

Fast Fourier transform

Fast Fourier transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT).

Computing DFT of N points in the naive way, using the definition, takes $O(N^2)$ arithmetic operations, while an FFT can compute the same result in only $O(N \log N)$ operations. The difference in speed can be substantial, especially for large data sets. This improvement made many DFT-based algorithms practical.

Since the inverse of a DFT is also a DFT any FFT algorithm can be used in for the inverse DFT as well.

The most well known FFT algorithms, like the Cooley-Tukey algorithm, depend upon the factorization of N . However, there are FFTs with $O(N \log N)$ complexity for all N , even for prime N .

Discrete Fourier Transform

For a set of complex numbers x_n , $n = 0, \dots, N - 1$, the DFT is defined as a set of complex numbers c_k ,

$$c_k = \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{nk}{N}}, k = 0, \dots, N - 1. \quad (1)$$

The inverse DFT is given by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k e^{+2\pi i \frac{nk}{N}}. \quad (2)$$

These transformations can be viewed as expansion of the vector x_n in terms of the orthogonal basis of vectors $e^{2\pi i \frac{kn}{N}}$,

$$\sum_{n=0}^{N-1} \left(e^{2\pi i \frac{kn}{N}} \right) \left(e^{-2\pi i \frac{k'n}{N}} \right) = N \delta_{kk'}. \quad (3)$$

The DFT represent the amplitude and phase of the different sinusoidal components in the input data x_n .

The DFT is widely used in different fields, like spectral analysis, data compression, solution of partial differential equations and others.

Applications

Data compression Several lossy (that is, with certain loss of data) image and sound compression methods employ DFT as an approximation for the Fourier series. The signal is discretized and transformed, and then the Fourier coefficients of high/low frequencies, which are assumed to be unnoticeable, are discarded. The decompressor computes the inverse transform based on this reduced number of Fourier coefficients.

Partial differential equations Discrete Fourier transforms are often used to solve partial differential equations, where the DFT is used as an approximation for the Fourier series (which is recovered in the limit of infinite N). The advantage of this approach is that it expands the signal in complex exponentials e^{inx} , which are eigenfunctions of differentiation: $\frac{d}{dx} e^{inx} = in e^{inx}$. Thus, in the Fourier representation, differentiation is simply multiplication by in .

A linear differential equation with constant coefficients is transformed into an easily solvable algebraic equation. One then uses the inverse DFT to transform the result back into the ordinary spatial representation. Such an approach is called a *spectral method*.

Convolution and Deconvolution FFT can be used to efficiently compute convolutions of two sequences. A convolution is the pairwise product of elements from two different sequences, such as in multiplying two polynomials or multiplying two long integers.

Another example comes from data acquisition processes where the detector introduces certain (typically Gaussian) blurring to the sampled signal. A reconstruction of the original signal can be obtained by deconvoluting the acquired signal with the detector's blurring function.

Cooley-Tukey algorithm

In its simplest incarnation this algorithm re-expresses the DFT of size $N = 2M$ in terms of two DFTs of size M ,

$$\begin{aligned}
 c_k &= \sum_{n=0}^{N-1} x_n e^{-2\pi i \frac{nk}{N}} \\
 &= \sum_{m=0}^{M-1} x_{2m} e^{-2\pi i \frac{mk}{M}} + e^{-2\pi i \frac{k}{N}} \sum_{m=0}^{M-1} x_{2m+1} e^{-2\pi i \frac{mk}{M}} \\
 &= \begin{cases} c_k^{(\text{even})} + e^{-2\pi i \frac{k}{N}} c_k^{(\text{odd})} & , k < M \\ c_{k-M}^{(\text{even})} - e^{-2\pi i \frac{k-M}{N}} c_{k-M}^{(\text{odd})} & , k \geq M \end{cases} \quad (4)
 \end{aligned}$$

where $c^{(\text{even})}$ and $c^{(\text{odd})}$ are the DFTs of the even- and odd-numbered sub-sets of x .

This re-expression of a size- N DFT as two size- $\frac{N}{2}$ DFTs is sometimes called the Danielson-Lanczos lemma. The exponents $e^{-2\pi i \frac{k}{N}}$ are called *twiddle factors*.

The operation count by application of the lemma is reduced from the original N^2 down to $2(N/2)^2 + N/2 = N^2/2 + N/2 < N^2$.

For $N = 2^p$ Danielson-Lanczos lemma can be applied recursively until the data sets are reduced to one datum each. The number of operations is then reduced to $O(N \ln N)$ compared to the original $O(N^2)$. The established library FFT routines, like FFTW and GSL, further reduce the operation count (by a constant factor) using advanced programming techniques like precomputing the twiddle factors, effective memory management and others.

Multidimensional DFT

For example, a two-dimensional set of data $x_{n_1 n_2}$, $n_1 = 1 \dots N_1$, $n_2 = 1 \dots N_2$ has the discrete Fourier transform

$$c_{k_1 k_2} = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x_{n_1 n_2} e^{-2\pi i \frac{n_1 k_1}{N_1}} e^{-2\pi i \frac{n_2 k_2}{N_2}} \quad (5)$$

C implementation

```

#include<complex.h>
#include<tgmath.h>
#define PI 3.14159265358979323846264338327950288
void dft(int N, complex * x, complex * c, int sign)
{
    complex w = cexp(sign * 2 * PI * I / N);
    for (int k = 0; k < N; k++) {
        complex sum = 0;
        for (int n = 0; n < N; n++)
            sum += x[n] * cpow(w, n * k);
        c[k] = sum / sqrt(N);
    }
}

void fft(int N, complex * x, complex * c, int sign)
{
    if (N % 2 == 0) {
        complex w = exp(sign * 2 * PI * I / N);
        int M = N / 2;
        complex xo[M], co[M], xe[M], ce[M]; // VLA: compile with -std=c99
        for (int m = 0; m < M; m++) {
            xo[m] = x[2 * m + 1];
            xe[m] = x[2 * m];
        }
        fft(M, xo, co, sign);
        fft(M, xe, ce, sign);
        for (int k = 0; k < M; k++)
    
```

```
    c[k] = (ce[k] + cpow(w, k) * co[k]) / sqrt(2);  
for (int k = M; k < N; k++)  
    c[k] = (ce[k - M] + cpow(w, k) * co[k - M]) / sqrt(2);  
} else  
dft(N, x, c, sign);  
}
```