2\pi}{N}k} \sum_{n=0}^{N/2-1} x_{\text{odd}}[n] e^{-j\frac{2\pi}{N/2}kn}
\end{aligned}$$

Đặt $X_{\text{even}}[k]$ là DFT của $x_{\text{even}}[n]$ và $X_{\text{odd}}[k]$ là DFT của $x_{\text{odd}}[n]$, ta có:

$$X[k] = X_{\text{even}}[k] + W_N^k X_{\text{odd}}[k]$$

Và do tính tuần hoàn, ta có thêm công thức cho $k + N/2$:

$$X[k + N/2] = X_{\text{even}}[k] - W_N^k X_{\text{odd}}[k]$$

Đây là cấu trúc "butterfly"cơ bản của FFT DIT.

2.8.2 Ví dụ

Tín hiệu đầu vào: $x = [1, 2, 3, 4]$, với $N = 4$.

Bước 1: Chia nhỏ tín hiệu (Decimation-In-Time)

- **Lớp 1:** $N = 4$
  - Tín hiệu chẵn: $x_{\text{even}} = [x_0, x_2] = [1, 3]$
  - Tín hiệu lẻ: $x_{\text{odd}} = [x_1, x_3] = [2, 4]$
- **Lớp 2:** $N = 2$
  - Đối với $x_{\text{even}} = [1, 3] \rightarrow x_{\text{even,even}} = [1]$, $x_{\text{even,odd}} = [3]$
  - Đối với $x_{\text{odd}} = [2, 4] \rightarrow x_{\text{odd,even}} = [2]$, $x_{\text{odd,odd}} = [4]$

Khi tín hiệu chỉ có 1 phần tử, DFT của nó chính là giá trị đó. **Bước 2: Kết hợp các kết quả DFT con (Butterfly Operations)**

- **Kết hợp tạo DFT 2 điểm:**
  - *Đối với* $[1, 3]$:
    $X_{\text{even}}[k] = \text{DFT}\{[1]\}$, $X_{\text{odd}}[k] = \text{DFT}\{[3]\}$.
    Với $N = 2, k = 0$, $W_2^0 = 1$.

    $$X_0 = X_{\text{even}}[0] + W_2^0 X_{\text{odd}}[0] = 1 + 1 \times 3 = 4$$
    $$X_1 = X_{\text{even}}[0] - W_2^0 X_{\text{odd}}[0] = 1 - 1 \times 3 = -2$$

    Vậy, $\text{DFT}\{[1, 3]\} = [4, -2]$.
  - *Đối với* $[2, 4]$:
    $X_{\text{even}}[k] = \text{DFT}\{[2]\} = [2]$, $X_{\text{odd}}[k] = \text{DFT}\{[4]\} = [4]$.
    Với $N = 2, k = 0$, $W_2^0 = 1$.

    $$X_0 = X_{\text{even}}[0] + W_2^0 X_{\text{odd}}[0] = 2 + 1 \times 4 = 6$$
    $$X_1 = X_{\text{even}}[0] - W_2^0 X_{\text{odd}}[0] = 2 - 1 \times 4 = -2$$

    Vậy, $\text{DFT}\{[2, 4]\} = [6, -2]$.
- **Kết hợp tạo DFT 4 điểm:**
  $X_{\text{even}}[k] = \text{DFT}\{[1, 3]\} = [4, -2]$ và $X_{\text{odd}}[k] = \text{DFT}\{[2, 4]\} = [6, -2]$.
  Với $N = 4$.
  - $k = 0 : W_4^0 = 1$

    $$X_0 = X_{\text{even}}[0] + W_4^0 X_{\text{odd}}[0] = 4 + 1 \times 6 = 10$$
    $$X_2 = X_{\text{even}}[0] - W_4^0 X_{\text{odd}}[0] = 4 - 1 \times 6 = -2$$

– $k = 1 : W_4^1 = e^{-j\pi/2} = \cos(-\frac{\pi}{2}) + j\sin(-\frac{\pi}{2}) = -j$ (Theo công thức Euler)

$$X_1 = X_{\text{even}}[1] + W_4^1 X_{\text{odd}}[1] = -2 + (-j) \times (-2) = -2 + 2j$$
$$X_3 = X_{\text{even}}[1] - W_4^1 X_{\text{odd}}[1] = -2 - (-j) \times (-2) = -2 - 2j$$

Kết quả cuối cùng: DFT của tín hiệu $x = [1, 2, 3, 4]$ là:

$$X = [10, -2 + 2j, -2, -2 - 2j]$$

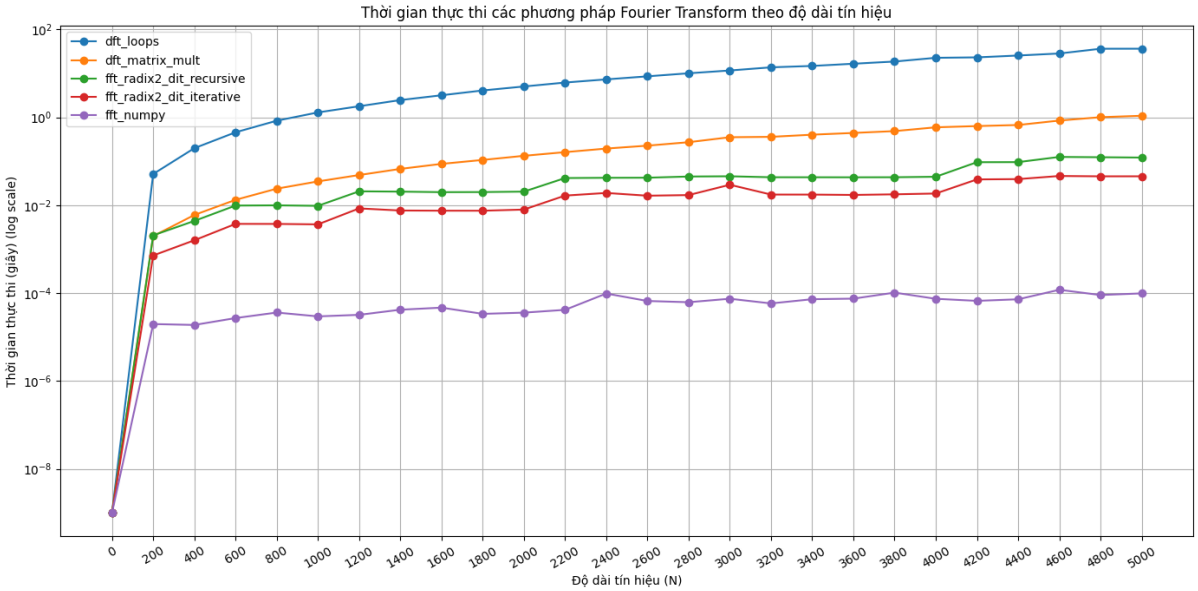
2.8.3 Phân tích độ phức tạp

Độ phức tạp xuất phát từ hai yếu tố chính:

1. **Số tầng chia ($\log_2 N$):** Để chia một dãy N phần tử đến khi chỉ còn 1, cần $\log_2 N$ lần chia. Đây chính là số "tầng"tính toán của thuật toán. Ví dụ, với $N = 8$, cần $\log_2 8 = 3$ tầng.

2. **Phép tính mỗi tầng (N):** Tại mỗi tầng, FFT cần thực hiện N phép toán "butterfly"để kết hợp kết quả từ các phép biến đổi con.

Tổng số phép toán: Với $\log_2 N$ tầng và N phép tính tại mỗi tầng, tổng độ phức tạp là $N \cdot \log_2 N$. (Điều này nhanh hơn đáng kể so với N^2 của DFT truyền thống, đặc biệt với N lớn.)



Hình 4: Thời gian thực thi các phương pháp Fourier Transform theo độ dài tín hiệu.

2.9 Biến đổi Fourier Nhanh Ngược (IFFT) - Dựa trên FFT

2.9.1 Cơ sở lý thuyết

IFFT (Inverse Fast Fourier Transform) là thuật toán hiệu quả để tính Biến đổi Fourier Rời rạc Ngược (IDFT). Một phương pháp hiệu quả là tái sử dụng thuật toán FFT (thuận) dựa trên mối quan hệ giữa DFT và IDFT.

- Công thức IDFT: $x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}nk}$
- Công thức DFT: $X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}nk}$
- Ta có thể tính IFFT bằng cách tính DFT của liên hợp phức $\overline{X[k]}$.
  Áp dụng các tính chất của liên hợp phức: $\overline{(AB)} = \bar{A}\bar{B}$ và $\overline{(e^{j\theta})} = e^{-j\theta}$, ta có thể viết lại DFT của $\overline{X[k]}$ như sau:

$$\text{DFT}\{\overline{X[k]}\} = \sum_{k=0}^{N-1} \overline{X[k]} e^{-j\frac{2\pi}{N}nk} = \overline{\left(\sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}nk} \right)}$$

  Biểu thức trong dấu ngoặc chính là $N \cdot x[n]$ từ công thức IDFT. Do đó:

$$\text{DFT}\{\overline{X[k]}\} = \overline{N \cdot x[n]} = N \cdot \overline{x[n]}$$

  Từ đó suy ra công thức tính $x[n]$:

$$x[n] = \overline{\left(\frac{1}{N} \text{DFT}\{\overline{X[k]}\} \right)} = \frac{1}{N} \overline{\text{DFT}\{\overline{X[k]}\}}$$

2.9.2 Quy trình tính IFFT bằng FFT

1. Lấy liên hợp phức của phổ đầu vào ($\overline{X[k]}$).
2. Áp dụng thuật toán FFT (thuận) lên $\overline{X[k]}$.
3. Lấy liên hợp phức của kết quả FFT.
4. Chia kết quả cuối cùng cho N.

2.9.3 Ví dụ

Tín hiệu đầu vào: $X = [10, -2 + 2j, -2, -2 - 2j]$.

Bước 1: Lấy liên hợp phức của X

$$\overline{X} = [10, -2 - 2j, -2, -2 + 2j]$$

Bước 2: Áp dụng FFT cho $\overline{X}$

- **Chia nhỏ tín hiệu (Decimation-In-Time):**

- *Lớp 1:* $N = 4$
  - Tín hiệu chẵn: $\overline{X}_{\text{even}} = [10, -2]$
  - Tín hiệu lẻ: $\overline{X}_{\text{odd}} = [-2 - 2j, -2 + 2j]$
- *Lớp 2:* $N = 2$
  - Đối với $\overline{X}_{\text{even}} \rightarrow [10]$ và $[-2]$
  - Đối với $\overline{X}_{\text{odd}} \rightarrow [-2 - 2j]$ và $[-2 + 2j]$

- **Kết hợp các kết quả DFT con (Butterfly Operations):**
  - *Kết hợp tạo DFT 2 điểm:*
    - **Đối với** $[10, -2]$:
      $X_{\text{even}}[0] = 10, X_{\text{odd}}[0] = -2, W_2^0 = 1.$

      $$X_0 = 10 + 1 \times (-2) = 8$$
      $$X_1 = 10 - 1 \times (-2) = 12$$

      $\rightarrow \text{DFT}\{[10, -2]\} = [8, 12].$
    - **Đối với** $[-2 - 2j, -2 + 2j]$:
      $X_{\text{even}}[0] = -2 - 2j, X_{\text{odd}}[0] = -2 + 2j, W_2^0 = 1.$

      $$X_0 = (-2 - 2j) + 1 \times (-2 + 2j) = -4$$
      $$X_1 = (-2 - 2j) - 1 \times (-2 + 2j) = -4j$$

      $\rightarrow \text{DFT}\{[-2 - 2j, -2 + 2j]\} = [-4, -4j].$
  - *Kết hợp tạo DFT 4 điểm:*
    $X_{\text{even}}[k] = [8, 12]$ và $X_{\text{odd}}[k] = [-4, -4j]$.
    Với $N = 4$.
    - $k = 0 : W_4^0 = 1$

      $$X_0 = X_{\text{even}}[0] + W_4^0 X_{\text{odd}}[0] = 8 + 1 \times (-4) = 4$$
      $$X_2 = X_{\text{even}}[0] - W_4^0 X_{\text{odd}}[0] = 8 - 1 \times (-4) = 12$$

    - $k = 1 : W_4^1 = -j$

      $$X_1 = X_{\text{even}}[1] + W_4^1 X_{\text{odd}}[1] = 12 + (-j) \times (-4j) = 12 - 4 = 8$$
      $$X_3 = X_{\text{even}}[1] - W_4^1 X_{\text{odd}}[1] = 12 - (-j) \times (-4j) = 12 + 4 = 16$$

- **Kết quả trung gian:** $\text{FFT}(\overline{X}) = [4, 8, 12, 16]$.

Bước 3: Lấy liên hợp phức của kết quả

Kết quả FFT($\overline{X}$) là dãy số thực, nên liên hợp phức của nó là chính nó:

$$\overline{[4, 8, 12, 16]} = [4, 8, 12, 16]$$

Bước 4: Chia kết quả cho N Với $N = 4$:

$$x[n] = \frac{1}{4}[4, 8, 12, 16] = [1, 2, 3, 4]$$

Kết quả này khớp với tín hiệu gốc ban đầu.

3 Ý tưởng khử nhiễu

3.1 Vì sao cần xử lý âm thanh trên miền tần số thay vì trên miền thời gian?

Để hiểu được tại sao âm thanh rất khó xử lý trên miền thời gian, ta cần nhìn xem "hình dạng" của nó trên miền thời gian, tức là âm thanh sau khi được thu, lưu trữ và đọc bằng các phần mềm trên máy tính.

Ta sẽ lần lượt đi qua các đề mục chính sau đây để thấy được hình dạng cuối cùng của âm thanh khi đưa vào trên máy.

3.1.1 Nguyên lý chồng chất và tổng hợp dao động

Âm thanh là dao động của áp suất trong không khí (hoặc môi trường truyền âm), có thể được biểu diễn bằng các hàm sóng hình sin hoặc cos trong toán học.

Nguyên lý chồng chất (superposition principle) phát biểu rằng:

"Đối với mọi hệ tuyến tính, phản ứng tổng thể do hai hay nhiều tác động gây ra chính là tổng các phản ứng mà từng tác động riêng lẻ đó sẽ gây ra." [3.1.1]

Xét ví dụ thực tế: tại một thời điểm t ở một ngã tư đông đúc, ta đặt một micro tại đèn giao thông để thu âm tiếng ồn từ nhiều nguồn khác nhau. Micro sẽ thu được nhiều loại âm thanh cùng lúc như tiếng còi ô tô, tiếng động cơ xe máy, tiếng bước chân người đi bộ, v.v.

Ta sẽ biểu diễn các âm thanh đó dưới dạng các hàm sóng sin như sau:

$$s_1(t) = 0.6 \times \sin(2\pi \times 1000 \times t)$$

[3.1.1] Trích từ Gabriel Popescu, *Principles of Biophotonics, Volume 1: Linear Systems and the Fourier Transform in Optics*, IOP Publishing, 2020, trang 1.

$$s_2(t) = 0.3 \times \sin(2\pi \times 120 \times t)$$

$$s_3(t) = 0.1 \times \sin(2\pi \times 2 \times t)$$

Trong đó:

- $s_1(t)$, $s_2(t)$, $s_3(t)$ lần lượt đại diện cho tiếng còi ô tô, tiếng động cơ xe máy, và tiếng bước chân người đi bộ.
- Các hệ số 0.6, 0.3, 0.1 là biên độ của từng thành phần sóng, biểu thị độ lớn của âm thanh tương ứng.
- Các giá trị 1000, 120, 2 là tần số (đơn vị: Hz), đại diện cho cao độ của từng loại âm thanh.
- t là thời gian (đơn vị: giây).

Mỗi dao động ở trên là một hàm sóng âm riêng biệt. Trong một hệ tuyến tính như không khí (môi trường truyền âm lý tưởng), thì theo nguyên lý chồng chất, dao động tổng hợp tại thời điểm t sẽ bằng tổng các dao động thành phần do từng nguồn âm gây ra và được tính như sau:

$$x(t) = s_1(t)+s_2(t)+s_3(t) = 0.6\times\sin(2\pi\times1000\times t)+0.3\times\sin(2\pi\times120\times t)+0.1\times\sin(2\pi\times2\times t)$$

Như vậy, một tín hiệu âm thanh thực tế thường là sự tổng hợp của nhiều thành phần sóng khác nhau.

3.1.2 Nguyên tắc lấy mẫu rời rạc

Âm thanh là một tín hiệu dao động liên tục theo thời gian. Tuy nhiên, bộ nhớ máy tính là hữu hạn và không thể lưu trữ tín hiệu với số lượng vô hạn. Vì vậy, máy tính chỉ ghi lại giá trị của tín hiệu tại các thời điểm rời rạc và cách đều nhau. Số lượng giá trị của tín hiệu được ghi lại trong mỗi giây được gọi là tần số lấy mẫu (kí hiệu: f_s).

Tiếp tục với ví dụ trước ở phần 3.1.1, giả sử ta sử dụng tần số lấy mẫu $f_s = 44{,}100$ Hz, nghĩa là mỗi giây hệ thống sẽ ghi lại 44,100 giá trị của sóng âm.

Với $f_s = 44{,}100$ Hz, khoảng thời gian giữa hai lần lấy mẫu liên tiếp là:

$$\Delta t = \frac{1}{f_s} = \frac{1}{44{,}100} \approx 22.68\,\mu\text{s}$$

Tức là cứ sau mỗi 22.68 micro giây, tín hiệu sẽ được lấy mẫu một lần.

Ta thử tính giá trị dao động tổng hợp tại thời điểm $t = 22{,}68$ micro giây ($t = 22{,}68 \cdot 10^{-6}$ giây) tức là thời điểm tương ứng với lần lấy mẫu thứ nhất kể từ khi $t = 0$ (mẫu thứ 0):

$$\begin{aligned} x(22{,}68\,\mu s) &= 0.6 \times \sin(2\pi \times 1000 \times 22.68 \times 10^{-6}) \\ &\quad + 0.3 \times \sin(2\pi \times 120 \times 22.68 \times 10^{-6}) \\ &\quad + 0.1 \times \sin(2\pi \times 2 \times 22.68 \times 10^{-6}) \\ &\approx 0.0852 + 0.00513 + 0.0000285 \\ &\approx 0.0904 \end{aligned}$$

Giá trị 0.0904 này chính là biên độ tổng hợp của sóng âm tại thời điểm lấy mẫu $t = 22{,}68$ micro giây. Nó đại diện cho mức dao động của áp suất không khí tại thời điểm đó.

3.1.3 Lượng tử hóa

Lượng tử hóa (Quantization): là quá trình chuyển đổi các giá trị biên độ tổng hợp (số thực) của tín hiệu thành các giá trị rời rạc (số nguyên), nhằm mục đích lưu trữ và xử lý bằng máy tính.

Vì sao cần lượng tử hóa tín hiệu?

- Máy tính không thể lưu trữ giá trị số thực với độ chính xác vô hạn (ví dụ: $0.30141\ldots$).
- Dữ liệu âm thanh cần được mã hóa dưới dạng nhị phân để máy tính có thể hiểu và thao tác.

Tiếp tục với ví dụ trong phần 3.1.2, ta sẽ lượng tử hóa giá trị biên độ tổng hợp $x = 0.0904$ sử dụng mức 16-bit signed integer và lưu giá trị đã lượng tử hóa của x vào máy tính. Quá trình được thực hiện qua lần lượt các bước như sau:

Bước 1: Đầu tiên ta cần chuẩn hóa tín hiệu x về đoạn $[-1, 1]$, lý do chính của việc này là nhằm để tránh hiện tượng biên độ vượt quá ngưỡng cho phép và bị cắt ngưỡng (clipping) làm méo tín hiệu.

Giả sử tín hiệu thực $x = 0.0904$ nằm trong đoạn $[x_{\min}, x_{\max}] = [-3.17,\ 2.84]$. Ta thực hiện công thức chuẩn hóa như sau:

$$
\begin{aligned}
x_{\text{norm}} &= 2 \times \frac{x - x_{\min}}{x_{\max} - x_{\min}} - 1 \\
&= 2 \times \frac{0.0904 - (-3.17)}{2.84 - (-3.17)} - 1 \\
&= 2 \times \frac{3.2604}{6.01} - 1 \\
&\approx 2 \times 0.5425 - 1 \\
&= 1.085 - 1 \\
&= 0.085
\end{aligned}
\tag{3.1.3}
$$

Bước 2: Với hệ thống sử dụng 16-bit signed integer, số mức lượng tử là:

$$2^{16} = 65536$$

Mặc dù có tổng cộng 65536 mức lượng tử, nhưng do sử dụng kiểu số nguyên có dấu theo biểu diễn bù hai (two's complement), các giá trị có thể biểu diễn được nằm trong khoảng:

$$[-32768,\ 32767]$$

Điều này có nghĩa là một nửa số mức được dùng để biểu diễn số âm, một nửa cho số dương (bao gồm cả số 0), giúp mô phỏng đầy đủ dao động âm thanh quanh mức không.

Bước 3: Tính bước lượng tử Δ trên miền đã chuẩn hóa $[-1, 1]$:

$$
\begin{aligned}
\Delta &= \frac{1 - (-1)}{65536} \\
&= \frac{2}{65536} \\
&= 3.0518 \times 10^{-5}
\end{aligned}
$$

Bước 4: Lượng tử hóa giá trị chuẩn hóa $x_{\text{norm}} = 0.085$:

$$q = \text{round}\left(\frac{x_{\text{norm}}}{\Delta}\right)$$

[3.1.3] Trích từ Zach Bobbit, *How to Normalize Data Between -1 and 1*, Statology Blog, truy cập tại: `https://www.statology.org/normalize-data-between-1-and-1/`

$$= \text{round}\left(\frac{0.085}{3.0518 \times 10^{-5}}\right)$$
$$= \text{round}(2785.24) = 2785$$

Bước 5: Sau khi lượng tử hóa, máy tính sẽ lưu lại biên độ $x = 0.0904$ dưới dạng số nguyên $q = 2785$, hoặc mã nhị phân tương ứng với số đó.

3.1.4 Giải lượng tử

Qua ba quá trình chính ở trên gồm: tổng hợp biên độ dao động, lấy mẫu rời rạc và lượng tử hóa, tín hiệu âm thanh đã được lưu vào máy tính dưới dạng các giá trị số nguyên. Các giá trị này có thể được ghi lại trong các tệp âm thanh (như định dạng WAV) để sử dụng cho sau này.

Để sử dụng lại file âm thanh, ta cần dùng các thư viện lập trình để đọc và giải mã tín hiệu. Giả sử ta sử dụng thư viện **soundfile** trong ngôn ngữ Python để đọc tệp âm thanh. Sau khi đọc, tín hiệu sẽ được trả về dưới dạng số thực (float), chứ không phải số nguyên (interger). Điều này là do quá trình *giải lượng tử* (*dequantization*) đã được thực hiện ngầm.

Giải lượng tử (Dequantization): là quá trình khôi phục xấp xỉ giá trị ban đầu từ chỉ số lượng tử (số nguyên) đã được lưu trữ.

Tiếp tục với ví dụ ở phần 3.1.3, sau khi đã tính được chỉ số lượng tử tương ứng với biên độ x là:

$$q = 2785$$

ta thực hiện quá trình giải lượng tử bằng công thức:

$$x_{\text{dequantized}} = q \times \Delta = 2785 \times 3.0518 \times 10^{-5} = 0.0849$$

So sánh với giá trị chuẩn hóa ban đầu là $x_{\text{norm}} = 0.085$, ta thấy:

$$\text{Sai số lượng tử} = x_{\text{norm}} - x_{\text{dequantized}} \approx 0.0001$$

Qua ví dụ trên, ta cần chấp nhận một thực tế rằng quá trình giải lượng tử không thể khôi phục lại tín hiệu ban đầu một cách chính xác tuyệt đối. Thay vào đó, nó chỉ cung cấp một giá trị gần đúng. Sai số lượng tử giữa tín hiệu ban đầu và tín hiệu đã được giải lượng tử là không thể tránh khỏi, bất kể máy tính có hiện đại đến đâu. Đây là giới hạn mang tính nguyên lý trong xử lý tín hiệu số.

3.1.5 Hình dạng cuối cùng của âm thanh

Khi dùng thư viện soundfile trong python để đọc file âm thanh .wav, ta sẽ nhận được một mảng chứa các giá trị số thực nằm trong khoảng [-1, 1] như:

```
array([ -0.0173, 0.0849, -0.0493, ..., 0.0155, -0.0469, -0.0019])
```

Có thể quan sát thấy, sau khi âm thanh được thu và lưu dưới dạng số nguyên (qua lượng tử hóa), các thư viện xử lý âm thanh như soundfile sẽ đọc và giải mã dữ liệu đó thành một mảng các số thực, phản ánh biên độ tổng hợp (đã được giải lượng tử và chuẩn hóa) của sóng âm tại từng thời điểm lấy mẫu.

Tuy nhiên, trong bài toán khử nhiễu âm thanh, các giá trị trong miền thời gian như trên không cung cấp đủ thông tin để nhận diện và loại bỏ nhiễu. Lý do là vì biên độ dao động tại mỗi thời điểm chỉ phản ánh kết quả tổng hợp của nhiều nguồn sóng, bao gồm cả tín hiệu chính và nhiễu nên không thể phân biệt riêng rẽ từng thành phần chỉ dựa trên một giá trị số thực tại mỗi mẫu thời gian.

Để giải quyết vấn đề này, ta cần biểu diễn tín hiệu âm thanh trong một hệ quy chiếu khác, nơi mà các đặc trưng như biên độ, tần số, và pha của từng thành phần sóng âm có thể được tách biệt và quan sát riêng biệt. Phép biến đổi Fourier rời rạc (DFT) chính là lời giải mấu chốt cho bài toán đó.

3.2 Nhận diện nhiễu trong miền tần số

3.2.1 Nhận diện nhiễu bằng biên độ

Sau khi biến đổi DFT ta sẽ có một dãy các hệ số phức X[k] biểu diễn thành phần tần số của tín hiệu, Ta thường quan sát biên độ phổ của tín hiệu |X[k]| để phát hiện nhiễu.

Giả sử mỗi hệ số $X[k]$ có dạng:

$$X[k] = a + bj \quad \text{với } j^2 = -1$$

thì biên độ phổ $|X[k]|$ được tính theo công thức:

$$|X[k]| = \sqrt{a^2 + b^2}$$

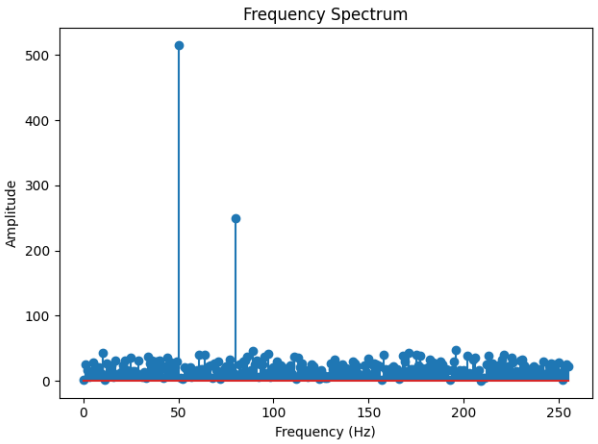
Hoặc tổng quát hơn, sử dụng phần thực và phần ảo của $X[k]$:

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

Dựa vào đặc điểm về biên độ, ta có thể nhận diện và phân loại một số loại nhiễu sau:

Loại nhiễu	**Biểu hiện trong miền tần số**
Nhiễu trắng (white noise)	Trải đều trên toàn bộ dải tần, biên độ nhỏ nhưng liên tục
Nhiễu tần số thấp (hum noise, ví dụ 50Hz)	Một đỉnh rõ ràng ở tần số thấp, thường là 50Hz hoặc bội số của nó
Nhiễu đột biến (impulse noise)	Xuất hiện ở tất cả tần số, biên độ rất cao tại một vài điểm
Nhiễu giọng người khác xen lẫn	Có dạng phổ gần giống tiếng người (300Hz–3400Hz), nhưng cấu trúc không đều hoặc méo
Nhiễu nền nhẹ (background noise)	Biên độ thấp, rải rác, không có cấu trúc rõ ràng

Bảng 1: Nhận diện các loại nhiễu phổ biến trong miền tần số



Hình 5: Đồ thị biên độ - tần số

Có thể thấy tín hiệu chính tập trung rõ ràng tại 2 đỉnh lớn ở khoảng 50 Hz và 80 Hz, trong khi phần nhiễu (white noise) lan rộng và phân bố ngẫu nhiên ở hầu hết các tần số còn lại với biên độ thấp.

3.2.2 Nhận diện nhiễu bằng mật độ phổ năng lượng

Ngoài cách sử dụng đặc trưng biên độ trên phổ tần số để giải thích và phân loại nhiễu, ta còn một cách khác để phân loại nhiễu đó là dựa vào mật độ phổ năng

lượng (Power Spectral Density).

Mật độ phổ năng lượng (PSD) - cũng là một đặc trưng của âm thanh trên miền tần số, nó cho biết cách năng lượng (công suất) của tín hiệu phân bố theo tần số.

Giả sử ta có tín hiệu rời rạc $x[n]$ với độ dài N, sau khi được biến đổi qua DFT trở thành dãy hệ số phức $X[k]$ như sau:

$$X[k] = \text{DFT}(x[n])$$

Mật độ phổ công suất (PSD) tại tần số tương ứng với chỉ số k được tính bằng:

$$\text{PSD}[k] = \frac{1}{N \cdot f_s} \cdot |X[k]|^2$$

hoặc có thể chuẩn hóa theo năng lượng toàn phần:

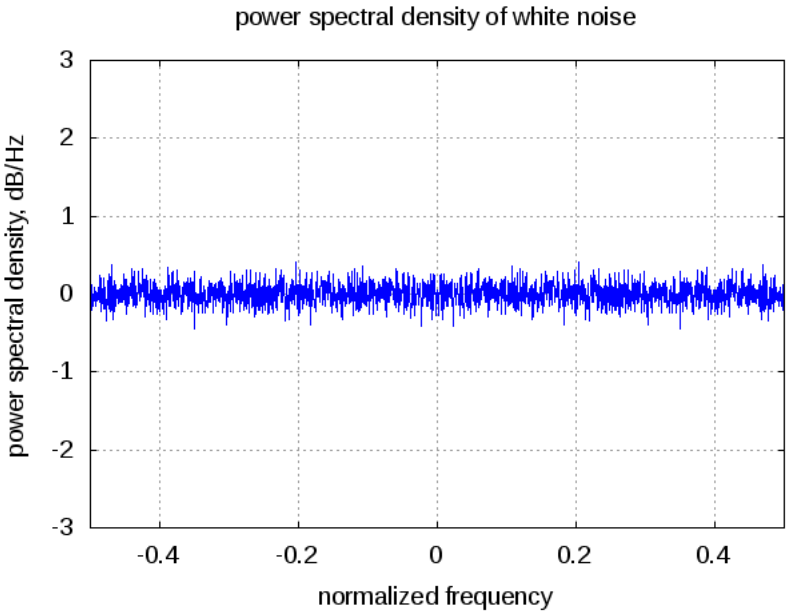
$$\text{PSD}[k] = \frac{1}{N^2} \cdot |X[k]|^2$$

Trong đó:

- $X[k]$: hệ số phổ tại tần số k
- f_s: tần số lấy mẫu (sampling rate)
- N: số mẫu trong tín hiệu $x[n]$
- $|X[k]|$: biên độ phổ (modulus của số phức)

Loại nhiễu	**PSD theo tần số**	**Giảm dB/octave**	**Âm thanh tương tự**
White noise	Phẳng (bằng nhau mọi tần số)	0 dB	TV rè rè, quạt điện mạnh
Pink noise	Giảm dần theo $\frac{1}{f}$	3 dB	Gió nhẹ, mưa rơi đều
Brown noise (hoặc Red noise)	Giảm mạnh theo $\frac{1}{f^2}$	6 dB	Thác nước, tiếng sấm xa
Blue noise	Tăng theo tần số f	+3 dB	Âm cao chói, tiếng rít
Violet noise	Tăng mạnh theo f^2	+6 dB	Tiếng cực cao, hiếm gặp
Grey noise	Điều chỉnh theo cảm nhận thính giác	Không đều	Mô phỏng âm thanh đều với tai người

Bảng 2: Phân loại nhiễu theo mật độ phổ công suất (PSD)



Hình 6: Đồ thị PSD - tần số

Đường PSD thể hiện nhiễu trắng vì năng lượng trải đều toàn phổ, không tập trung vào một dải cụ thể nào. **Chú ý:** Nhiễu trắng (White noise) theo biên độ phổ và theo PSD là hai cách mô tả khác nhau cho cùng một loại tín hiệu: nhiễu có năng lượng đều trên mọi tần số.

Khía cạnh	**Biên độ phổ** $\lvert X[k]\rvert$	**PSD**
Định nghĩa	Biên độ FFT tại từng tần số rời rạc k	Trung bình công suất theo tần số, xấp xỉ $\lvert X(f)\rvert^2$ chuẩn hóa theo thời gian
Biểu hiện white noise	Dao động mạnh, không đều, hình răng cưa	Phẳng, gần như hằng số trên toàn dải tần
Đặc trưng	Chứa yếu tố ngẫu nhiên, thay đổi mỗi lần tính FFT	Kỳ vọng ổn định, phản ánh năng lượng trung bình theo tần số
Tính chất	Quan sát một đoạn tín hiệu cụ thể	Mô tả tổng thể tín hiệu dừng
Ứng dụng	Phân tích nhanh một đoạn tín hiệu	Mô hình hóa, nhận diện nhiễu và lọc phổ

Bảng 3: So sánh biểu hiện của white noise theo biên độ phổ và PSD

4 Thuật toán

4.1 Biến đổi tín hiệu sang miền tần số (DFT)

4.1.1 Công thức biến đổi DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1$$

Trong đó:

- $x[n]$: tín hiệu đầu vào ở miền thời gian (thời điểm n)
- $X[k]$: hệ số phổ ở miền tần số tại chỉ số k
- N: tổng số mẫu (độ dài tín hiệu)
- k: chỉ số tần số (biểu diễn các tần số rời rạc)
- n: chỉ số thời gian (biểu diễn các thời điểm rời rạc)
- j: đơn vị ảo, $j^2 = -1$
- $e^{-j\frac{2\pi}{N}kn}$: nhân tử cơ sở của biến đổi Fourier, biểu diễn sự xoay pha (rotating phasor)

4.1.2 Triển khai thuật toán DFT

Đầu vào:

- $x = [x_0, x_1, \cdots, x_{N-1}]$: tín hiệu âm thanh theo thời gian
- N: Số mẫu

Đầu ra:

- $X = [X_0, X_1, \cdots, X_{N-1}]$: Phổ tần số

Các bước thực hiện:

1. Tạo ma trận $F \in \mathbb{C}^{N \times N}$ với mỗi phần tử:

$$F_{k,n} = e^{-j\frac{2\pi kn}{N}}, \quad 0 \leq k, n \leq N-1$$

2. Tính:

$$X = F \cdot x$$

3. Trả về X

4.2 Tái tạo tín hiệu (IDFT)

4.2.1 Công thức khôi phục tín hiệu IDFT

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi kn}{N}}, \quad n = 0, 1, \cdots, N-1$$

Trong đó:

- $X[k]$: Tín hiệu ở miền tần số (phổ DFT)
- $x[n]$: Tín hiệu thu được sau khi áp dụng IDFT (dạng thời gian)
- N: Số mẫu
- k: Chỉ số tần số
- n: Chỉ số thời gian
- j: Đơn vị ảo, $j^2 = -1$
- $e^{j\frac{2\pi kn}{N}}$: Nhân tử cơ sở của IDFT, biểu diễn sự xoay pha ngược

4.2.2 Triển khai thuật toán IDFT

Đầu vào:

- $X = [X_0, X_1, \ldots, X_{N-1}]$: Phổ tín hiệu
- N: Số mẫu

Đầu ra:

- $x = [x_0, x_1, \ldots, x_{N-1}]$: Tín hiệu thời gian

Các bước thực hiện:

1. Tạo ma trận $F' \in \mathbb{C}^{N \times N}$ với mỗi phần tử:

$$F'_{n,k} = e^{j\frac{2\pi kn}{N}}, \quad 0 \leq k, n \leq N-1$$

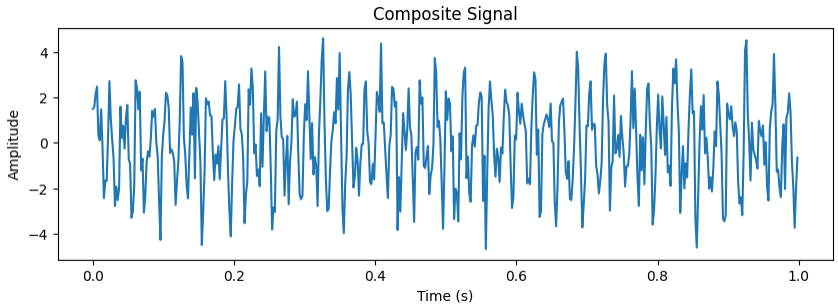
2. Tính:

$$x = \frac{1}{N} \cdot F' \cdot X$$

3. Trả về x

4.3 Thực hiện thuật toán DFT và IDFT đã triển khai

Ví dụ: 5.8.3 (tham khảo[4]) Giả sử ta đặt một microphone dưới một chiếc trực thăng đang lơ lửng, trong vòng 1 giây micro ghi lại tín hiệu âm thanh như biểu đồ hình 7 . Tín hiệu có nhiều dao động, nhưng do nhiễu nên không rõ ràng.



Hình 7: Composite signal

Mục tiêu: Dùng DFT để phân tích tín hiệu và tìm ra những tần số chính.

Ta giả định tín hiệu thu được có dạng:

$y(\tau) = \cos(2\pi \cdot 80\tau) + 2\sin(2\pi \cdot 50\tau) + \text{Noise}$

- Dao động thật: cos 80Hz và sin 50Hz

- Noise: ngẫu nhiên, che khuất dao động chính

- Tín hiệu được lấy mẫu tại 512 điểm đều nhau trong khoảng thời gian 1 giây:

$$t = \left\{0, \frac{1}{512}, \frac{2}{512}, \ldots, \frac{511}{512}\right\}$$

- Vector tín hiệu mẫu:

$$x = \left[y(0), y\left(\frac{1}{512}\right), y\left(\frac{2}{512}\right), \ldots, y\left(\frac{511}{512}\right)\right]$$

Code sinh dữ liệu tín hiệu mẫu

```
n = 512             # số mẫu
T = 1.0             # thời lượng (giây)
t = np.linspace(0, T, n, endpoint=False)
f1 = 80             # tần số 1 (Hz)
f2 = 50             # tần số 2 (Hz)

np.random.seed(42)
noise = np.random.normal(0, 1, n)
y = np.cos(2 * np.pi * f1 * t) + 2 * np.sin(2 * np.pi * f2 * t) + noise
```

Phân tích tín hiệu bằng DFT

Code hàm DFT

```python
def dft(x):
    """
    Discrete Fourier Transform (DFT) using Fourier Matrix.
    x: array-like, shape (n,)
    return: array of complex numbers, shape (n,)
    """
    x = np.asarray(x, dtype=float)  # chuyển về dạng float
    n = x.shape[0]  # lấy số lượng sample

    # Tạo ma trận Fourier F
    F = np.zeros((n, n), dtype=complex)
    for j in range(n):  # Duyệt qua các dòng (chỉ số thời gian)
        for k in range(n):  # Duyệt qua các cột (chỉ số tần số)
            F[j][k] = np.exp(-2j * np.pi * j * k / n)  # Tính F[j][k]

    # Tính X: F[n x n] @ x [n x 1] = X [n x 1]
    X = F @ x

    return X
```

Sử dụng DFT để chuyển đổi tín hiệu từ miền thời gian sang miền tần số:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\frac{2\pi kn}{N}}, \quad k = 0, 1, \ldots, N-1$$

Trong đó:

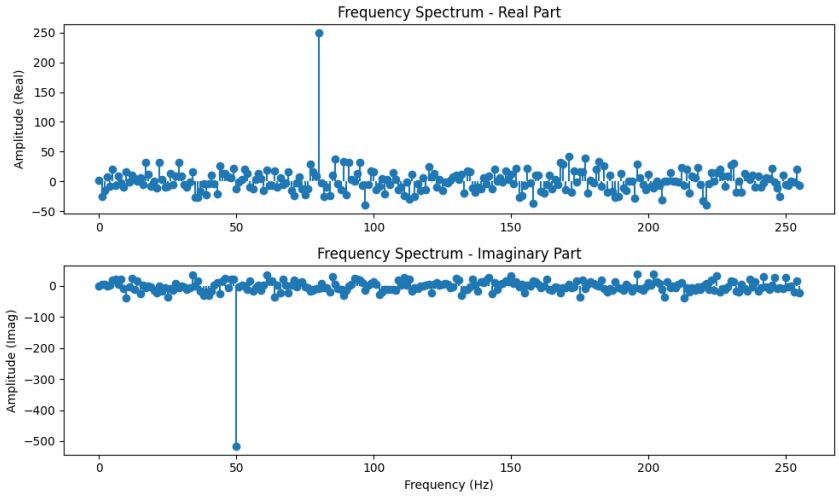
- x_n: Giá trị tín hiệu tại mẫu thứ n.
- X_k: Giá trị tại tần số k.
- $N = 512$: Số lượng điểm mẫu.

Từ kết quả DFT thông qua biểu đồ miền tần số, giúp xác định các thành phần dao động chính.

Sau khi áp dụng DFT:

- Thành phần thực (**Real Axis**): Xuất hiện một đỉnh tại tần số 80 Hz.
- Thành phần ảo (**Imaginary Axis**): Xuất hiện một đỉnh tại tần số 50 Hz.

Biểu đồ miền tần số được thể hiện như hình 8.



Hình 8: Phân tích miền tần số bằng DFT

Thông qua thuật toán DFT đã giúp xác định các thành phần dao động chính trong tín hiệu:

$$y(\tau) = \cos(2\pi \cdot 80\tau) + 2\sin(2\pi \cdot 50\tau) + \text{Noise}$$

Tín hiệu bao gồm một dao động cosine (80 Hz) với biên độ 1 và một dao động sine (50 Hz) với biên độ 2.

Sau khi phân tích tín hiệu nhiễu bằng DFT, ta sử dụng IDFT để trả ngược tín hiệu về miền thời gian. IDFT được định nghĩa như sau:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{i\frac{2\pi kn}{N}}$$

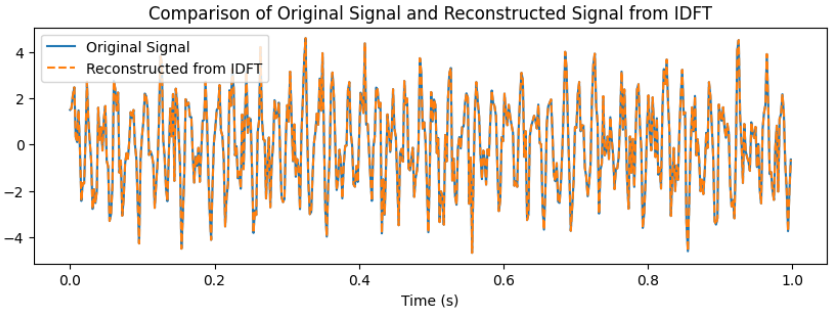
Code hàm IDFT

```
def idft(X):
    """
    Inverse Discrete Fourier Transform (IDFT) from scratch.
    X: array-like, shape (n,)
    return: array of complex numbers, shape (n,)
    """
    X = np.asarray(X, dtype=complex)  # Đảm bảo dữ liệu đầu vào là số phức
    n = X.shape[0]  # lấy số lượng sample

    # Tạo ma trận Inverse Fourier F
    F_inv = np.zeros((n, n), dtype=complex)
    for j in range(n):  # Duyệt qua các dòng (chỉ số thời gian)
        for k in range(n):  # Duyệt qua các cột (chỉ số tần số)
            F_inv[j][k] = np.exp(2j * np.pi * j * k / n)  # Tính F[j][k]

    # Tính X: F_inv [n x n] @ X [n x 1] = x [n x 1]
    x = F_inv @ X
    return x / n
```

Tín hiệu khôi phục được so sánh với tín hiệu gốc như hình 9.



Hình 9: Khôi phục tín hiệu bằng IDFT

Có thể thấy rằng tín hiệu đã được khôi phục lại một cách hoàn hảo. Giống như dao động ban đầu. Nhưng *làm sao DFT lại "nhìn thấy" được các tần số ẩn bên trong một tín hiệu có nhiều nhiễu đến vậy?*

Tại sao F_n tiết lộ các tần số ẩn?

Để hiểu tại sao F_n lại có thể tiết lộ các thành phần dao động trong tín hiệu, xét tương tác giữa F_n với các vector cos/sin rời rạc:

$$\cos(2\pi ft) = \left(\cos(\tfrac{2\pi f\cdot 0}{n}) \vdots \cos(\tfrac{2\pi f\cdot (n-1)}{n})\right), \ \sin(2\pi ft) \ = \left(\sin(\tfrac{2\pi f\cdot 0}{n}) \vdots \sin(\tfrac{2\pi f\cdot (n-1)}{n})\right),$$

với $t = (0, \frac{1}{n}, \ldots, \frac{n-1}{n})^T$.

Ta có:

$$e^{i2\pi ft} = \cos(2\pi ft) + i\sin(2\pi ft), \quad e^{-i2\pi ft} = \cos(2\pi ft) - i\sin(2\pi ft)$$

Dễ chứng minh:

$$F_n e^{i2\pi ft} = ne_f, \quad F_n e^{-i2\pi ft} = ne_{n-f}$$

Từ đó suy ra:

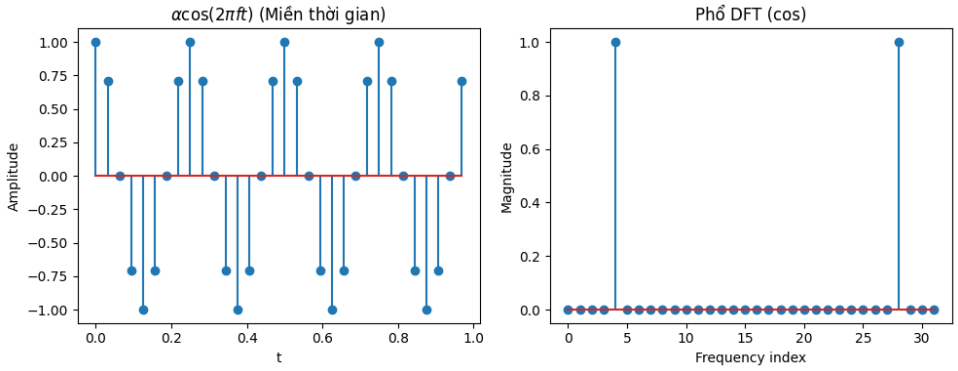
$$\frac{2}{n}F_n(\alpha\cos(2\pi ft)) = \alpha e_f + \alpha e_{n-f} \qquad (5.8.5\ [4])$$

$$\frac{2}{n}F_n(\beta\sin(2\pi ft)) = -\beta i e_f + \beta i e_{n-f} \qquad (5.8.6\ [4])$$

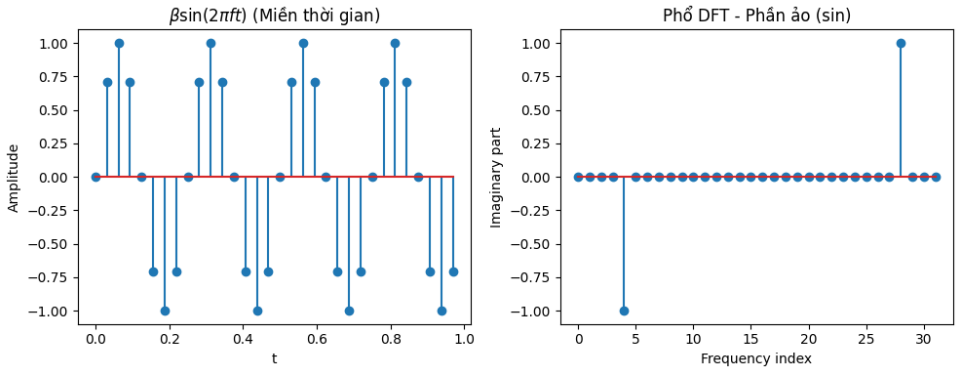
Nghĩa là, trong phổ DFT:

$\cos(2\pi ft)$ tạo xung tại f và $n-f$ trong **phần thực** như hình 10.

$\sin(2\pi ft)$ tạo xung tại f và $n - f$ trong **phần ảo** như hình 11.



Hình 10: Thành phần $\alpha \cos(2\pi ft)$ tạo hai xung tại f và $n - f$



Hình 11: Thành phần $\beta \sin(2\pi ft)$ tạo hai xung đối nhau trong phần ảo

Kết luận

Tổng quát hóa, nếu tín hiệu có dạng:

$$x(\tau) = \sum_k \alpha_k \cos(2\pi f_k \tau) + \beta_k \sin(2\pi f_k \tau)$$

thì:

$$\frac{2}{n} F_n x = \sum_k \alpha_k (e_{f_k} + e_{n-f_k}) + i \sum_k \beta_k (-e_{f_k} + e_{n-f_k})$$

Do đó, có thể **trực tiếp đọc được tần số và biên độ** từ phổ DFT. **Ví dụ: 5.8.3 (tham khảo[4])** cho thấy rõ ràng hai tần số chiếm đa số trong tín hiệu gốc, bị che bởi nhiễu, nay đã hiển thị rõ ràng.

4.4 Thuật toán FFT dạng đệ quy Cooley-Tukey Radix-2 DIT

Mô tả: Đây là triển khai đệ quy của thuật toán FFT Cooley-Tukey Radix-2 DIT (Decimation-In-Time). Nó hoạt động bằng cách liên tục chia đôi tín hiệu đầu vào cho

đến khi đạt đến trường hợp cơ bản (tín hiệu có 1 điểm), sau đó kết hợp các kết quả DFT con lên.

Đầu vào:

- **Tín hiệu đầu vào**: Một mảng (hoặc danh sách) các số đại diện cho tín hiệu miền thời gian. Hàm sẽ tự động thêm padding bằng 0 nếu độ dài không phải là lũy thừa của 2.

Đầu ra:

- **Hệ số FFT**: Một mảng chứa các hệ số FFT (tức là DFT) trong miền tần số, có độ dài là lũy thừa của 2 gần nhất và lớn hơn hoặc bằng độ dài ban đầu.

Các bước thực hiện:

1. **Tiền xử lý (Padding):**
    - Xác định độ dài ban đầu của tín hiệu.
    - Tìm độ dài mới (padding): lũy thừa của 2 nhỏ nhất lớn hơn hoặc bằng độ dài ban đầu.
    - Nếu độ dài mới khác độ dài ban đầu, thêm các số 0 vào cuối tín hiệu để đạt độ dài mới.
2. **Hàm Đệ quy:**
    - **Đầu vào:** Một đoạn tín hiệu con.
    - **Đầu ra:** DFT của đoạn tín hiệu con đó.
    - **Trường hợp cơ bản:** Nếu độ dài của tín hiệu con nhỏ hơn hoặc bằng 1, trả về chính tín hiệu con đó (DFT của 1 điểm là chính nó).
    - **Chia tín hiệu:**
        - Gọi đệ quy hàm cho các phần tử có chỉ số chẵn của tín hiệu con.
        - Gọi đệ quy hàm cho các phần tử có chỉ số lẻ của tín hiệu con.
    - **Kết hợp (Butterfly Operation):**
        - Khởi tạo một mảng có kích thước bằng độ dài tín hiệu con để chứa kết quả.
        - Lặp qua các chỉ số cần thiết:
            * Tính toán hệ số xoắn (twiddle factor) dựa trên chỉ số hiện tại và kích thước tín hiệu con.

* Áp dụng phép toán "butterfly"để kết hợp kết quả từ các phần chẵn và lẻ, tính ra các phần tử của DFT con.

- Trả về mảng DFT đã được tính.

3. **Thực thi:**

- Gọi hàm đệ quy với tín hiệu đã được padding để nhận về kết quả FFT cuối cùng.

Code

```
    def _apply_fft_radix2_dit_recursive(signal_vector):
        """
        Triển khai Biến đổi Fourier Nhanh (FFT) Cooley-Tukey DIT.
        """
        x = signal_vector
        original_N = len(x)
        # Tính toán độ dài mới (lũy thừa của 2) để padding nếu cần
        padded_N = 1
        while padded_N < original_N:
            padded_N *= 2
        # Nếu độ dài tín hiệu không phải lũy thừa của 2, thêm padding
        if padded_N != original_N:
            x = np.pad(x, (0, padded_N - original_N), 'constant')

        def recursive_function(signal_vector):
            x = signal_vector
            N = len(x)

            if N <= 1: # Trường hợp cơ bản: DFT 1 điểm chính là giá trị đó
                return x # Trả về x

            # Chia tín hiệu thành các phần chẵn và lẻ
            x_even = recursive_function(x[0::2])     # Đệ quy cho phần chẵn
            x_odd = recursive_function(x[1::2])      # Đệ quy cho phần lẻ

            X = np.zeros(N, dtype=np.complex128) # Kết quả tổng hợp

            # Kết hợp các kết quả DFT con
            for k in range(N // 2):
                twiddle = np.exp(-1j * 2 * np.pi * k / N)
                X[k] = x_even[k] + twiddle * x_odd[k]
                X[k + N // 2] = x_even[k] - twiddle * x_odd[k]
            return X
        return recursive_function(x)
```

4.5 Thuật toán FFT dạng vòng lặp Cooley-Tukey Radix-2 DIT

Mô tả: Đây là triển khai lặp (không đệ quy) của thuật toán FFT Cooley-Tukey Radix-2 DIT. Nó đạt được hiệu quả bằng cách sắp xếp lại dữ liệu đầu vào thông qua hoán vị đảo bit (bit-reversal permutation) trước, sau đó thực hiện các phép toán "butterfly"theo từng "stage"(giai đoạn) tăng dần.

Đầu vào:

- **Tín hiệu đầu vào**: Một mảng (hoặc danh sách) các số đại diện cho tín hiệu miền thời gian. Hàm sẽ tự động thêm padding bằng 0 nếu độ dài không phải là lũy thừa của 2.

Đầu ra:

- **Hệ số FFT**: Một mảng chứa các hệ số FFT (tức là DFT) trong miền tần số, có độ dài là lũy thừa của 2 gần nhất và lớn hơn hoặc bằng độ dài ban đầu.

Các bước thực hiện:

1. **Tiền xử lý (Padding):**
    - Tương tự như phiên bản đệ quy, tính độ dài đã được padding và thêm padding bằng 0 nếu cần.
    - Gán tín hiệu đã được padding vào một mảng kết quả và chuyển đổi sang kiểu số phức.
2. **Hoán vị đảo bit (Bit-Reversal Permutation):**
    - Sắp xếp lại các phần tử của mảng sao cho chúng nằm ở vị trí cần thiết cho các phép toán butterfly sau này. Vị trí mới của một phần tử được tìm bằng cách đảo ngược các bit trong biểu diễn nhị phân của chỉ số gốc của nó.
    - Lặp qua các phần tử và thực hiện hoán đổi vị trí nếu cần thiết, đảm bảo mỗi cặp chỉ được hoán đổi một lần.
3. **Tính toán Butterfly theo giai đoạn (Iterative Butterfly Computation):**
    - Xác định tổng số giai đoạn cần thiết, dựa trên $\log_2$ của độ dài tín hiệu.
    - Lặp qua từng giai đoạn, từ 1 đến tổng số giai đoạn:
        - Xác định kích thước của các nhóm con (sub-DFT) trong giai đoạn hiện tại.
        - Tính toán hệ số xoắn cơ bản (twiddle factor base) cho giai đoạn này.
        - Lặp qua các khối dữ liệu trong mảng:
            * Khởi tạo hệ số xoắn hiện tại cho mỗi khối.
            * Lặp qua các cặp butterfly trong mỗi khối:
                · Thực hiện phép toán "butterfly", kết hợp hai phần tử và áp dụng hệ số xoắn để tạo ra hai kết quả mới.
                · Cập nhật hệ số xoắn cho cặp butterfly tiếp theo trong cùng khối.

4. **Trả về:**

   - Trả về mảng chứa các hệ số FFT đã được tính toán.

Code

```
    def _apply_fft_radix2_dit_iterative(signal_vector):
        """
        Triển khai Biến đổi Fourier Nhanh (FFT) Cooley-Tukey DIT (lặp).
        """
        x = signal_vector
        original_N = len(x)
        # Tính toán độ dài mới (lũy thừa của 2) để padding nếu cần
        padded_N = 1
        while padded_N < original_N:
            padded_N *= 2
        # Nếu độ dài tín hiệu không phải lũy thừa của 2, thêm padding
        if padded_N != original_N:
            x = np.pad(x, (0, padded_N - original_N), 'constant')
        N = len(x)

        # 1. Hoán vị đảo bit
        X = x.astype(np.complex128) # Chuyển sang kiểu phức để tính toán
        for i in range(1, N - 1):
            j = 0
            m = i
            p = N >> 1 # Tương đương N // 2
            while p > 0:
                j += (m & 1) * p
                m >>= 1
                p >>= 1
            if j > i:
                X[i], X[j] = X[j], X[i]

        # 2. Tính toán butterfly
        num_stages = int(np.log2(N))
        for stage in range(1, num_stages + 1):
            L = 2**stage
            L_half = L // 2
            twiddle_base = np.exp(-2j * np.pi / L)
            for k in range(0, N, L):
                twiddle = 1.0 + 0.0j
                for j in range(L_half):
                    t = twiddle * X[k + j + L_half]
                    u = X[k + j]
                    X[k + j] = u + t
                    X[k + j + L_half] = u - t
                    twiddle *= twiddle_base
        return X
```

4.6 Thuật toán biến đổi Fourier Nhanh Ngược (IFFT) - Dựa trên FFT

Mô tả: Hàm này tính toán IFFT của một phổ tín hiệu đầu vào bằng cách sử dụng thuật toán FFT trên phổ liên hợp, sau đó lấy liên hợp của kết quả và chia tỉ lệ.

Đầu vào:

- **Phổ tín hiệu đầu vào**: Một mảng (hoặc danh sách) các số phức đại diện cho phổ

tín hiệu trong miền tần số $X[k]$. Hàm này giả định phổ đã có độ dài là lũy thừa của 2 (hoặc đã được padding bên trong hàm FFT gọi tới).

Đầu ra:

- **Tín hiệu khôi phục**: Một mảng chứa các mẫu tín hiệu trong miền thời gian $x[n]$ đã được khôi phục. Độ dài của nó sẽ được cắt về độ dài ban đầu nếu có padding.

Các bước thực hiện:

1. **Khởi tạo:**
   - Gán phổ đầu vào cho biến đại diện.
   - Xác định độ dài của phổ tín hiệu đã được padding.
   - Xác định độ dài ban đầu của tín hiệu miền thời gian (cần thiết để cắt bỏ padding sau này).
2. **Liên hợp Phức và FFT thuận:**
   - Tính liên hợp phức của phổ đầu vào.
   - Áp dụng thuật toán FFT theo phương pháp Decimation-In-Time (DIT) đệ quy lên phổ liên hợp đã tính ở trên.
3. **Liên hợp Phức và Chia tỉ lệ:**
   - Lấy liên hợp phức của kết quả FFT vừa thu được.
   - Chia kết quả cuối cùng cho tổng số điểm (độ dài đã được padding).
4. **Cắt bỏ Padding (nếu có):**
   - Nếu độ dài của tín hiệu khôi phục sau FFT lớn hơn độ dài ban đầu (do quá trình padding), cắt bỏ phần dữ liệu được thêm vào.
5. **Trả về:**
   - Trả về mảng chứa các mẫu tín hiệu đã được khôi phục.

Code

```
    def _apply_ifft_radix2_dit_recursive(spectrum, signal_vector):
        """
        IFFT Cooley-Tukey DIT (đệ quy).
        """
        X = spectrum
        padded_N = len(X)
        original_N = len(signal_vector)
        # Sử dụng FFT dạng đệ quy trên phổ liên hợp, sau đó lấy liên hợp kết quả
        x_conj = _apply_fft_radix2_dit_recursive(np.conj(X))
        x_reconstructed = np.conj(x_conj) / padded_N
        if len(x_reconstructed) > original_N:
```

```
            x_reconstructed = x_reconstructed[:original_N]
        return x_reconstructed

    def _apply_ifft_radix2_dit_iterative(spectrum, signal_vector):
        """
        IFFT Cooley-Tukey DIT (lặp).
        """
        X = spectrum
        padded_N = len(X)
        original_N = len(signal_vector)
        # Sử dụng FFT dạng vòng lặp trên phổ liên hợp, sau đó lấy liên hợp kết quả
        x_conj = _apply_fft_radix2_dit_iterative(np.conj(X))
        x_reconstructed = np.conj(x_conj) / padded_N
        if len(x_reconstructed) > original_N:
            x_reconstructed = x_reconstructed[:original_N]
        return x_reconstructed
```

4.7 Đánh giá thuật toán

Sau khi hoàn tất việc cài đặt và kiểm thử các thuật toán, nhóm thực hiện phân tích, so sánh hiệu năng và đã đưa ra các đánh giá tổng quan về độ chính xác giữa các phương pháp đã đề ra dựa trên các sai số được thể hiện như hình 12.

```
--------------------------------------------------------------------------------
Đang kiểm tra phương pháp: dft
    Kết quả kiểm tra np.allclose: ĐẠT
    Sai số tuyệt đối lớn nhất (Max Diff): 2.83e-12
    Sai số tuyệt đối trung bình (Avg Abs Diff): 7.72e-13
    Sai số bình phương trung bình (MSE): 9.45e-25
--------------------------------------------------------------------------------
Đang kiểm tra phương pháp: fft_radix2_dit_recursive
    Kết quả kiểm tra np.allclose: ĐẠT
    Sai số tuyệt đối lớn nhất (Max Diff): 2.84e-14
    Sai số tuyệt đối trung bình (Avg Abs Diff): 5.98e-15
    Sai số bình phương trung bình (MSE): 9.53e-29
--------------------------------------------------------------------------------
Đang kiểm tra phương pháp: fft_radix2_dit_iterative
    Kết quả kiểm tra np.allclose: ĐẠT
    Sai số tuyệt đối lớn nhất (Max Diff): 1.85e-13
    Sai số tuyệt đối trung bình (Avg Abs Diff): 4.95e-14
    Sai số bình phương trung bình (MSE): 3.99e-27
--------------------------------------------------------------------------------
```

Hình 12: Phân tích sai số thuật toán

Thông qua sai số tìm được, dễ dàng nhận thấy rằng tất cả các phương pháp đều hoạt động chính xác và có khả năng khôi phục tín hiệu đầu vào một cách gần như hoàn hảo. Các kiểm tra về sai số tuyệt đối và sai số bình phương trung bình đều cho kết quả rất nhỏ, nằm trong giới hạn cho phép của tính toán số học trên máy tính.

Đặc biệt, các thuật toán FFT cho thấy ưu thế rõ rệt về độ chính xác so với DFT truyền thống. Trong đó, phương pháp FFT đệ quy có độ chính xác vượt trội, trong khi FFT vòng lặp lại có cấu trúc phù hợp hơn cho tối ưu hóa hiệu năng thực thi.

Nhìn chung, việc xây dựng các thuật toán từ đầu không chỉ giúp nhóm hiểu sâu hơn về bản chất toán học của biến đổi Fourier, mà còn chứng minh được tính đúng đắn và độ tin cậy của các thuật toán khi áp dụng vào xử lý tín hiệu thực tế.

5 Thực hiện khử nhiễu âm thanh bằng các phương pháp khác nhau

5.1 Phương pháp lọc kết hợp ngưỡng hóa, band-stop, Gaussian qua FFT convolve

Dựa trên lý thuyết về tích chập trong xử lý tín hiệu, đặc biệt là việc tích chập trong miền thời gian tương ứng với nhân trong miền tần số, nhóm em tận dụng `fftconvolve` để tăng tốc độ xử lý so với tích chập truyền thống. Bên cạnh đó, việc sử dụng DFT và IDFT giúp trực quan hóa thành phần tần số của tín hiệu, từ đó dễ dàng xác định các dải nhiễu cố định cần loại bỏ. Từ việc quan sát phổ tín hiệu trong từng đoạn (frequency spectrum), nhóm tiến hành xác định các vùng nhiễu và đề xuất phương pháp lọc kết hợp gồm ba bước như sau:

(1) Ngưỡng hóa biên độ trong một đoạn thời gian cụ thể để loại bỏ các giá trị bất thường (threshold).

(2) Áp dụng bộ lọc chặn dải (band-stop) để loại bỏ nhiễu tần số cố định.

(3) Làm mượt toàn bộ tín hiệu bằng hàm Gauss sử dụng tích chập nhanh (`fftconvolve`) nhằm giảm nhiễu mềm và giữ tín hiệu ổn định.

Cách thực hiện trong mã nguồn

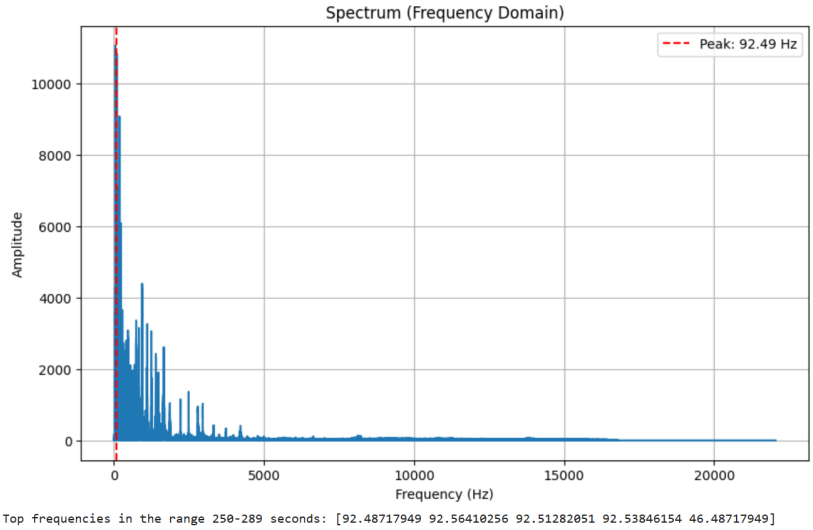
Các bước xử lý trong hàm `filter_combined()` được triển khai như sau:

- **Đọc và tiền xử lý:**
  - `sf.read(input_file)`: Đọc tín hiệu âm thanh từ file. Nếu là stereo thì chuyển sang mono.
  - Chuyển thời gian (giây) thành chỉ số mẫu (sample index) để xác định đoạn cần xử lý.
- **Ngưỡng hóa biên độ:**
  - `segment[segment > threshold] = threshold`: Cắt các giá trị biên độ quá cao (hoặc quá thấp).
  - Chỉ áp dụng trên đoạn tín hiệu từ `start_time` đến `end_time`.
- **Lọc chặn dải (Band-Stop):**
  - Dùng hàm `butter(..., btype='bandstop')` để tạo bộ lọc Butterworth.
  - Dùng `filtfilt(...)` để lọc tín hiệu một cách mượt mà, không gây trễ pha.
- **Làm mượt Gaussian trên toàn bộ tín hiệu:**

  - Tạo kernel Gaussian bằng công thức hàm mũ.
  - Dùng `fftconvolve(...)` để tích chập nhanh tín hiệu với kernel (hiệu quả hơn so với tích chập thông thường).
- **Chuẩn hóa và lưu lại:**
  - Tín hiệu được chuẩn hóa lại để đảm bảo biên độ không vượt ngưỡng.
  - Ghi tín hiệu đã xử lý ra file mới bằng `sf.write()`.
- **Trực quan hóa và kiểm tra:**
  - Vẽ 3 đồ thị: tín hiệu gốc, đoạn đã lọc, và tín hiệu sau khi làm mượt.
  - Có thể nghe thử âm thanh sau xử lý nhờ `play_audio`.

Ứng dụng với file âm thanh thực tế

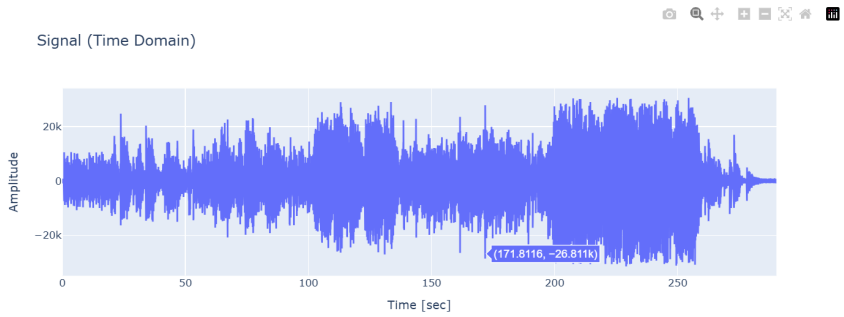
Trong thực tế, do nhiễu phân bố không đều và thay đổi theo thời gian, tín hiệu được chia nhỏ thành các đoạn ngắn (ví dụ 5 đến 10 giây) để xử lý riêng biệt (xem hình 13). Với mỗi đoạn, việc xác định tham số lọc (dải tần cần loại bỏ và ngưỡng biên độ) được thực hiện thông qua kết hợp giữa quan sát phổ tần số và nghe trực tiếp âm thanh.



Hình 13: Biểu đồ top 5 tần số xuất hiện nhiều nhất trong một khoảng thời gian

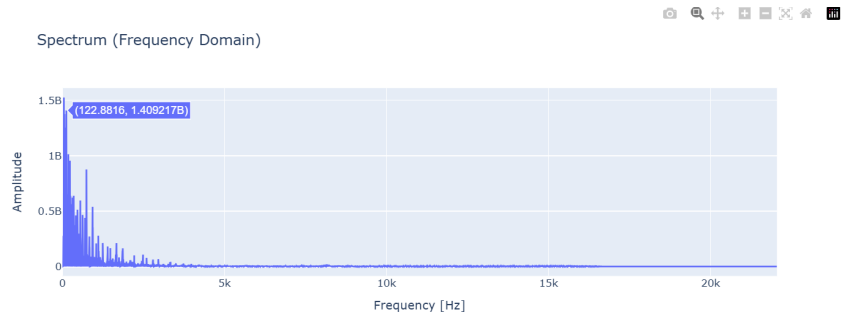
Cụ thể thì quá trình lọc một đoạn tín hiệu diễn ra như sau:

1. **Ngưỡng hóa biên độ:** các giá trị nghi ngờ là nhiễu được cắt lại nhằm loại bỏ các đỉnh bất thường trong tín hiệu. Tham khảo hình 14 để quan sát tín hiệu trong miền thời gian.



Hình 14: Dạng sóng tín hiệu trong miền thời gian

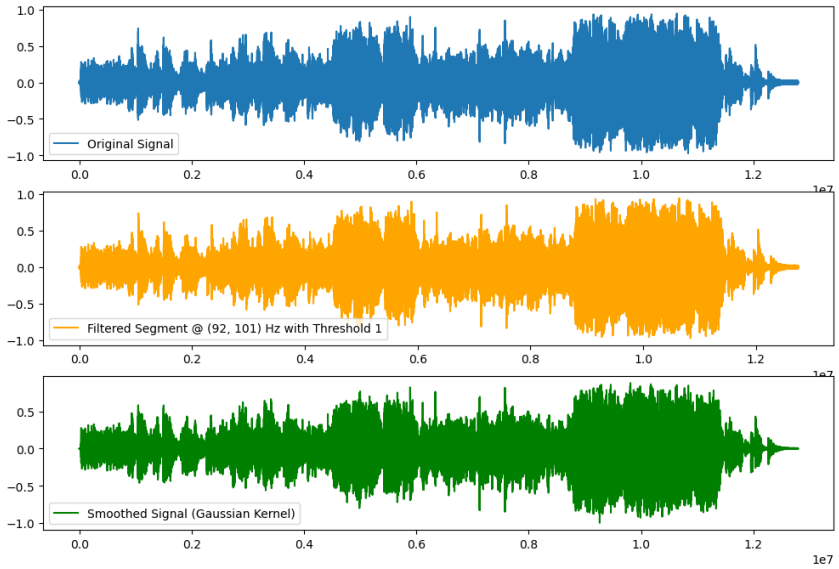
2. **Lọc chặn dải (band-stop):** loại bỏ nhiễu trong khoảng tần số 92–101 Hz. Việc xác định dải tần cần loại bỏ dựa trên biểu đồ tần số (hình 13) kết hợp với phân tích phổ (hình 15).



Hình 15: Tín hiệu trong miền tần số

3. **Làm mượt Gaussian:** được áp dụng trên toàn bộ tín hiệu để giảm dao động nhỏ, tăng tính ổn định. Hàm `fftconvolve` được sử dụng để thực hiện tích chập nhanh, giúp rút ngắn thời gian xử lý so với phương pháp truyền thống.

Kết quả: Sau khi lọc, tín hiệu nghe rõ hơn, ít nhiễu hơn cũng như thời gian xử lý nhanh nhờ sử dụng `fftconvolve`. Hình 16 minh họa sự khác biệt giữa tín hiệu trước và sau lọc.



Hình 16: So sánh tín hiệu trước và sau khi sử dụng bộ lọc kết hợp

Hạn chế: Phương pháp yêu cầu phải điều chỉnh thủ công (ngưỡng, dải tần) theo từng đoạn tín hiệu, nên khó áp dụng tự động hoặc hàng loạt trong các hệ thống thời gian thực (real time).

5.2 Phương pháp trừ phổ (SPECTRAL SUBTRACTION)

Tổng quan về phương pháp trừ phổ:

Phương pháp Trừ phổ là một kỹ thuật khử nhiễu trong miền tần số dựa trên giả định cơ bản rằng tín hiệu có nhiễu $X(f)$ có thể được biểu diễn dưới dạng tổng của tín hiệu sạch $S(f)$ và nhiễu $N(f)$. Trong miền thời gian, mối quan hệ này là:

$$x(t) = s(t) + n(t) \tag{5.2a}$$

Khi biến đổi sang miền tần số thông qua phép biến đổi Fourier, mối quan hệ này trở thành:

$$X(f) = S(f) + N(f) \tag{5.2b}$$

Trong đó $X(f)$, $S(f)$, $N(f)$ là các phổ phức của tín hiệu có nhiễu, tín hiệu sạch và nhiễu tương ứng.

(5.2a), (5.2b) Trích từ Steven F. Boll, "Suppression of Acoustic Noise in Speech Using Spectral Subtraction", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 2, pp. 114, April 1979.

Các bước triển khai thuật toán:

Thuật toán khử nhiễu được triển khai qua các bước tuần tự và chi tiết sau:

1. **Chuẩn hóa tín hiệu đầu vào:**

   Tín hiệu âm thanh đầu vào, được biểu diễn bởi một vector các mẫu thời gian signal_vector, được chuẩn hóa biên độ về khoảng $[-1, 1]$ với mục đích:

   - Ngăn ngừa tràn số (Clipping), đảm bảo rằng biên độ của tín hiệu không vượt quá giới hạn dải động mà hệ thống có thể xử lý, từ đó tránh gây ra hiện tượng méo tín hiệu.
   - Duy trì dải động và độ chính xác tính toán, giúp duy trì tính toàn vẹn của dữ liệu số.

   Quá trình chuẩn hóa được thực hiện bằng cách kiểm tra nếu max(|signal_vector|) > 1 thì ta thực hiện chia toàn bộ signal_vector cho max(|signal_vector|)

2. **Phân khung và cửa sổ hóa tín hiệu:**

   Tín hiệu âm thanh đã chuẩn hóa được chia thành các **khung (frames)** nhỏ hơn để phân tích hiệu quả hơn trong miền tần số.

   Các thông số phân khung quan trọng được sử dụng bao gồm:

   - **Kích thước khung (N_{frame}):** 2048 mẫu, lựa chọn kích thước cung cấp độ phân giải tần số đủ tốt mà vẫn đảm bảo tính dừng cục bộ của tín hiệu.
   - **Kích thước bước nhảy (N_{hop}):** 512 mẫu. Tương ứng với tỷ lệ chồng lấn 75% giữa các khung ($N_{\text{frame}} - N_{\text{hop}} = 2048 - 512 = 1536$ mẫu chồng lấn)

   Mỗi khung sau khi trích xuất ký hiệu là $x_k[n]$ (khung thứ k), được nhân với một hàm cửa sổ Hanning (Hann window), ký hiệu là $w[n]$. Hàm cửa sổ Hanning có dạng toán học:

$$w(n) = \begin{cases} \frac{1}{2}\left(1 - \cos\left(\frac{2\pi n}{N_{\text{frame}}-1}\right)\right), & 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases} \tag{5.2c}$$

   Việc tạo ra khung đã cửa sổ hóa $x_k^w[n] = x_k[n] \cdot w[n]$ là bước cần thiết trước khi thực hiện FFT nhằm *giảm hiện tượng rò rỉ phổ (spectral leakage)* khi một đoạn tín hiệu được cắt đột ngột. Cửa sổ Hanning với đặc tính dạng hình chuông và giảm dần về 0 ở hai biên, giúp làm mịn sự chuyển tiếp và giảm đáng kể hiện tượng rò rỉ phổ, từ đó cải thiện độ chính xác của phân tích tần số.

(5.2c) J. G. Proakis and D. G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 4th ed., Pearson New International Edition, 2006.

3. **Biến đổi Fourier Nhanh (FFT):**
   Sau khi phân khung và cửa sổ hóa, phép biến đổi Fourier nhanh (FFT) được áp dụng cho từng khung đã được cửa sổ hóa $x_k^w[n]$. Mỗi khung tín hiệu thời gian thực sẽ được biến đổi thành một phổ phức $X_k(m)$, trong đó m là chỉ số tần số:

$$X_k(m) = \sum_{n=0}^{N_{\text{frame}}-1} x_k^w[n] \cdot e^{-j\frac{2\pi mn}{N_{\text{frame}}}}$$

   Phổ phức $X_k(m)$ chứa cả thông tin về *biên độ* ($|X_k(m)|$) và *pha* ($\arg(X_k(m))$) của các thành phần tần số trong khung.

   - **Biên độ:** Đại diện cho cường độ (mức năng lượng) của từng tần số cụ thể.
   - **Pha:** Đại diện cho vị trí tương đối của sóng hình sin ở tần số đó tại điểm khởi đầu của khung.

   Kết quả của bước này là một tập hợp các phổ phức $\{X_k(m)\}_{k=0}^{N_{\text{frames}}-1}$

4. **Áp dụng khử nhiễu bằng Phương Pháp Trừ Phổ (Spectral Subtraction):**

   Thực hiện trên từng khung phổ phức $X_k(m)$.

   Bước 1: Ước lượng phổ biên độ nhiễu:

   Trong triển khai bài toán, nhiễu có tính xuyên suốt và chiếm ưu thế do đó phổ biên độ trung bình của toàn bộ tín hiệu được sử dụng làm ước lượng cho phổ biên độ nhiễu, ký hiệu là $|\hat{N}(m)|$ với:

$$\hat{N}(m) = \frac{1}{N_{\text{frames}}} \sum_{k=0}^{N_{\text{frames}}-1} |X_k(m)| \tag{5.2d}$$

   Bước 2: Tính toán ngưỡng nhiễu

   Ngưỡng này được xác định dựa trên giá trị trung bình (mean_mag) và độ lệch chuẩn (std_mag) của phổ biên độ ước lượng nhiễu trên các tần số dương do tính đối xứng của FFT cho tín hiệu thực. Với độ lệch chuẩn:

$$\sigma_N(m) = \sqrt{\frac{1}{N} \sum_{k=0}^{N-1} \left(|X_k(m)| - \hat{N}(m)\right)^2}$$

(5.2d) Trích từ Steven F. Boll, "Suppression of Acoustic Noise in Speech Using Spectral Subtraction", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 2, pp. 114, April 1979.

Công thức tính ngưỡng nhiễu là:

$$\text{noise\_threshold} = \hat{N}(m) + \text{spectral\_sub\_factor} \times \sigma_N(m) \tag{5.2e}$$

Trong đó, spectral_sub_factor là một hệ số điều chỉnh mức độ loại bỏ nhiễu.

- Giá trị này càng cao sẽ dẫn đến việc loại bỏ nhiễu mạnh mẽ hơn, làm cho tín hiệu sạch hơn nhưng có nguy cơ cao tạo ra *"tiếng ồn nhạc"(musical noise)* hoặc làm suy giảm chất lượng tín hiệu gốc.
- Và ngược lại giá trị càng thấp sẽ loại bỏ nhiễu một cách nhẹ nhàng hơn, giữ lại một phần nhiễu nhưng đồng thời giảm thiểu khả năng phát sinh các hiện tượng không mong muốn.

Bước 3: Trừ phổ và sàn hóa (Soft Thresholding)

Đối với mỗi phổ phức $X_k(m)$ của khung thứ k:

- **Tách Biên độ và Pha:** Biên độ $|X_k(m)|$ và pha $\arg(X_k(m))$ của phổ phức được tách riêng.
- **Trừ Biên độ:** Biên độ đã lọc $|\hat{S}_k(m)|$ được tính bằng cách trừ đi noise_threshold từ biên độ gốc.
- **Sàn Hóa (Flooring):** Một ngưỡng sàn bằng 0 được áp dụng bằng hàm $\max(\cdot, 0)$, đảm bảo rằng biên độ sau khi trừ không là giá trị âm, tránh tạo ra năng lượng âm ảo ở các tần số mà tín hiệu thực sự yếu hơn nhiễu, giúp giảm đáng kể hiện tượng "musical noise"bằng cách ngăn chặn sự dao động của phổ ở các vùng năng lượng thấp. Công thức soft thresholding là:

$$\hat{S}_k(m)| = \max(|X_k(m)| - \text{noise\_threshold}, 0) \tag{5.2f}$$

- **Tái tạo phổ phức sạch:** Phổ phức đã lọc $\hat{S}_k(m)$ được tái tạo bằng cách kết hợp biên độ đã lọc với pha gốc:

$$\hat{S}_k(m) = |\hat{S}_k(m)| \cdot e^{j \cdot \arg(X_k(m))} \tag{5.2g}$$

(5.2e), (5.2f), (5.2g) Trích từ M. Karam, H. F. Khazaal, H. Aglan, and C. Cole, "Noise removal in speech processing using spectral subtraction", *Journal of Signal and Information Processing*, vol. 5, no. 2, pp. 6 , May 2014.

Kết quả của giai đoạn này là một tập hợp các phổ phức đã được khử nhiễu $\{\hat{S}_k(m)\}_{k=0}^{N_{\text{frames}}-1}$.

5. **Biến đổi Fourier Nhanh Ngược (IFFT)**

   Các phổ đã khử nhiễu $\hat{S}_k(m)$ sau đó được chuyển đổi ngược lại miền thời gian bằng phép biến đổi Fourier nhanh ngược (IFFT) để thu được khung tín hiệu sạch ước lượng trong miền thời gian $\hat{s}_k[n]$:

$$\hat{s}_k[n] = \frac{1}{N_{\text{frame}}} \sum_{m=0}^{N_{\text{frame}}-1} \hat{S}_k(m) \cdot e^{j\frac{2\pi mn}{N_{\text{frame}}}}$$

   Vì tín hiệu gốc là tín hiệu thực, phần ảo của kết quả IFFT thường rất nhỏ do lỗi tính toán số học hoặc ảnh hưởng của việc xử lý nên bị loại bỏ, chỉ giữ lại phần thực để đảm bảo tính hợp lệ của tín hiệu âm thanh. Kết quả thu được là các khung tín hiệu đã được khử nhiễu trong miền thời gian $\{\hat{s}_k[n]\}_{k=0}^{N_{\text{frames}}-1}$.

6. **Tái tạo tín hiệu âm thanh bằng Overlap-Add**

   Cuối cùng, các khung tín hiệu đã khử nhiễu $\hat{s}_k[n]$ được tổng hợp lại để tái tạo tín hiệu âm thanh hoàn chỉnh bằng phương pháp Overlap-Add (chồng lấn và cộng), là đảo ngược của quá trình phân khung và cửa sổ hóa, đảm bảo tái tạo liền mạch và chính xác tín hiệu ban đầu.

   Tín hiệu khử nhiễu toàn bộ $s_{\text{denoised}}[p]$ được xây dựng bằng cách cộng các khung chồng lấn sau khi nhân chúng với bình phương của cửa sổ Hanning:

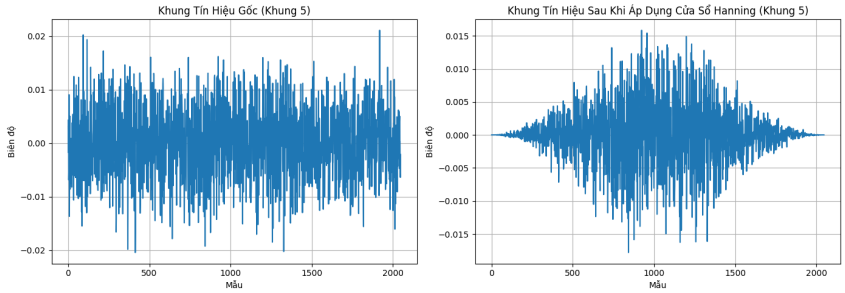
$$s_{\text{denoised}}[p] = \frac{\sum_k \hat{s}_k[p - k \cdot N_{\text{hop}}] \cdot w^2[p - k \cdot N_{\text{hop}}]}{\sum_k w^2[p - k \cdot N_{\text{hop}}]} \tag{5.2h}$$

   Trong đó, p là chỉ số mẫu trong tín hiệu toàn bộ, và $w^2[n]$ là bình phương của hàm cửa sổ Hanning.

   Sau khi tái tạo, tín hiệu tổng hợp s_{denoised} được chuẩn hóa lại biên độ về khoảng $[-1, 1]$ một lần nữa, đảm bảo tín hiệu đầu ra có mức biên độ an toàn và tiêu chuẩn, ngăn ngừa hiện tượng tràn hoặc méo tín hiệu khi lưu trữ hoặc phát lại.

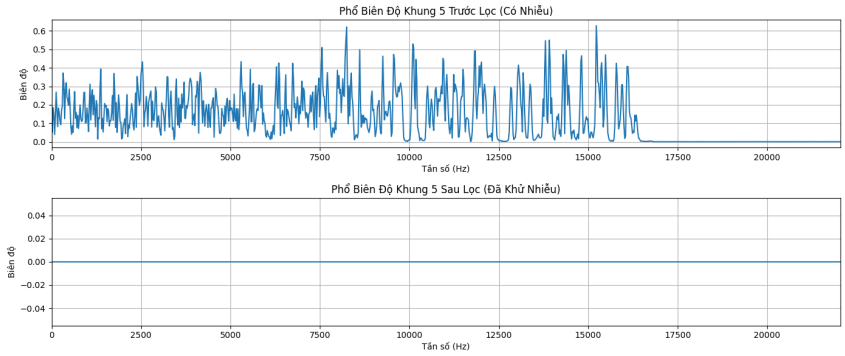
Kết quả

(5.2h) Trích từ Julius O. Smith III,"Lecture 8A – FFT Signal Processing: The Overlap-Add (OLA) Method for Fourier Analysis, Modification, and Resynthesis", MUS421: Signal Processing for Music Applications, Center for Computer Research in Music and Acoustics (CCRMA), Department of Music, Stanford University, Stanford, California, USA, June 27, 2020.



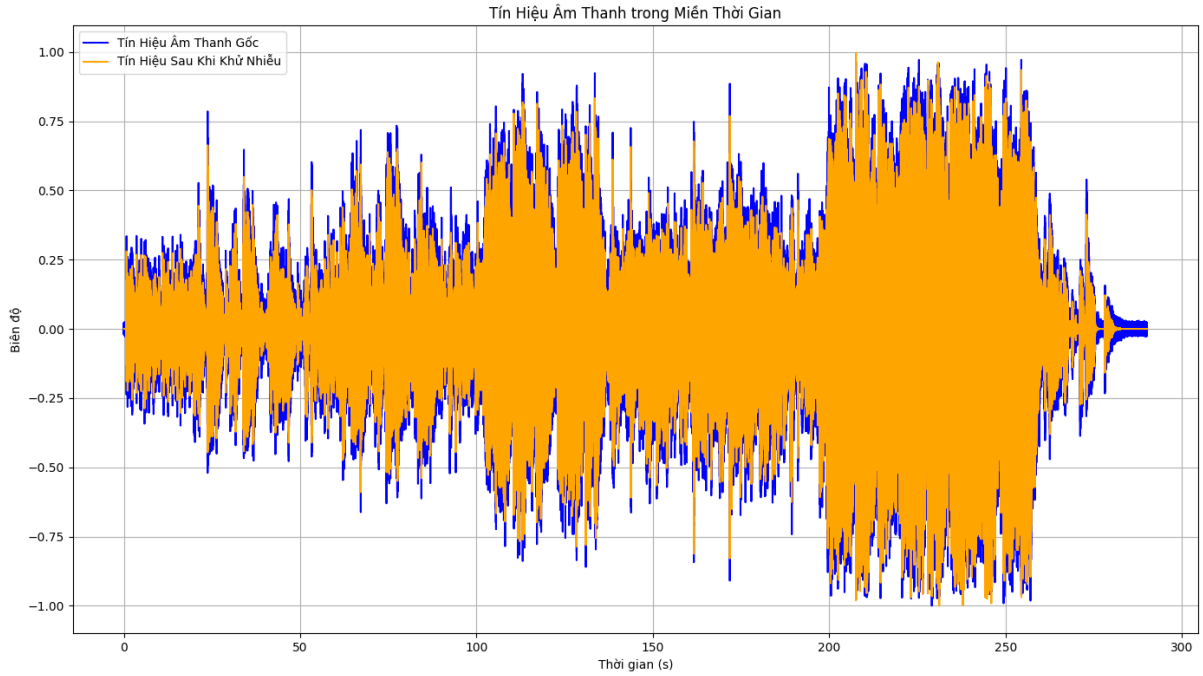
Hình 17: Khung tín hiệu gốc và khung tín hiệu sau khi áp dụng cửa sổ Hanning.

Biểu đồ(Hình 17) cho thấy tác động của cửa sổ hóa lên biên độ của một khung tín hiệu, sự giảm dần biên độ ở hai đầu khung, giúp giảm hiện tượng rò rỉ phổ trong miền tần số.



Hình 18: Phổ biên độ trước và sau lọc.

Bằng cách so sánh phổ biên độ của cùng một khung trước và sau khi áp dụng thuật toán khử nhiễu, biểu đồ (Hình 18) đã mô tả sự giảm đáng kể của năng lượng ở các dải tần số nơi nhiễu chiếm ưu thế.



Hình 19: Tín hiệu âm thanh gốc và tín hiệu sau khi khử nhiễu trong miền thời gian.

So sánh dạng sóng của tín hiệu gốc (có nhiễu) với tín hiệu đã được xử lý (đã khử nhiễu) trong miền thời gian (Hình 19), trực quan hóa sự "sạch"hơn và mượt mà hơn của tín hiệu sau khi thực hiện qua thuật toán.

Kết luận:
Thuật toán khử nhiễu tín hiệu âm thanh bằng phương pháp trừ phổ là một kỹ thuật hiệu quả và tương đối đơn giản để cải thiện chất lượng âm thanh. Bằng cách áp dụng một chuỗi các bước xử lý tín hiệu bao gồm chuẩn hóa tín hiệu, phân khung, biến đổi Fourier, ước lượng và trừ nhiễu trong miền tần số, và cuối cùng là tái tạo tín hiệu bằng Overlap-Add, chúng ta có thể giảm thiểu đáng kể nhiễu nền.

Thách thức chính của phương pháp này nằm ở việc ước lượng chính xác phổ nhiễu $|\hat{N}(f)|$. Nếu ước lượng nhiễu không chuẩn xác hoặc nhiễu có tính chất biến đổi nhanh chóng, có thể dẫn đến các hiện vật âm thanh không mong muốn, phổ biến nhất là *"tiếng ồn nhạc"(musical noise)* là các âm thanh nghe như tiếng lách tách, vo ve hoặc tiếng vang do việc loại bỏ nhiễu không đồng đều trên phổ.

5.3 Phương pháp khử nhiễu bằng ngưỡng mật độ phổ công suất (PSD Thresholding) trực tiếp

Mô tả:

Ý tưởng chính là loại bỏ các thành phần tần số có năng lượng (công suất) dưới một ngưỡng xác định, xác định rằng những thành phần này chủ yếu là nhiễu. Áp dụng trong các trường hợp khi nhiễu có phổ công suất thấp và phân bố rộng, trong khi tín hiệu mong muốn có năng lượng tập trung ở các tần số nhất định và cao hơn ngưỡng nhiễu.

Công thức của mật độ phổ công suất $S_{xx}(f)$ của một tín hiệu $x(t)$ có thể được ước lượng từ biến đổi Fourier của tín hiệu, $X(f)$:

$$S_{xx}(f) = \frac{|X(f)|^2}{N} \tag{5.3a}$$

Trong đó N là số lượng mẫu của tín hiệu. Bằng cách so sánh $S_{xx}(f)$ với một ngưỡng λ, các thành phần tần số được giữ lại nếu $S_{xx}(f) \geq \lambda$, và được đặt về 0 nếu $S_{xx}(f) < \lambda$.

Các bước triển khai:

Việc triển khai phương pháp PSD Thresholding trực tiếp bao gồm các bước sau:

- **Bước 1: Biến đổi Fourier Nhanh (FFT) của toàn bộ tín hiệu**
  Khác với phương pháp Trừ Phổ, bước này không yêu cầu phân khung hay cửa sổ hóa. Phép biến đổi Fourier nhanh (FFT) được áp dụng trực tiếp lên toàn bộ tín hiệu có nhiễu $x[n]$ để thu được phổ phức $X[k]$:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\frac{2\pi nk}{N}}$$

- **Bước 2: Tính mật độ phổ công suất (PSD)**
  Mật độ phổ công suất (PSD) $P(k)$ được tính từ phổ phức $X[k]$ của toàn bộ tín hiệu. Là bình phương của biên độ phổ, chuẩn hóa theo kích thước tín hiệu:

$$P(k) = \frac{|X[k]|^2}{N}$$

  Với $P(k)$ là giá trị thực sau phép nhân với liên hợp phức.

- **Bước 3: Xác định ngưỡng cố định**
  Một ngưỡng năng lượng cố định λ_{PSD} được định nghĩa trước. Giá trị này được xác định thông qua thử nghiệm hoặc dựa trên đặc tính của nhiễu.

(5.3a) Trích từ National Semiconductor. (1980). *Application Note 255: Power Spectrum Estimation* (Section 6.0, p. 7). Texas Instruments.

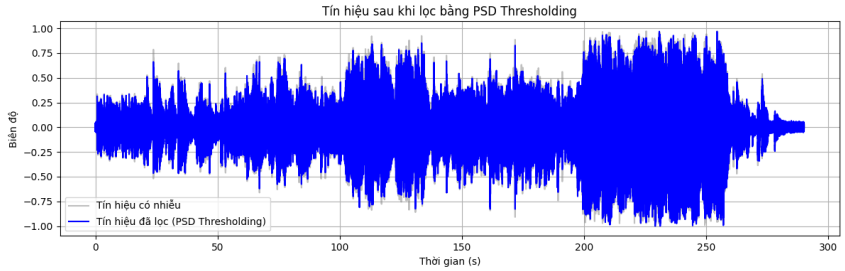
- **Bước 4: Áp dụng ngưỡng lên PSD và phổ phức**
  Ccác thành phần tần số trong phổ phức $X[k]$ có năng lượng thấp hơn ngưỡng λ_{PSD} được coi là nhiễu và bị loại bỏ bằng cách đặt giá trị PSD của chúng về 0.

- **Bước 5: Biến đổi Fourier Nhanh Ngược (IFFT)**
  Phổ phức đã lọc $\hat{X}(k)$ sau đó được biến đổi ngược về miền thời gian bằng phép biến đổi Fourier nhanh ngược (IFFT) để tái tạo tín hiệu sạch $\hat{x}[n]$:

$$\hat{x}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \hat{X}[k] \cdot e^{j\frac{2\pi nk}{N}}$$

- **Bước 6: Chuẩn hóa tín hiệu đã lọc**
  Tín hiệu cuối cùng đã được khử nhiễu $\hat{x}[n]$ được chuẩn hóa biên độ về khoảng $[-1, 1]$ để đảm bảo mức biên độ phù hợp và an toàn cho việc phát lại hoặc xử lý. Với biên độ tuyệt đối lớn nhất của $\hat{x}[n]$ là $\max(|\hat{x}|)$, thì tín hiệu chuẩn hóa $\hat{x}_{\text{norm}}[n]$ là $\hat{x}_{\text{norm}}[n] = \frac{\hat{x}[n]}{\max(|\hat{x}|)}$ nếu $\max(|\hat{x}|) > 1$.

Kết quả:



Hình 20: So sánh tín hiệu gốc và tín hiệu đã lọc bằng phương pháp PSD Thresholding.

Biểu đồ (Hình 20) cho thấy các thành phần nhiễu có năng lượng thấp đã được loại bỏ. Phương pháp này đơn giản và hiệu quả khi xử lý nhiễu nền ổn định. Bên cạnh đó, việc lựa chọn ngưỡng phù hợp là thách thức. Nếu quá cao, có thể làm mất tín hiệu; nếu quá thấp, sẽ không lọc được nhiễu.

Kết luận

Phương pháp PSD Thresholding trực tiếp là một giải pháp đơn giản và nhanh chóng để khử nhiễu tín hiệu âm thanh. Mặc dù không mạnh như phương pháp Trừ Phổ trong môi trường có nhiễu phức tạp, nó vẫn là một lựa chọn tốt khi nhiễu nền ổn định.

Tuy nhiên, việc lựa chọn ngưỡng phù hợp đòi hỏi sự tinh chỉnh cẩn thận theo từng trường hợp cụ thể để đạt được kết quả tốt nhất và có thể không linh hoạt bằng các phương pháp dựa trên phân tích khung tín hiệu.

5.4 Phương pháp lọc nhiễu tín hiệu dựa trên ngưỡng biên độ

Giới thiệu

Đây là một kỹ thuật giảm nhiễu cơ bản, hoạt động bằng cách loại bỏ các thành phần tần số không mong muốn dựa trên biên độ của chúng. Bằng cách thiết lập một ngưỡng phù hợp trong miền tần số, phương pháp này có thể lọc bỏ nhiễu và cải thiện chất lượng tín hiệu một cách hiệu quả và nhanh chóng.

Cơ sở của phương pháp

Khi một tín hiệu từ miền thời gian được chuyển đổi sang miền tần số (thông qua Biến đổi Fourier), ta thu được một phổ tần số. Trong phổ này:

- Biên độ (Amplitude) của một thành phần tần số cụ thể cho biết cường độ hoặc năng lượng (độ mạnh hay to) của tần số đó trong tín hiệu gốc.
- Các tần số có biên độ lớn là những thành phần nổi bật, chiếm ưu thế trong tín hiệu.
- Các tần số có biên độ nhỏ thường đại diện cho các thành phần yếu hơn hoặc nhiễu.

Giả định **biên độ thấp là nhiễu** dựa trên một nguyên lý cơ bản được chấp nhận rộng rãi trong xử lý tín hiệu:

- Tín hiệu hữu ích (như giọng nói, âm nhạc) thường tập trung năng lượng vào các dải tần số cụ thể, tạo ra các đỉnh có biên độ cao trên phổ tần số.
- Ngược lại, nhiễu ngẫu nhiên (ví dụ: nhiễu trắng) có xu hướng phân bố năng lượng tương đối đồng đều trên toàn bộ phổ tần số, nhưng ở mức biên độ thấp và không có đỉnh rõ rệt.

Do đó, việc đặt một ngưỡng biên độ và loại bỏ tất cả các thành phần tần số nằm dưới ngưỡng đó là một cách hiệu quả để loại bỏ phần lớn nhiễu mà vẫn giữ được các thành phần quan trọng của tín hiệu.

Các bước thực hiện

Bước 1: Chuyển đổi tín hiệu sang miền tần số

Áp dụng Biến đổi Fourier Nhanh (FFT) lên tín hiệu gốc để phân tích và thu được phổ tần số của nó. Phổ này bao gồm thông tin về biên độ và pha của từng thành phần tần số.

Khi thực hiện FFT, ta sẽ nhận được các số phức. Mỗi số phức Z_f tại tần số f có dạng $Z_f = Re_f + jIm_f$, trong đó Re_f là phần thực và Im_f là phần ảo.

- Biên độ của thành phần tần số f được tính là: $A_f = |Z_f| = \sqrt{Re_f^2 + Im_f^2}$
- Pha của thành phần tần số f được tính là: $\phi_f = \arctan_2(Im_f, Re_f)$

Bước 2: Lọc phổ tần số theo ngưỡng

Đây là bước cốt lõi của phương pháp. Ta sẽ loại bỏ các thành phần tần số được cho là nhiễu bằng cách so sánh biên độ của chúng với một ngưỡng $A_{threshold}$.

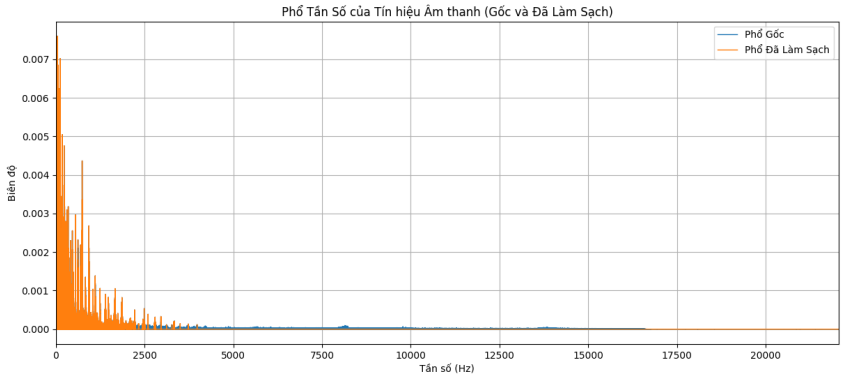
- **Xác định ngưỡng lọc ($A_{threshold}$):** Có hai cách tiếp cận phổ biến:
  - Ngưỡng tương đối: Thiết lập ngưỡng dựa trên biên độ lớn nhất (A_{max}) trong phổ.

    $$A_{threshold} = \alpha \times A_{max}$$

    Trong đó, α là một hệ số tỷ lệ nhỏ (ví dụ: 0.01 đến 0.05).
  - Ngưỡng dựa trên mẫu nhiễu: Phân tích một đoạn tín hiệu chỉ chứa nhiễu (nếu có). Biên độ trung bình hoặc biên độ cực đại của phổ nhiễu này có thể được dùng để ước lượng ngưỡng lọc cho toàn bộ tín hiệu. Đây là cách tiếp cận cho kết quả chính xác hơn.
- **Áp dụng ngưỡng (Điều chỉnh cho Abs và ngưỡng bằng 0):** Duyệt qua toàn bộ phổ tần số. Với mỗi thành phần tần số f có biên độ A_f (đã tính bằng giá trị tuyệt đối như trên):
  - Nếu $A_f \geq A_{threshold}$, giữ nguyên thành phần này.
  - Nếu $A_f < A_{threshold}$, đặt biên độ của thành phần này về 0.

    $$Z_f = 0 + j0$$

Kết quả là một phổ tần số đã được "làm sạch", như minh hoạ ở Hình 21.

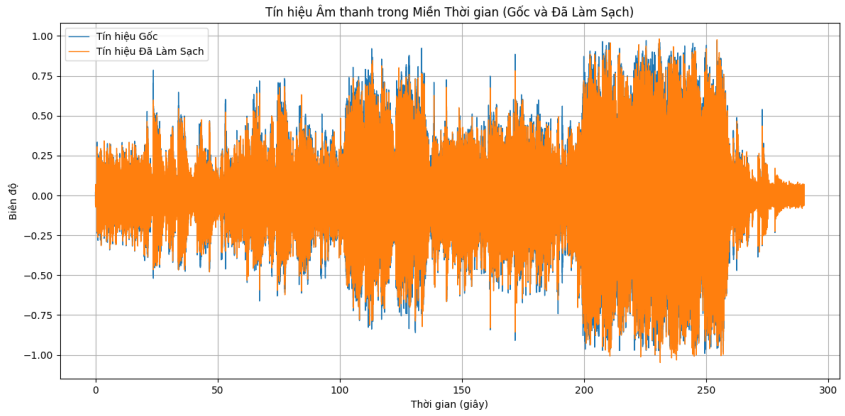


Hình 21: Phổ Tần Số của Tín hiệu Âm thanh (Gốc và Đã Làm Sạch)

Bước 3: Tái tạo tín hiệu về miền thời gian

Áp dụng Biến đổi Fourier Ngược (IFFT) lên phổ tần số đã được lọc để chuyển đổi nó trở lại tín hiệu trong miền thời gian. Tín hiệu thu được chính là phiên bản đã được giảm nhiễu của tín hiệu gốc.

Kết quả: Tín hiệu gốc đã được lọc nhiễu bằng cách loại bỏ các thành phần tần số yếu (Hình 22).



Hình 22: Tín hiệu Âm thanh trong Miền Thời gian (Gốc và Đã Làm Sạch)

So sánh ngưỡng trên biên độ và ngưỡng trên mật độ phổ công suất (PSD)

Việc đặt ngưỡng để lọc nhiễu có thể thực hiện trên biên độ (A_f) hoặc mật độ phổ công suất (PSD - $P_f = A_f^2$) của tín hiệu trong miền tần số. Sự lựa chọn này ảnh hưởng đến độ nhạy và hiệu quả lọc.

- **Ngưỡng trên biên độ (A_f)**:
  - **Ưu điểm**: Trực quan, dễ hiểu, dễ triển khai. Phù hợp cho các ứng dụng cơ bản, cần tốc độ.
  - **Nhược điểm**: Ít nhạy hơn với các nhiễu rất nhỏ, có thể yêu cầu tinh chỉnh ngưỡng cẩn thận để tránh loại bỏ nhầm tín hiệu yếu.
- **Ngưỡng trên mật độ phổ công suất (PSD - $P_f = A_f^2$)**:
  - **Ưu điểm**: PSD khuếch đại sự khác biệt giữa các thành phần mạnh và yếu. Do đó, các thành phần nhiễu có biên độ nhỏ sẽ có giá trị PSD rất nhỏ, giúp phân biệt rõ ràng hơn và lọc nhiễu hiệu quả hơn, đặc biệt với nhiễu trắng.
  - **Nhược điểm**: Yêu cầu thêm bước tính toán bình phương. Có thể làm mất chi tiết nếu ngưỡng quá chặt do tính phi tuyến.

Kết luận: Dùng ngưỡng trên PSD thường mang lại hiệu quả cao hơn trong việc loại bỏ nhiễu có năng lượng thấp như nhiễu trắng, bởi nó giúp làm nổi bật sự khác biệt giữa tín hiệu và nhiễu. Tuy nhiên, ngưỡng trên biên độ vẫn là một lựa chọn tốt cho các ứng dụng cần sự đơn giản và tốc độ.

Ưu và nhược điểm

Ưu điểm

- **Đơn giản và nhanh chóng**: Thuật toán rất dễ triển khai và có tốc độ xử lý nhanh, phù hợp cho các ứng dụng thời gian thực.
- **Hiệu quả với nhiễu biên độ thấp**: Hoạt động tốt trong việc loại bỏ các loại nhiễu nền có biên độ thấp và phân bố rộng.

Nhược điểm

- **Khó khăn khi tín hiệu và nhiễu chồng lấn hoặc ngưỡng không phù hợp**: Đây là một hạn chế lớn. Nếu nhiễu có tần số tương tự hoặc nằm chồng lấn với các thành phần quan trọng của tín hiệu gốc, việc đặt ngưỡng biên độ trở nên cực kỳ khó khăn. Đặt ngưỡng quá cao có thể loại bỏ nhầm cả những phần tín hiệu hữu ích, gây ra suy hao, méo tiếng hoặc mất chi tiết. Ngược lại, đặt ngưỡng quá thấp sẽ không loại bỏ được nhiễu hiệu quả.
- **Hiện tượng rò rỉ phổ (Spectral Leakage)**: Việc áp dụng FFT trên toàn bộ tín hiệu mà không qua bước xử lý bằng hàm cửa sổ (windowing) có thể gây ra hiện tượng rò rỉ năng lượng từ các tần số chính sang các tần số lân cận, làm giảm độ chính xác của phổ và ảnh hưởng đến chất lượng lọc.

Kết luận:

Phương pháp lọc nhiễu dựa trên ngưỡng biên độ là một kỹ thuật nền tảng, mạnh mẽ nhờ sự đơn giản và tốc độ. Dù có những hạn chế nhất định như nguy cơ làm suy yếu tín hiệu và hiện tượng rò rỉ phổ, nó vẫn là một lựa chọn hiệu quả cho nhiều bài toán cơ bản. Để cải thiện chất lượng, có thể kết hợp phương pháp này với các kỹ thuật tiên tiến hơn như sử dụng hàm cửa sổ hoặc các thuật toán xác định ngưỡng thích ứng.

Bảng so sánh ưu nhược điểm của các phương pháp

Phương pháp	**Ưu điểm**	**Nhược điểm**
Phương pháp lọc kết hợp ngưỡng hóa, band-stop, Gaussian (FFT)	- Gaussian filter giúp duy trì tính mượt và hạn chế méo tín hiệu. - FFT và convolution giúp tăng tốc độ xử lý với tín hiệu dài. - Có thể điều các tham số để phù hợp từng loại nhiễu.	- Cần tinh chỉnh nhiều tham số (ngưỡng, tần số chặn, Gaussian...). - Lọc thủ công nên tùy thuộc vào người lọc mà chất lượng âm thanh khác nhau.
Phương pháp trừ phổ	- Thuật toán cơ bản, dễ hiểu và áp dụng. - Loại bỏ tốt các loại nhiễu không thay đổi theo thời gian. - Có thể kết hợp với các kỹ thuật khác (windowing, smoothing). - Xử lý phổ nhanh trong miền tần số bằng FFT.	- Nếu ước lượng phổ nhiễu sai lọc sai. - Có thể gây âm thanh lạ. - Gây méo hoặc mất dữ liệu.
Phương pháp khử nhiễu bằng ngưỡng mật độ phổ công suất	- Tốt khi nhiễu có mức công suất ổn định. - Dễ phát hiện dải nhiễu nhờ phân tích mật độ phổ. - Linh hoạt trong thiết kế ngưỡng theo phổ.	- Ước lượng sai sẽ làm giảm hiệu quả. - PSD không phản ứng kịp với nhiễu thay đổi nhanh.
Phương pháp lọc nhiễu tín hiệu dựa trên ngưỡng biên độ	- Cài đặt dễ, không cần biến đổi tín hiệu. - Hiệu quả nếu tín hiệu mạnh hơn nhiễu.	- Tín hiệu có biên độ thấp dễ bị loại bỏ nhầm. - Dễ nhầm lẫn giữa tín hiệu và nhiễu. - Không thích hợp với tín hiệu phức tạp nhất (biên độ lớn). - Không sử dụng thông tin tần số nên bỏ qua nhiễu có tần số rõ ràng.

Bảng 4: So sánh các phương pháp lọc nhiễu tín hiệu

6 Tổng kết

Thông qua phần trình bày phía trên, chúng em trực tiếp xây dựng và triển khai thuật toán **DFT (Discrete Fourier Transform)** bằng code thủ công sau đó ứng dụng nó vào bài toán khử nhiễu âm thanh. Tuy nhiên, khi tiến hành thực nghiệm cho thấy DFT bộc lộ một số hạn chế, đặc biệt là hiệu suất tính toán trên các tín hiệu lớn và phức tạp. Để khắc phục vấn đề này, chúng em đã triển khai thuật toán **FFT (Fast Fourier Transform)** nhằm tối ưu hóa tốc độ xử lý một hạn chế rất lớn của DFT.

Trong quá trình thực hiện, chúng em không chỉ tự xây dựng các thuật toán DFT và FFT để hiểu rõ nguyên lý hoạt động, mà còn thử nghiệm nhiều phương pháp khác nhau trên dữ lệ thực tế là một bài hát bị nhiễu bởi tiếng ồn xung quanh. Một số phương pháp đã áp dụng: lọc kết hợp ngưỡng hóa, band-stop, Gaussian qua FFT convolve, trừ phổ, khử nhiễu bằng ngưỡng mật độ phổ công suất (PSD Thresholding) trực tiếp và lọc nhiễu tín hiệu dựa trên ngưỡng biên độ.Mặc dù các phương pháp đã mang lại kết quả khả quan, cải thiện chất lượng âm thanh trong việc lọc nhiễu nhưng vẫn còn một số nhược điểm như hiệu quả chưa cao với tín hiệu phức tạp hoặc nhiễu có đặc tính thay đổi theo thời gian. Đặc biệt, việc xác định chính xác tần số nhiễu trong môi trường thực vẫn là một bài toán khó và nan giải.

Với những vấn đề còn tồn tại, nhóm chúng em nhận thấy hướng phát triển tiếp theo có thể tập trung vào việc cải thiện các phương pháp đã xây dựng được theo hướng tự động hóa trong việc nhận diện và phân tích nhiễu, chẳng hạn như sử dụng các kỹ thuật học máy.

Thông qua đồ án này, chúng em đã hiểu sâu hơn về cách hoạt động của các thuật toán biến đổi Fourier rời rạc, biết cách áp dụng lý thuyết vào thực tế và xây dựng nền tảng để tiếp tục phát triển các hướng tiếp cận mới trong xử lý tín hiệu âm thanh.

7 Bảng đánh giá thành viên nhóm 4

MSSV	Họ và tên	Công việc thực hiện	Mức độ đóng góp
22280052	Phan Thị Ngọc Linh	Triển khai hàm DFT và IDFT, code mở rộng phương pháp, thuyết trình.	109%
22280003	Phạm Bá Hoàng Anh	Lý thuyết, làm slide thuyết trình.	94%
22280004	Trương Bình Ba	Ý tưởng khử nhiễu và thuật toán, code mở rộng phương pháp, DFT issues.	104%
22110110	Quách Vũ Luân	Code mở rộng phương pháp, thuyết trình.	93%
22280013	Phạm Lê Hồng Đức	Triển khai hàm FFT và IFFT, code mở rộng phương pháp, DFT issues.	102%
22280016	Bạch Ngọc Lê Duy	Ý tưởng khử nhiễu và thuật toán, làm slide thuyết trình.	98%
22280017	Ngô Thị Mỹ Duyên	Lý thuyết, làm slide thuyết trình.	100%

Tài liệu

[1] Isaac Amidror. *Mastering the Discrete Fourier Transform in One, Two or Several Dimensions: Pitfalls and Artifacts*. Springer, 2013.

[2] Steven F Boll. Suppression of acoustic noise in speech using spectral subtraction. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 27(2), 1979.

[3] Marc Karam, Hasan F. Khazaal, Heshmat Aglan, and Cliston Cole. Noise removal in speech processing using spectral subtraction. *Journal of Signal and Information Processing*, 5, 2014.

[4] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2000. chapter 5, section 8, pp. 356–382.

[5] National Semiconductor. Application note 255: Power spectrum estimation. Technical report, Texas Instruments, 1980. Section 6.0, p. 7.

[6] Gabriel Popescu. *Principles of Biophotonics, Volume 1: Linear Systems and the Fourier Transform in Optics*. IOP Publishing, 2020. chapter 1, p. 1.

[7] John G. Proakis and Dimitris K. Manolakis. *Digital Signal Processing: Principles, Algorithms, and Applications*. Pearson Education (New International Edition), 4 edition, 2006.