Understanding the Fast Fourier Transform (快速傅里叶变换)

2025-12-01 | Riguz

Tags: math, algorithms, signal-processing

1 Introduction

computational mathematics and signal processing. It converts a sequence of N complex numbers into another sequence of N complex numbers, revealing the frequency content of the original signal.

The **Discrete Fourier Transform** (DFT)[1] is one of the most important tools in computational mathematics and signal processing. It converts a sequence of N complex numbers into another sequence of N complex numbers, revealing the frequency content of the original signal.

DFT is the foundation of modern signal analysis.

For a historical perspective, see [1].

然而，直接计算 DFT 需要 $O(N^2)$ 次运算，对于大规模数据来说效率太低。1965 年，Cooley 和 Tukey 发表了**快速傅里叶变换**（FFT）算法，将复杂度降低到 $O(N \log N)$，使得频谱分析在实际工程中变得可行。

DFT 的直接算法复杂度很高。

FFT 是信号处理领域的革命性突破。

You can also use a sidenote in English:

This article walks through the mathematical foundation of the DFT, derives the radix-2 Cooley–Tukey FFT algorithm, and provides a Python implementation.

This is a right margin comment for extra context.

2 The Discrete Fourier Transform

Definition 2.1 (Discrete Fourier Transform)**.** *Given a sequence $x_0, x_1, \ldots, x_{N-1}$ of complex numbers, its DFT is the sequence $X_0, X_1, \ldots, X_{N-1}$ defined by:*

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

We often write $\omega_N = e^{-2\pi i/N}$, the *primitive N-th root of unity*, so the transform becomes:

$$X_k = \sum_{n=0}^{N-1} x_n\, \omega_N^{kn}$$

The inverse DFT recovers the original sequence:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k\, \omega_N^{-kn}$$

[1] See *The Scientist and Engineer's Guide to Digital Signal Processing* by Steven W. Smith.

© 2026 Riguz. All rights reserved.

2.1 Key Properties

The DFT satisfies several important properties:

- **Linearity**: $\mathrm{DFT}(\alpha x + \beta y) = \alpha\,\mathrm{DFT}(x) + \beta\,\mathrm{DFT}(y)$
- **Parseval's theorem**: $\sum_n |x_n|^2 = \frac{1}{N}\sum_k |X_k|^2$
- **Convolution theorem**: pointwise multiplication in the frequency domain corresponds to circular convolution in the time domain:
$$\mathrm{DFT}(x * y) = \mathrm{DFT}(x) \cdot \mathrm{DFT}(y)$$
- **Shift property**: 时域中的移位对应频域中的相位旋转。若 $y_n = x_{n-m}$，则 $Y_k = \omega_N^{mk} X_k$。

3 The Cooley–Tukey FFT Algorithm

The key insight of the FFT is to exploit the symmetry and periodicity of ω_N.

Theorem 3.1 (Danielson–Lanczos Lemma)**.** *An N-point DFT (where N is even) can be decomposed into two $N/2$-point DFTs:*

$$X_k = \underbrace{\sum_{m=0}^{N/2-1} x_{2m}\,\omega_{N/2}^{km}}_{E_k} + \omega_N^k \underbrace{\sum_{m=0}^{N/2-1} x_{2m+1}\,\omega_{N/2}^{km}}_{O_k}$$

where E_k is the DFT of the even-indexed elements and O_k is the DFT of the odd-indexed elements.

This gives us the **butterfly operation**:

$$X_k = E_k + \omega_N^k\, O_k \tag{1}$$
$$X_{k+N/2} = E_k - \omega_N^k\, O_k \tag{2}$$

Since E_k and O_k are periodic with period $N/2$, we only need to compute them for $k = 0, 1, \ldots, N/2-1$. 通过递归地应用这一分解，我们可以将 N 点 DFT 的计算分解为 $\log_2 N$ 层蝶形运算，每层包含 $N/2$ 次蝶形操作。

3.1 Complexity Analysis

Table 1: Comparison of DFT and FFT computational complexity

Algorithm	Multiplications	Additions	Total
Naive DFT	N^2	$N(N-1)$	$O(N^2)$
Radix-2 FFT	$\frac{N}{2}\log_2 N$	$N\log_2 N$	$O(N\log N)$

For a concrete example, consider $N = 2^{20} \approx 10^6$:

- Naive DFT: $\sim 10^{12}$ operations
- FFT: $\sim 10^7$ operations
- Speedup factor: $\sim 10^5$

© 2026 Riguz. All rights reserved.

4 Python Implementation

Below is a recursive radix-2 FFT implementation in Python.

```python
import numpy as np

def fft(x):
    """Compute the FFT of sequence x (length must be a power of 2)."""
    N = len(x)
    if N == 1:
        return x

    # Split into even and odd
    even = fft(x[0::2])
    odd  = fft(x[1::2])

    # Twiddle factors
    T = np.exp(-2j * np.pi * np.arange(N // 2) / N)

    # Butterfly
    return np.concatenate([
        even + T * odd,
        even - T * odd
    ])

# Example usage
if __name__ == "__main__":
    # Generate a signal: 50 Hz + 120 Hz
    fs = 1024           # Sampling rate
    t = np.arange(fs) / fs
    signal = np.sin(2 * np.pi * 50 * t) + 0.5 * np.sin(2 * np.pi * 120 * t)

    # Compute FFT
    spectrum = fft(signal)
    freqs = np.arange(fs) * fs / fs
    magnitudes = np.abs(spectrum) / fs

    print(f"Peak frequencies: {freqs[np.argsort(magnitudes)[-4:]]} Hz")
```

Listing 1: Recursive radix-2 FFT in Python

We can verify our implementation against NumPy's built-in FFT:

```python
x = np.random.random(1024)
assert np.allclose(fft(x), np.fft.fft(x))
print("FFT implementation verified!")
```

Listing 2: Verification against NumPy

© 2026 Riguz. All rights reserved.

5 Applications

The FFT has far-reaching applications across many domains:

1. **Signal processing**: spectral analysis, filtering, compression (e.g., MP3, JPEG)
2. **Polynomial multiplication**: multiplying two degree-n polynomials in $O(n \log n)$ instead of $O(n^2)$
3. **Large integer multiplication**: the Schönhage–Strassen algorithm uses FFT to multiply n-digit integers in $O(n \log n \log \log n)$
4. **Partial differential equations**: 谱方法利用 FFT 在频域中高效求解偏微分方程，在流体力学和量子力学模拟中广泛使用
5. **Convolution**: fast computation of convolutions via the convolution theorem, used in deep learning (convolutional neural networks)

6 Conclusion

For more on the DFT, refer back to Section 2. For the FFT algorithm, see Section 3. For code, see Section 4. For real-world uses, see Section 5.

References

[1] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation*, vol. 19, no. 90, pp. 297–301, 1965.

FFT 是计算数学中最优美、最实用的算法之一。它将 DFT 的计算复杂度从 $O(N^2)$ 降低到 $O(N \log N)$，使得大规模频谱分析成为可能。From signal processing to number theory, from image compression to solving PDEs, the FFT remains an indispensable tool in the modern computational toolkit.

The key idea — *divide and conquer via the symmetry of roots of unity* — is both mathematically elegant and practically powerful. Understanding the FFT provides deep insight into the interplay between the time domain and the frequency domain, a duality that lies at the heart of much of applied mathematics.

© 2026 Riguz. All rights reserved.