

Assignment 1: Fast Fourier Transform

Junjia Liu¹

¹School of Mechanical Engineering, Shanghai Jiaotong University,
junjialiu@sjtu.edu.cn

October 23, 2018

Contents

1	Fourier Transform	2
1.1	History of Fourier Transform	2
1.2	The meaning of Fourier Transform	2
1.3	Defination	3
2	Fast Fourier Transform	3
2.1	Introduction	3
2.2	Algorithm	4
2.2.1	Root of unity	4
2.2.2	Butterfly diagram	5
2.2.3	Simplify of DFT	6
2.2.4	FFT in Matlab	7

1 Fourier Transform

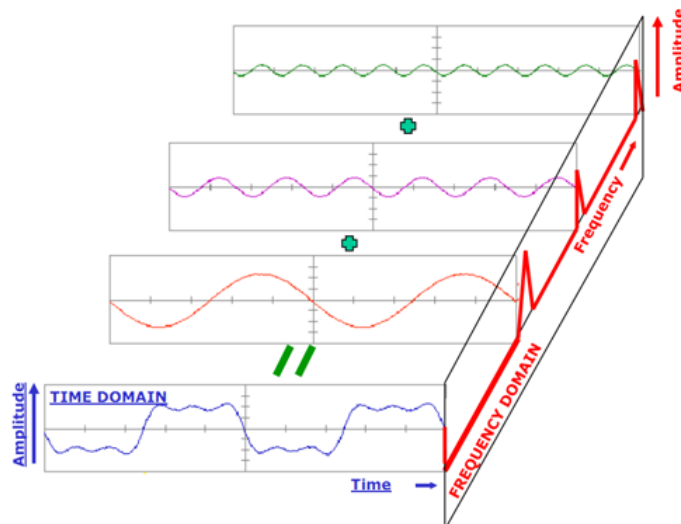
1.1 History of Fourier Transform

First of all, Fast Fourier Transform (FFT) is a fast algorithm of Discrete Fourier Transform (DFT). When it comes to FFT, we naturally have to explain the Fourier Transform first. Let's first take a look at where the Fourier transform came from? Fourier is the name of a French mathematician and physicist. He is very interested in heat transfer. In 1807, he published a paper in the French Academy of Sciences which use a sinusoidal curve to describe the temperature distribution. The paper has a controversial proposition at the time: **Any continuous cycle signals can be composed of a set of appropriate sinusoidal signals.** Two of the people who reviewed the paper at the time were famous mathematicians such as Lagrange (1736-1813) and Laplace (1749-1827). When Laplace and other reviewers voted to publish the paper, Lagrange resolutely opposed it. For nearly 50 years, Lagrange insisted that Fourier's method could not express an angular signal. The French Science Society succumbed to the authority of Lagrange and rejected the work of Fourier. It was not until 15 years after Lagrange's death that the paper was published. Who is right? Lagrange is right: sinusoids cannot be combined into a signal with an angular angle. However, we can use sinusoids to represent it very approximately without energy differences. So based on this, Fourier is right.

1.2 The meaning of Fourier Transform

The reason why we use sinusoids to replace original signal rather than square waves or triangle waves is that the sine and cosine have properties that other signals do not have: sinusoidal fidelity. If the input is a sinusoidal signal, then the output is still sinusoidal, only the amplitude and phase may change, but the frequency and shape of the wave are still the same. It is a property which only the sinusoid has, that is why we don't use other waves.

The Fourier principle shows that any continuously measured sequence or signal can be represented as an infinite superposition of sinusoidal signals of different frequencies. According to this principle, the Fourier transform algorithm uses the original signal to calculate the frequency, amplitude and phase of different sinusoidal signals in an accumulated manner. In the physical perspective, it is actually a way to help us change the mind of traditional time domain analysis to the frequency domain. The following 3D graphics can help us have a better understanding:



1.3 Defination

Suppose $x(t)$ is a continuous time signal (the signal must be not a periodic signal) and satisfies

$$\int_{-\infty}^{\infty} |x(t)| dt < \infty$$

Then, FT of this signal exists, defined as

$$X(j\Omega) = \int_{-\infty}^{\infty} x(t)e^{-j\Omega t} dt$$

Its inverse transform is defined as

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(j\Omega)e^{j\Omega t} d\Omega$$

However, if a CT periodic signal $x(t) = x(t + nT)$ satisfies *Dirichlet conditions*, it can also be rewritten into a Fourier series.

- $x(t)$ must have a finite number of extrema in any given interval.
- $x(t)$ must have a finite number of discontinuities in any given interval.
- $x(t)$ must be absolutely integrable over a period.
- $x(t)$ must be bounded.

2 Fast Fourier Transform




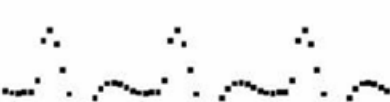
"The FFT is one of the truly great computational developments of this [20 th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT." (Charles van Loan)

2.1 Introduction

According to the type of input signal being transformed, the Fourier transform can be divided into four types:

1. Fourier Transform
2. Fourier Series
3. Discrete Time Fourier Transform
4. Discrete Fourier Transform

Here are four legends of the original signal :

Type of Transform	Example Signal
Fourier Transform <i>signals that are continuous and aperiodic</i>	
Fourier Series <i>signals that are continuous and periodic</i>	
Discrete Time Fourier Transform <i>signals that are discrete and aperiodic</i>	
Discrete Fourier Transform <i>signals that are discrete and periodic</i>	

Here we are talking about Discrete Fourier Transform. The DFT can transform the signal from the time domain to the frequency domain, and both of them are discrete. In other words, it can be obtained which sine waves a signal consists of, and the result is the amplitude and phase of these sine waves. So how can we know whether a sine wave is included or not. We can use the correlation of the signal to detect whether the signal wave contains another signal wave with a certain frequency: multiply the original signal by another wave, obtain a new signal wave, and then add all the amplitudes of each points in the new signal wave. The similarity of the two signals can be judged from the results. This is the principle of DFT, which is defined as:

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

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

For each value of DFT, there are N complex multiplications and $N - 1$ complex additions, so if we calculate by using the definition, it requires N^2 complex multiplications and $N(N - 1)$ complex additions. *FFT* manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size.

2.2 Algorithm

2.2.1 Root of unity

An n th root of unity, where n is a positive integer (*i.e.* $n = 1, 2, 3, \dots$), is a number ω satisfying the equation

$$\omega^n = 1$$

For example, 3th and 8th root of unity are shown:

Properties of ω_N^k :

1. Periodic: $\omega_N^{nk} = \omega_N^{(n+N)k}$

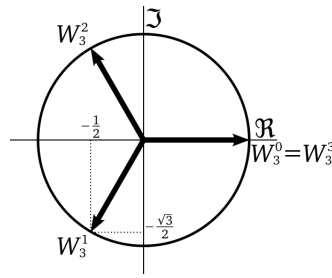


Figure 1: 3th root of unity

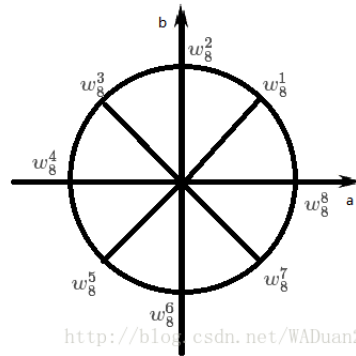


Figure 2: 8th root of unity

2. Conjugate Symmetry: $(\omega_N^{nk})^* = \omega_N^{(N-n)k} = \omega_N^{n(N-k)}$
3. Symmetric: $\omega_N^{k+\frac{N}{2}} = -\omega_N^{nk}$
4. $\omega_N^{\frac{N}{2}} = \omega_N^{-\frac{N}{2}} = -1$
5. $\omega_N^{2n} = \omega_{\frac{N}{2}}^n$

2.2.2 Butterfly diagram

In the context of fast Fourier transform algorithms, a butterfly is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into subtransforms).

Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite size $n = rm$ into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via size- r butterflies, which themselves are DFTs of size r (performed m times on corresponding outputs of the sub-transforms) pre-multiplied by roots of unity (known as twiddle factors).

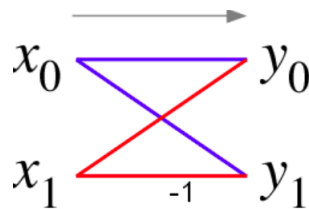


Figure 3: Radix-2 butterfly diagram

In the case of the radix-2 Cooley–Tukey algorithm, the butterfly is simply a DFT of size-2 that takes two inputs (x_0, x_1) (corresponding outputs of the two sub-transforms) and gives two outputs (y_0, y_1) by the formula:

$$y_0 = x_0 + x_1$$

$$y_1 = x_0 - x_1$$

More specifically, a radix-2 decimation-in-time FFT algorithm on $n = 2^p$ inputs with respect to a primitive n -th root of unity $\omega_N^k = e^{-\frac{j2\pi nk}{N}}$ relies on $O(n \log n)$ butterflies of the form:

$$y_0 = x_0 + x_1 \omega_N^k$$

$$y_1 = x_0 - x_1 \omega_N^k$$

where k is an integer depending on the part of the transform being computed.

2.2.3 Simplify of DFT

Based on the properties of the root of unity, the series $y_k = \sum_{n=0}^{N-1} \omega_N^{kn} x_n$ can be divided into two parts,

$$\begin{aligned} y_k &= \sum_{n=2t} \omega_N^{kn} x_n + \sum_{n=2t+1} \omega_N^{kn} x_n \\ &= \sum_t \omega_N^{kt} x_{2t} + \omega_N^k \sum_t \omega_N^{kt} x_{2t+1} \\ &= F_{\text{even}}(k) + \omega_N^k F_{\text{odd}}(k) \end{aligned} \quad (i \in \mathbb{Z})$$

Where $F_{\text{even}}(k)$ and $F_{\text{odd}}(k)$ is $N/2$ -point transformation for the even and odd sequences of $\{x_n\}_0^{N-1}$. The equation just calculate top $N/2$ points of y_k , and the other points can be easily obtained by using the symmetry of root of unity, because both $F_{\text{even}}(k)$ and $F_{\text{odd}}(k)$ are functions with a period of $N/2$,

$$\begin{aligned} y_k &= F_{\text{even}}(k) + \omega_N^k F_{\text{odd}}(k) \\ y_{k+\frac{N}{2}} &= F_{\text{even}}(k) - \omega_N^k F_{\text{odd}}(k) \end{aligned}$$

Thus, an N -point transformation is divided into two $N/2$ -point transformations. Continue to decompose as such. This is the basic principle of the Cooley-Tukey Fast Fourier Transform algorithm. According to the master theorem, it is not difficult to analyze the time complexity of the algorithm which is $O(n \log n)$.

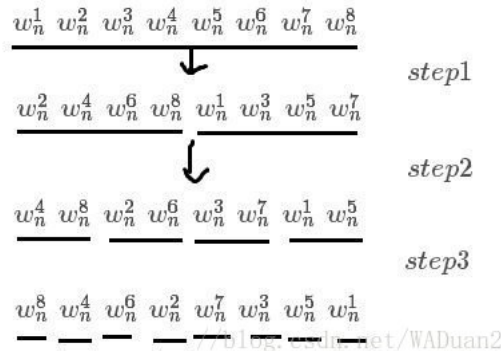


Figure 4: Schematic diagram of FFT of 8-point transformation

2.2.4 FFT in Matlab

It is easy to achieve FFT algorithm in Matlab,

```
1 Y = fft(X)
2 Y = fft(X,n)
3 %returns n-point DFT
4 Y = fft(X,n,dim)
5 %returns the Fourier transform along the dimension dim.
```

```
1 function [X,Y] = Real2FFT( x,y )
2 % N-point FFT is used for 2 real signals both with length N
3 N = length(x);
4 z = x+y*i ;
5 Z = fft(z);
6 X = 0.5*(Z+conj(Z(1+mod(0:-1:1-N,N))));
7 Y = 0.5*(Z-conj(Z(1+mod(0:-1:1-N,N))))/i;
8 end
```