Discrete Fourier Transform [1]

The discrete Fourier transform (DFT) has several practical applications including:

- signal processing
- partial differential equations
- polynomial multiplication and interpolation

A simple practical example: Consider the Fourier series representation of a continuous periodic function on the interval $[0\, 2\pi]$:

$$f(x) = a_0 + \sum_{k=1}^{\infty}(a_k \cos kx + b_k \sin kx)$$

Using the Euler's formulas

$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}, \quad \sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

the complex form of the Fourier series may be written

$$f(x) = \sum_{k=-\infty}^{\infty} c_k e^{-ikx}$$

where

$$c_0 = a_0, \quad c_k = \frac{a_k + ib_k}{2} \tag{1}$$

The complex Fourier coefficients are given by the formula:

$$c_k = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{ikx}\, dx \tag{2}$$

and if f is a real-valued function then $c_{-k} = \bar{c}_k$. Once the complex coefficients are found, a_k and b_k may be recovered from (1).

If numerical integration with the nodes $h = 2\pi/n$, $x_j = 2j\pi/n$, $j = 0, 1, \ldots n$ is used to evaluate (2) the approximation formula become:

$$c_k = \frac{1}{n}\sum_{j=0}^{n-1} f(x_j)e^{ikx_j} = \frac{1}{n}\sum_{j=0}^{n-1} f(x_j)e^{2\pi ikj/n} = \frac{1}{n}\sum_{j=0}^{n-1} f(x_j)\left(e^{2\pi i/n}\right)^{kj}, k = 0, 1, \ldots \tag{3}$$

Definition: Given a vector $a = (a_0, a_1, \ldots, a_{n-1})$ we define the Discrete Fourier Transform of a as the vector $y = (y_0, y_1, \ldots, y_{n-1})$ whose components are

$$y \stackrel{def}{=} DFT_n(a); \quad y_k = \sum_{j=0}^{n-1} a_j \omega_n^{kj}, \quad k = 0, 1, \ldots, n-1 \tag{4}$$

[1] Reference: "Introduction to Algorithms" by T.H. Cormen, C.E. Leiserson and R.L. Rivest. MIT Press, 1990.

where ω_n is the principal n^{th} root of the unity

$$\omega_n = e^{2\pi i/n}$$

All other complex roots of order n of the unity are powers of ω_n

$$\omega_n^0, \omega_n^1, \ldots, \omega_n^{n-1}$$

The problem of evaluating $DFT(a)$ is equivalent to the problem of evaluating the polynomial

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \tag{5}$$

at the complex roots of order n of the unity, since

$$y_k = A(\omega_n^k), \quad k = 0, 1, \ldots n-1$$

If we define the DFT matrix as

$$F \in \mathbb{R}^{n\times n}, F_{k,j} = \omega_n^{kj}, \quad 0 \le k, j \le n-1 \tag{6}$$

then y may be written as the matrix vector product

$$y = Fa \tag{7}$$

For example, with $n = 4$ the DFT matrix is:

$$F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega_4 & \omega_4^2 & \omega_4^3 \\ 1 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ 1 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{bmatrix}$$

The inverse discrete Fourier transform (IDFT) is

$$DFT^{-1}(y) = F^{-1}y \tag{8}$$

and one notices that IDFT solves the problem of interpolation at the roots of the unity:

> *Given the data points* $(\omega_n^k, y_k), k = 0, 1, \ldots n-1$ *find the polynomial of* $A(x)$ *degree* $n-1$ *such that* $A(\omega_n^k) = y_k, k = 0, 1, \ldots n-1$

Evaluating $DFT(a)$ using standard multiplication has a computational complexity of order $O(n^2)$. The *Fast Fourier Transform* (FFT, Cooley-Tukey 1965) provides an algorithm to evaluate DFT with a computational complexity of order $O(n \log n)$ where $\log = \log_2$. The IDFT may be then evaluated as well using $O(n \log n)$ operations. The computational savings are significant. For example, if $n = 10^3$, the standard DFT requires $\sim 10^6$ operations, whereas FFT will involve only $\sim 10^4$, thus a computational saving by a factor of 100.