Fast Fourier Transform

Contents

Relationships among Fourier X

Fourier Series, Fourier Transform, Discrete Fourier Transform, Fast Fourier Transform…There seems to be many Fourier XXX, but what are the differences?

1. Fourier Series

   FS is to describe any integrable **periodic** function. It's a SERIES.

2. Fourier Transform
   FT is to describe any continuous integrable common function. It doesn't have to be periodic, or to say its periodic is infinity.
3. Discrete Fourier Transform
   DFT is to describe any discrete **array** with $O(n^2)$ time complexity.
4. Fast Fourier Transform
   $O(n^2)$ is not practical for ordinary use. FFT is suggested to speed up calculation with $O(n\log(n))$ time complexity.

Fourier Series

First of all, we put out a formula called Fourier Series without any proofing:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}\left(a_n \cos\frac{2\pi}{T}nx + b_n \sin\frac{2\pi}{T}nx\right)$$

where $a_n = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)\cos\frac{2\pi}{T}nx$, $b_k = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)\sin\frac{2\pi}{T}kx$, $n \in N, k \in N^*$[i],

T is the period of the function $f(x)$.

Using Eula's Formula:

$$\mathbb{e}^{jx} = \cos x + j\sin x$$
$$\cos x = \frac{\mathbb{e}^{jx} + \mathbb{e}^{-jx}}{2}$$
$$\sin x = \frac{\mathbb{e}^{jx} - \mathbb{e}^{-jx}}{2}$$

, where $j^2 = -1$.

one can get:

$$\begin{aligned} f(x) &= \frac{a_0}{2} + \sum_{n=1}\left[a_n\left(\frac{\mathbb{e}^{j\frac{2\pi}{T}nx} + \mathbb{e}^{-j\frac{2\pi}{T}nx}}{2}\right) + b_n\left(\frac{\mathbb{e}^{j\frac{2\pi}{T}nx} - \mathbb{e}^{-j\frac{2\pi}{T}nx}}{2}\right)\right] \\ &= \frac{a_0}{2} + \sum_{n=1}\left(\frac{a_n + b_n}{2}\mathbb{e}^{j\frac{2\pi}{T}nx} + \frac{a_n - b_n}{2}\mathbb{e}^{-j\frac{2\pi}{T}nx}\right) \end{aligned}$$

Let

$$c_0 = \frac{a_0}{2}$$
$$c_k = \frac{a_n + b_n}{2}, k > 0$$
$$c_l = \frac{a_n - b_n}{2}, l < 0$$

So,

$$f(x) = c_0 \mathbb{e}^{i\times 0x} + \sum_{n=1}\left(c_n \mathbb{e}^{j\frac{2\pi}{T}nx} + c_{-n}\mathbb{e}^{-j\frac{2\pi}{T}nx}\right)$$

$$= \sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{\frac{2\pi}{T}nxj} \qquad (1)$$

, where $c_n = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(x)\mathbb{e}^{-\frac{2\pi}{T}nxj}\mathbb{d}x$ [ii].

Now, Fourier Series has been extended to complex numbers.

Physical Meaning of Fourie Series

Now we'll have to discuss the physical meaning of Fourier Series. If you and your desk mate made an agreement that when playing phones in class, if the teacher is approaching you, and any of you has found it, they must play a sound containing both 60dB 3kHz and 90dB 6kHz signals to warn another one. As a bio-body, it's easy to estimate different frequency components, but if you record the sound signal into a computer, how can you analyze it through electronic devices?

We have learned two ways to (approximately) express functions: Power Series and Fourier Series. We want to analyze the frequency characteristic of the sound signal (a function), and we surprisingly find that Fourier Series consists of trigonometric functions that contains the information of frequency! And this seems to provide an approach to analyze the frequency components of signal data.

Or we can put out another example. We know the frequency of C major 1 is 440Hz, but the sound of C major 1 of guitar and piano are quite different. Then what is the exact difference? In junior high school, we know the three characteristics of sound are: **loudness, frequency, timbre**. So, the difference can be timbre only. Timbre actually refers to different combinations of different frequencies. For example, for C major 1, the main frequency is 440Hz with 60dB energy (loudness), but the second strong frequency will influence its timbre. So, we'll have to analyze the frequency components of the sound.

If a signal we captured can be expressed as:

$$f(t) = 3\sin(2\pi \times 4x) + 3\sin(2\pi \times 5x) + 6\sin(2\pi \times 8x)$$

we will know this signal consists of 4Hz of 3 magnitude, 5Hz of 3 magnitude, and 8Hz of 6 magnitude signals. If we draw its frequency in a graph, in which the abscissa means frequency, and the ordinate means magnitude, we can get Figure 1:

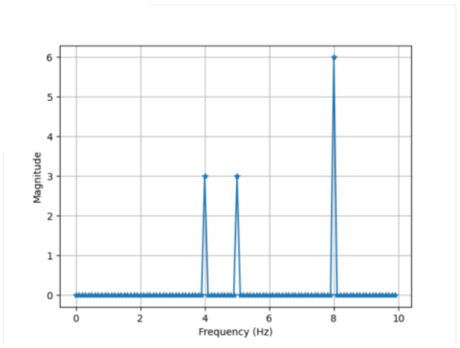

Figure 1 Demonstration of Frequency Space

After figuring out the physical meaning of Fourier Series, let's further induce Fourier Transform.

Induction of Fourier Transform

If the period T becomes very large ($T \to +\infty$), which means $f(t)$ is not a periodic function, then frequency $\nu = \frac{n}{T}$ and circular frequency $\omega = \frac{2\pi}{T} n$ tend to be continuous. In this case, n in $c_n = \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \mathbb{e}^{-\frac{2\pi}{T} nxj} \mathbb{d}x$ doesn't have clear physical meaning, but we know c_n is the "magnitude" of circular frequency $\omega = 2\pi \frac{n}{T}$. So, we can define, the "magnitude" of the circular frequency $\omega = \frac{2\pi n}{T}$ is

$$\begin{aligned} F'(\omega) = c_n &= \lim_{T\to+\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \mathbb{e}^{-\frac{2\pi}{T} nxj} \mathbb{d}x \\ &= \lim_{T\to+\infty} \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \mathbb{e}^{-\frac{2\pi}{T} nxj} \mathbb{d}x \\ &= \lim_{T\to+\infty} \frac{1}{T} \times \lim_{T\to+\infty} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \mathbb{e}^{-\frac{2\pi}{T} nxj} \mathbb{d}x \\ &= \frac{1}{T} \times \int_{-\infty}^{+\infty} f(x) \mathbb{e}^{-\omega xj} \mathbb{d}x \qquad (2) \end{aligned}$$

Equation (2) is actually the famous formula called **Fourier Transform**, but it seems quite different from that in the textbook. To get the textbook form, we need to examine its inverse transform known as **Inverse Fourier Transform**. Actually, IFT has already been induced in Equation (1):

$$f(t) = \sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{\frac{2\pi}{T} ntj}$$

Since $T \to +\infty$, let $\mathbb{d}\omega = \frac{2\pi}{T}$, then $\omega = \frac{2\pi}{T} n$. It can be written as:

$$f(t) = \frac{T}{2\pi} \sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{\frac{2\pi}{T} ntj} \times \mathbb{d}\omega \qquad (3)$$

This formula looks just like the definition of definite integral:

$$\int_a^b f(x) \mathbb{d}x = \lim_{\max \Delta x_i \to 0} \sum_i f(\xi_i) \Delta x_i \qquad (4)$$

So, we may transform it into an integral, but there is a problem then. We say $T \to +\infty$, and also, $|n| \to +\infty$, and how do we judge the range of $\frac{n}{T} = \frac{\omega}{2\pi}$, and to further determine the integral limits (a and b in formula (4))? Here, we think from the most ordinary aspect that **we want to analyze the frequency components of function $f(t)$, and its frequency range should be $\mathbb{R}$.**

According to the above statement, we know the range of $\frac{n}{T} = \frac{\omega}{2\pi}$ should also be $\mathbb{R}$, which is $(-\infty, +\infty)$. Referring to Formula (2): $F'(\omega) = c_n$, then Formula (3) can be written as:

$$\begin{aligned} f(t) &= \frac{T}{2\pi}\sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{\frac{2\pi}{T}ntj} \times \mathbb{d}\omega \\ &= \frac{T}{2\pi}\sum_{i=-\infty}^{+\infty} F'(\omega_i)\mathbb{e}^{\omega_i tj} \times \mathbb{d}\omega \\ &= \frac{T}{2\pi}\int_{-\infty}^{+\infty} F'(\omega)\mathbb{e}^{\omega tj}\mathbb{d}\omega \\ &= \frac{1}{2\pi}\int_{-\infty}^{+\infty} TF'(\omega)\mathbb{e}^{\omega tj}\mathbb{d}\omega \end{aligned} \tag{5}$$

Now, the answer is close. Combining Equation (4) and (5):

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} TF'(\omega)\mathbb{e}^{\omega tj}\mathbb{d}\omega$$

$$F'(\omega) = \frac{1}{T} \times \int_{-\infty}^{+\infty} f(x)\mathbb{e}^{-\omega xj}\mathbb{d}x$$

one can get:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} T \times \frac{1}{T} \times \int_{-\infty}^{+\infty} f(x)\mathbb{e}^{-\omega xj}\mathbb{d}x\,\mathbb{e}^{\omega tj}\mathbb{d}\omega$$

$$\boldsymbol{f}(\boldsymbol{t}) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} \left[\int_{-\infty}^{+\infty} \left(\boldsymbol{f}(\boldsymbol{x})\mathbb{e}^{-\omega xj}\right)\mathbb{d}x\,\mathbb{e}^{\omega tj}\right]\mathbb{d}\omega \tag{6}$$

Expression in the [ ] in Equation (6) is Fourier Transform, and the outer part is Inverse Fourier Transform. As we can see, $f(x)$ returns to $f(t)$ after the two reciprocal transforms. To be formal, let $F(\omega) = TF'(\omega)$, Fourier Transform is:

$$\boxed{F(\omega) = \int_{-\infty}^{+\infty} f(t)\mathbb{e}^{-j\omega t}\,\mathbb{d}t} \tag{7}$$

And Inverse Fourier Transform is:

$$\boxed{f(x) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)\mathbb{e}^{j\omega x}\mathbb{d}\omega} \tag{8}$$

Discrete Fourier Transform

Define the discrete signal function (if you ask me why we have to multiply a $\delta(x)$ to the original signal, I don't know😕, and it just works):

$$d(t) = \sum_k f(t)\delta(t - k\Delta t)$$

where, Δt is the time interval of sampling (we can only record discrete signal, but if Δt is small enough, the discrete signal can be considered as continuous).

Then, its Fourier Transform is:

$$F(\omega) = \int_{-\infty}^{+\infty} \sum_k f(t)\delta(t - k\Delta t)\mathbb{e}^{-j\omega t}\mathbb{d}t$$

where $\delta(x)$ is the unit-impulsive function, that $\int_{-\infty}^{+\infty}\delta(t)\mathbb{d}t = 1$, and $\delta(t) = 0$, if $t \neq 0$, and $\omega = \frac{2\pi n}{T}$ (9) .

So, $F(\omega)$ can be further written as:

$$F(\omega) = \sum_k \int_{-\infty}^{+\infty} f(t)\delta(t-k\Delta t)\mathbb{e}^{-j\frac{2\pi n}{T}t}\mathbb{d}t \qquad (10.1)$$

$$= \sum_k f(k\Delta t)\mathbb{e}^{-j\frac{2\pi n}{T}k\Delta t} \qquad (10.2)^1$$

If the sampling time interval is $t \in [0, T)$, then, $n, k \in \left[0, \frac{T}{\Delta t}\right)$, assuming $\frac{\Delta t}{T} = \frac{1}{N}, N \in \mathbb{N}$, and let $f(k\Delta t) = x(k)$, $X(n) = F(\omega) = F\left(\frac{2\pi n}{T}\right)$ (remember Equation (9) all the time):

$$X(n) = \sum_{k=0}^{N-1} x(k)\mathbb{e}^{-j\frac{2\pi k}{N}n}$$

This is relatively hard to understand, and in reality, the writer was confused when writing this part, and mistakenly used the notation k and n. So, if we swap the meaning of k and n, the textbook form of **Discrete Fourier Transform** appears:

$$\boxed{X(k) = \sum_{n=0}^{N-1} x(n)\mathbb{e}^{-j\frac{2\pi n}{N}k}} \qquad (11)$$

In Equation (11), $X(k)$ is the magnitude of frequency $\nu = \frac{2\pi k}{N}$. And we can also express it in matrix form:

$$\vec{X}(k) = \begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{pmatrix}, \vec{x}(n) = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

$$\begin{pmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{pmatrix} = \begin{pmatrix} \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times 1} & \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times 2} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times(N-1)} \\ \mathbb{e}^{-\frac{j2\pi}{N}\times 2\times 1} & \mathbb{e}^{-\frac{j2\pi}{N}\times 2\times 2} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times 2\times(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{e}^{-\frac{j2\pi}{N}\times(N-1)\times 1} & \mathbb{e}^{-\frac{j2\pi}{N}\times(N-1)\times 2} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times(N-1)\times(N-1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{pmatrix}$$

[1] Transform from Equation (10.1) to (10.2) seems quite hard to comprehend, but you can think in the following way. Since $\int_{-\infty}^{0} \delta(x)\mathbb{d}x = \int_{-\infty}^{0} \delta(x)\mathbb{d}x = 0$, we only need to consider

$$\int_{n\Delta t^-}^{n\Delta t^+} f(t)\delta(t-n\Delta t)\mathbb{e}^{-j\frac{2\pi n}{T}t}\mathbb{d}t \qquad (a.1)$$

Equation (a.1) seems absurd, but if you review the characteristic of unit-impulsive function:

$$\int_{-\infty}^{+\infty} \delta(t)\mathbb{d}t = 1$$
$$\delta(t) = 0, t \neq 0$$

So, $\delta(0)$ seems to be $+\infty$. Moreover, function $g(t) = f(t)\mathbb{e}^{-j\frac{2\pi n}{T}t}$ is a continuous function, when $t \to 0$, limit $g(t)$ exists.

Thus,

$$\int_{n\Delta t^-}^{n\Delta t^+} f(t)\delta(t-n\Delta t)\mathbb{e}^{-j\frac{2\pi n}{T}t}\mathbb{d}t = f(t)\mathbb{e}^{-j\frac{2\pi n}{T}t}\int_{n\Delta t^-}^{n\Delta t^+} \delta(t-n\Delta t)\mathbb{d}t$$
$$= f(t)\mathbb{e}^{-j\frac{2\pi n}{T}t} \times 1$$

Also, ω in (10.1) is not a continuous variable anymore, so we make it a series counting from 1, denoted as k.

Why did I promote its matrix form? Originally, I was about to use Equation (8) to induce **Inverse Discrete Fourier Transform (IDFT)**, but now $F(\omega)$ is not a continuous function anymore. Then how can you integrate it multiplied by $\mathbb{e}^{j\omega t}$:

$$d(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} F(\omega)\mathbb{e}^{j\omega t}\mathbb{d}\omega$$

Nevertheless, we can induce the formula via its matrix form. Let

$$\mathcal{F} = \begin{pmatrix} \mathbb{e}^{-\frac{j2\pi}{N}\times 0\times 0} & \mathbb{e}^{-\frac{j2\pi}{N}\times 0\times 1} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times 0\times (N-1)} \\ \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times 0} & \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times 1} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times 1\times (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{e}^{-\frac{j2\pi}{N}\times (N-1)\times 0} & \mathbb{e}^{-\frac{j2\pi}{N}\times (N-1)\times 1} & \cdots & \mathbb{e}^{-\frac{j2\pi}{N}\times (N-1)\times (N-1)} \end{pmatrix}$$

And there is

$$\vec{X} = \mathcal{F}\vec{x}$$
$$\mathcal{F}^{-1}\vec{X} = \mathcal{F}^{-1}\mathcal{F}\vec{x}$$
$$\mathcal{F}^{-1}\vec{X} = \vec{x}$$

All we have to do is to derive $\mathcal{F}^{-1}$, which is a pretty easy job. If the i-th row of $\mathcal{F}^{-1}$ is

$$\begin{pmatrix} \mathbb{e}^{\frac{j2\pi}{N}\times (i-1)\times 0} & \mathbb{e}^{\frac{j2\pi}{N}\times (i-1)\times 1} & \cdots & \mathbb{e}^{\frac{j2\pi}{N}\times (i-1)\times (N-1)} \end{pmatrix}$$

so that

$$\mathfrak{F}^{-1} = \begin{pmatrix} \mathbb{e}^{\frac{j2\pi}{N}\times 0\times 0} & \mathbb{e}^{\frac{j2\pi}{N}\times 0\times 1} & \cdots & \mathbb{e}^{\frac{j2\pi}{N}\times 0\times (N-1)} \\ \mathbb{e}^{\frac{j2\pi}{N}\times 1\times 0} & \mathbb{e}^{\frac{j2\pi}{N}\times 1\times 1} & \cdots & \mathbb{e}^{\frac{j2\pi}{N}\times 1\times (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{e}^{\frac{j2\pi}{N}\times (N-1)\times 0} & \mathbb{e}^{\frac{j2\pi}{N}\times (N-1)\times 1} & \cdots & \mathbb{e}^{\frac{j2\pi}{N}\times (N-1)\times (N-1)} \end{pmatrix}$$

Elements of $(\mathcal{F}\mathfrak{F}^{-1})$ can be easily obtained. The i-th row, and j-th column element is:

$$\sum_{k=0}^{N-1} \mathbb{e}^{j\frac{2\pi}{N}[(i-1)k-k(j-1)]} = \sum_{k=0}^{N-1} \mathbb{e}^{j\frac{2\pi}{N}k(i-j)}$$

Apparently,

If $i = j$ (for main diagonal elements):

$\sum_{k=0}^{N-1} \mathbb{e}^{j\frac{2\pi}{N}k(i-j)} = \sum_{k=0}^{N-1} \mathbb{e}^{j\frac{2\pi}{N}k\times 0} = N;$

if $i \neq j$ (for other elements):

$\sum_{k=0}^{N-1} \mathbb{e}^{j\frac{2\pi}{N}k(i-j)} = \sum_{k=0}^{N-1} \cos\left((i-j)k \times \frac{2\pi}{N}\right) + j\sin\left((i-j)k \times \frac{2\pi}{N}\right)$ is hard to calculate by algebraic method, but if we draw it in a complex plane as shown in Figure 2:

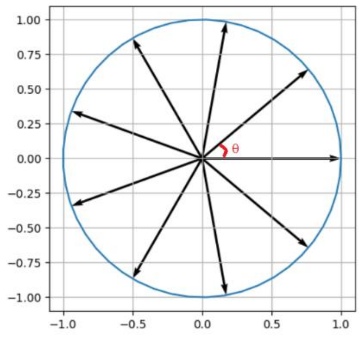

Figure 2 Summation of Complex Numbers in a Unit Circle

In Figure 2, $\theta = (i-j)\frac{2\pi}{N}$, and we can know the result is 0.

Thus, the presumed $\mathfrak{F}^{-1}$ and the real $\mathcal{F}^{-1}$ differ by only one coefficient N:

$$\mathcal{F}^{-1} = \frac{1}{N}\begin{pmatrix} e^{\frac{j2\pi}{N}\times 0\times 0} & e^{\frac{j2\pi}{N}\times 0\times 1} & \cdots & e^{\frac{j2\pi}{N}\times 0\times (N-1)} \\ e^{\frac{j2\pi}{N}\times 1\times 0} & e^{\frac{j2\pi}{N}\times 1\times 1} & \cdots & e^{\frac{j2\pi}{N}\times 1\times (N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ e^{\frac{j2\pi}{N}\times (N-1)\times 0} & e^{\frac{j2\pi}{N}\times (N-1)\times 1} & \cdots & e^{\frac{j2\pi}{N}\times (N-1)\times (N-1)} \end{pmatrix} \tag{12}$$

From Equation (12), we can obtain the IDFT formula:

$$\boxed{x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi k}{N}n}} \tag{13}$$

For IDFT and DFT, every $X(k)$ and $x(n)$ needs N times multiply and N times plus operations. And to get all the $X(k), k = 0, 1, 2, 3, \dots, N-1$, one need to calculate N^2 times multiply and plus operations, respectively. The time complexity is $O(n^2)$. As the data amount n increases, the time consumption will increase beyond imagination.

Fast Fourier Transform

Although $O(n^2)$ means square growth, if you see the thing reversely, it means $n^2 > \left(\frac{n}{2}\right)^2 + \left(\frac{n}{2}\right)^2$ that if we can divide the original job into two separate jobs, the cost of finishing two sub-jobs is lower than that of the original job.

How can we divide the job? Let's analyze the formula of DFT:

$$\boxed{X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi n}{N}k}}$$

We can find the following property:

$$\begin{aligned} X(N-k) &= \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi n}{N}(N-k)} \\ &= \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi n}{N}k}\, e^{-j\frac{2\pi n}{N}\times N} \\ &= \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi n}{N}k}\, e^{-j2\pi n} \\ &= \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi n}{N}k} \\ &= \overline{X(k)} \end{aligned}$$[2]

Now assume $N = 2^k, k \in \mathbb{N}^*$, then DFT can be written as:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) e^{-j\frac{2\pi\times 2n}{N}k} + \sum_{n=0.5}^{\frac{N-1}{2}} x(2n) e^{-j\frac{2\pi\times 2n}{N}k}$$

[2] $\bar{a}$ is the complex conjugate of a. $X(N-k)$ is the complex conjugate of $X(k)$

$$= \sum_{n=0}^{\frac{N}{2}-1} x(2n)\mathrm{e}^{-j\frac{2\pi\times 2n}{N}k} + \sum_{n=0}^{\frac{N}{2}-1} x(2n)\mathrm{e}^{-j\frac{2\pi\times 2(n+0.5)}{N}k}$$

Let $x_{\mathrm{o}}(n) = x(2n), n = 1,3,5,7,9, \dots$, $x_{\mathrm{e}}(n) = x(2n), n = 0,2,4,6,8, \dots$, then, for $0 < k \leq \frac{N}{2} - 1$:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x_{\mathrm{e}}(n)\mathrm{e}^{-j\frac{2\pi n}{\frac{N}{2}}k} + \sum_{n=0}^{\frac{N}{2}-1} x_{\mathrm{o}}(n)\mathrm{e}^{-j\frac{2\pi(n+0.5)}{\frac{N}{2}}k}$$

$$= \sum_{n=0}^{\frac{N}{2}-1} x_{\mathrm{e}}(n)\mathrm{e}^{-j\frac{2\pi n}{\frac{N}{2}}k} + \mathrm{e}^{-j\frac{2\pi}{N}k}\sum_{n=0}^{\frac{N}{2}-1} x_{\mathrm{o}}(n)\mathrm{e}^{-j\frac{2\pi n}{\frac{N}{2}}k}$$

$$= X_{\mathrm{e}}(k) + \mathrm{e}^{-j\frac{2\pi}{\frac{N}{2}}k} X_{\mathrm{o}}(k)\,[3] \tag{14}$$

For $k > \frac{N}{2}$,

$$X(k) = \overline{X\left(k - \frac{N}{2}\right)}$$

For $k = 0$,

$$X(0) = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\frac{2\pi n}{N}\times 0} = \sum_{n=0}^{N-1} x(n)$$

For $k = \frac{N}{2}$,

$$X\left(\frac{N}{2}\right) = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\frac{2\pi n}{N}k} = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\frac{2\pi n}{N}\times\frac{N}{2}} = \sum_{n=0}^{N-1} x(n)\mathrm{e}^{-j\pi n} = \sum_{n=0}^{N-1} (-1)^n x(n)$$

Now, the original job $X(k)$ with N length has been divided into two sub-jobs: $X_o(k)$ and $X_e(k)$ each with $\frac{N}{2}$ length. The story doesn't end yet. We can further divide $X_o(k)$ and $X_e(k)$ into smaller jobs until there is only one element left. Since $N = 2^k$, we have to divide $k = \log_2 N$ times, and every division cause one plus and on multiply operation. In total, k times plus operations, and $2k = 2\log_2 N$ times multiply operations for each $X(k), k = 0,1,2,3,4, \dots, N-1$. And there are N $X(k)$ in total, so we have to calculate $N \times 3k = N \times 3\log_2 N$ times. Thus, the time complexity is $O(n \log n)$.

The algorithm is shown below:

```
Algorithm description:
    This function receives one input array (length information must be included) and
    generate one output array. The output array is the DFT result of the input array. The
    length of the input array must be an integer power of 2.
Algorithm process:
    If the length of input array is 1:
        Just return the input scalar.
    Or else, if the length ($N$) of input array is larger than 1:
```

[3] The length of x_{e} is $\frac{N}{2}$, this is just the denominator of the power of e. So, this sub-item is just a DFT of another discrete series x_{e}. It's the same to x_{o}.

1. Divide the input array according to the order into odd order array x_o and even order array x_e.
2. Call myself to calculate DFT value of x_o and x_e. The return value is r_o and r_e respectively.
3. The number of DFT values of both x_o and x_e is $\frac{N}{2}$. So, Step 2 must be performed $\frac{N}{2}$ times. Write each DFT value into the corresponding position of the output array.
4. Return the output array.

The code of Python is shown in Table 1.

Table 1 Binary FFT in Python

```python
import numpy as np

def fft(x):
    if (type(x) != np.ndarray):
        x = np.array(x)
    F = np.zeros(x.shape, dtype=np.complex128)
    if(len(x) == 1):
        return x
    even = fft(x[::2])
    odd = fft(x[1::2])

    F[0] = x.sum()
    F[len(F)//2] = x[::2].sum()-x[1::2].sum()
    for k in range(1, len(F)//2):
        omega = np.exp(-2j*np.pi*k/len(x))
        F[k] = even[k] + omega*odd[k]
        F[len(F)-k] = F[k].conj()
    return F
```

The performance of Python is poor. C++ can improve the performance by about 400 times. The code of C++ is shown in Table 2. It is far more performant than Python code, but its run time is 8 times of that of fft of Scipy.

Table 2 Binary FFT in C++

```cpp
#include <complex>
#include <cmath>

namespace fft
{
        typedef long long LONG;
        //constexpr double M_PI = 3.14159265358979323846264338;

        template <typename Tn>
        std::complex<Tn>* fft(const std::complex<Tn>* data, std::complex<Tn>* F,
                LONG len, LONG start = -1, LONG end = -1, LONG step = -1)
        {
                if (!data || !F)
                {
                        return NULL;
                }

                if (start == -1)
                {
                        start = 0;
                        end = len;
                        step = 1;
                }

                if (len == 1)
                {
                        F[start] = data[start];
                        return F;
                }
                if ((len & (len-1)))
                        // len is not an interger power of 2
                {
                        return NULL;
                }

                fft(data, F, len / 2, start, end, step * 2);
                // calculate the even part.
                fft(data, F, len / 2, start + step, end, step * 2);
                // calculate the odd part.


                double theta = -2 * M_PI / len;
                double theta0 = -2 * M_PI / len;
                F[start] += F[start + step];
                std::complex<Tn> omega(std::cos(theta), std::sin(theta));
                auto omega0 = omega;
                for (LONG i = 1, j = start+step, k = start+2*step; i < len / 2; i +=
1, j+=step, k+=2*step)
                {
                        //std::complex<Tn> omega(std::cos(theta), std::sin(theta));
                        F[j] = F[k]             // the even part, even[i]
                                + omega * F[k+step];  // the odd part, odd[i]
                        omega *= omega0;
                }

                //std::complex<Tn> omega0(std::cos(theta0), std::sin(theta0));
                //auto omega = omega0;
                //for (LONG i = start+step, j = start+2*step, k = 1; k<len/2; i+=step,
j+=2*step, k+=1, theta+=theta0)
                //{
                //      //std::complex<Tn> omega(std::cos(theta), std::sin(theta));
                //      F[i] += F[j];
                //      F[i] += omega * F[j + step];
                //      omega *= omega0;
                //}

                for (LONG i = start+step, j = start+(len-1)*step, k = 1; k < len / 2;
                        i += step, j-=step, k+=1)
                {
                        F[j] = std::complex<Tn>(F[i].real(), -F[i].imag());
                }
                {
                        const auto N2 = start + (len / 2) * step;
                        F[N2] = 0;
                        for (LONG i = start; i < end; i += step * 2)
                        {
                                F[N2] += data[i];
                                F[N2] -= data[i + step];
                        }
                }
                return F;
        }


};

extern "C"
_declspec(dllexport)
double* fft_C(double* data, double* result, __int64 len)
// This function is a C-like style of fft::fft() function for external usage.
// data must have 2*len length. Its format is [real 0, imag 0, real 1, imag 1, ... ].
result is the same.
{
        if ((len & (len - 1)))
        {
                return NULL;
        }
        std::complex<double>* ddata = reinterpret_cast<std::complex<double>*> (data);
        std::complex<double>* rresult = reinterpret_cast<std::complex<double>*>
(result);


        return reinterpret_cast<double*>(fft::fft(ddata, rresult, len));
}
```

Given the original data is $\vec{x} = (0, 1, 2, 3, 4, \ldots, N)$. Time consumption of different codes is shown in Table 3.

Table 3 Time Consumption

Code	Time Consumption (ms)	
	$N = 32 \times 1024$	$N = 64 \times 1024$
Code of Python in Table 1	714	1520
Code of C++ in Table 2	4.49	11.494
Numpy.fft.fft()	0.761	1.86

It's seen that though C++ has better performance than Python, it is not as performant as Numpy.fft.fft().

[i] You'll have to proof cosine multiplies cosine; cosine, sine; sine, sine. I think this is boring. You can proof this simply using Prosthaphaeresis Formulas, and the proof here is omitted 😊 but the proof of the complex compression of Fourier Series is attached below.

[ii] Proofing:

$$f(x)=\sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{j\frac{2\pi}{T}nx}$$

Multiply $\mathbb{e}^{-j\frac{2\pi}{T}kx}$ on both sides of the equation:

$$f(x)\mathbb{e}^{-j\frac{2\pi}{T}kx}=\sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}$$

Integrate both sides of the equation from $x=0$ to $x=T$, and for the right-hand side:

$$\begin{aligned}\int_0^T \sum_{n=-\infty}^{+\infty} c_n \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\,\mathbb{d}x &= \sum_{n=-\infty}^{+\infty}\int_0^T c_n \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x \\ &= \sum_{n=-\infty}^{+\infty} c_n \int_0^T \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x\end{aligned}$$

If $\boldsymbol{n=k}$:

$$\begin{aligned}\boldsymbol{\int_0^T \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x} &= \int_0^T \mathbb{e}^{j\frac{2\pi}{T}\times 0x}\mathbb{d}x = \int_0^T 1\mathbb{d}x \\ &\boldsymbol{=T}\end{aligned}$$

If $\boldsymbol{n\neq k}$:

$$\int_0^T \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x=\frac{T}{j2\pi(n-k)}\mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\Big|_0^T=\frac{T}{j2\pi(n-k)}\left(\mathbb{e}^{2\pi j(n-k)}-1\right)$$

From Eula's Formular:

$$\mathbb{e}^{jx}=\cos x+j\sin x$$

, one can easily get $\mathbb{e}^{2\pi j(n-k)}=1$, so that $\boldsymbol{\int_0^T \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x=0}$.

Focusing on the **red bold font**, one can get the right-hand side after integrating:

$$\sum_{n=-\infty}^{+\infty} c_n \int_0^T \mathbb{e}^{j\frac{2\pi}{T}(n-k)x}\mathbb{d}x=c_kT$$

, while the left-hand side is:

$$\int_0^T f(x)\mathbb{e}^{-j\frac{2\pi}{T}kx}\mathbb{d}x$$

, and thus

$$c_k=\frac{1}{T}\int_0^T f(x)\mathbb{e}^{-j\frac{2\pi}{T}kx}\mathbb{d}x$$

Q.E.D. 😊