

MATH 302 Group Project Report

Fast Fourier Transform

Group members:

Jinchen Zhao, Tianyu Wu, Qianyu Pan, and Huyue Yan

Division of Applied Mathematics

Duke Kunshan University

December 17, 2020

Contents

1	Introduction	2
2	Literature Review	3
3	Methodology	4
3.1	Fourier Series	4
3.2	Fourier Transform	5
3.3	Discrete Fourier Transform	5
3.4	From Discrete Fourier Transform to Fast Fourier Transform	6
3.5	Extension: Butterfly Diagram	8
4	Applications	10
4.1	Large Polynomial and Large Integer Multiplication	10
4.2	Signal Decomposition	10
4.3	Signal Filtering	11
4.4	Image Processing	12
5	Conclusion	13
6	Appendix	15

1 Introduction

A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT). In Fourier analysis, we convert a signal from its original domain to a representation in the frequency domain, and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies. This method is very useful in many fields. However, computing it directly requires much storage and often achieves slow efficiency. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. A naive application of the Fourier transform requires $O(n^2)$ operations, where N is the size of the data. While FFT, surprisingly, can apply to the case where $N = 2^k$ for positive integer k and only needs $O(n \log n)$ operations, which is much faster for larger n , compared to DFT.

This tremendous improvement has had a significant impact on many fields and has promoted the development of areas of digital signal processing and numerical convolution. In this report, we will have a deep look at FFT, including its historical development, methodology and implementations. Some of the related theories and applications will also be introduced. We will also introduce an extension to the FFT computing.

2 Literature Review

In general, integral transforms are useful tools for solving problems involving certain types of partial differential equations (PDEs), mainly when their solutions on the corresponding domains of definition are difficult to deal with. For a given PDE defined on a domain, the application of a suitable integral transform allows it to be expressed in such a form that its mathematical manipulation is easier than the original one. Scientists are always trying to find an optimal method to deal with different forms of differential equations. Among these different integral transforms, Fourier Transform is one of the most well-known methods in the applied mathematics community [1].

The presentation of Fourier Transform is highly dependent on the concept of Fourier series, and this idea was initially put forward when Joseph Fourier, a French mathematician and physicist, tried to solve heat equations of the form

$$\begin{aligned}\frac{\partial u}{\partial t}(x, t) &= \frac{\partial^2 u}{\partial x^2}(x, t) \quad \text{for } (x, t) \in \mathbb{R} \times (0, \infty) \\ u(x, 0) &= f(x) \quad \text{for } x \in \mathbb{R}\end{aligned}$$

He noticed that the heat equation is linear, and the family of functions $\sin(kx)e^{-k^2t}$ and $\cos(kx)e^{-k^2t}$ are solutions of the heat equation. Therefore, if we can express $u(x, 0)$ as the sum of sine and cosine functions $\sin(k_i x)$, $\cos(k_i x)$ then each of these components will decay over time in a known way and we can write down an expression for $u(x, t)$ as the sum of these decaying sine and cosine components [2].

Finally, in 1811, Fourier officially presented the concept of Fourier Transform in mathematics research, where he introduced a method of transforming a finite set of evenly-distributed samples from its original domain (e.g. time) to the frequency domain.

This mathematical concept was soon widely applied to the infinite set of complex numbers and involves integration benefited by the Euler's Formula [3]. This continuous type of Fourier transform has many applications in physics and engineering, but it was noticed that a discrete form of the Fourier transform exists, and that can be more easily performed by computer systems [4]. The discrete Fourier Transform (DFT) and its applications in computer science led to the advent of the Fast Fourier Transform (FFT) in 1965 by mathematical scientists James Cooley and John Tukey [5]. This new method enables the FFT algorithm to reduce the computational complexity of the DFT from $O(n^2)$ to $O(n \log n)$. This significant decrease in computational complexity enabled the initial algorithm to be more viable to many difficult problems, such as digital signal processing and numerical convolution. These applications made FFT one of the most commonly utilized algorithms by modern computers [6].

In the past few decades, modern computers have become exponentially faster over the past several decades. As a result, scientists started to use parallelization to overcome computation deficiencies. The idea is to solve a large-scale problem in parallel through using many computers or their multi-core architectures, thus decreasing the overall computation time [7]. Specifically, in 1968, the first parallel FFT paper was introduced, outlining a parallel FFT algorithm and proposing hardware to implement this algorithm [8]. Although it did not present any empirical data, the idea gained some popularity. Shortly after, scientist Bergland published two theoretical papers that extended the idea of a parallel Fast Fourier Transform algorithm [9]. Three years later, he introduced an implementation of such a parallel algorithm, including details of mapping addresses to processors and correctly ordering the input/output arrays [10].

Concurrently, important research was being done in the field of sequential FFT algorithms, in which there are several common variations of the traditional Cooley-Tukey FFT algorithm [5]. For instance, the mixed-radix method is published to reduce the number of recursion steps in the calculation [11], and the split-radix method reduces the number of multiplications by half, compared to the original FFT algorithm [12].

3 Methodology

3.1 Fourier Series

Fourier series is a periodic function composed of harmonically related sinusoids, combined by a weighted summation.

First consider a real-valued function, $s(x)$, which is integrable on an interval of length P , which is the period of the Fourier series. The analysis process determines the weights, indexed by integer n , which is also the number of cycles of the n^{th} harmonic function in the analysis interval. Therefore, the length of a cycle, in the units of x , is P/n , and the corresponding harmonic frequency is n/P . Followed by these, the n^{th} harmonics are $\sin(2\pi \frac{n}{P})$ and $\cos(2\pi \frac{n}{P})$, and their amplitudes (weights) are found by integration over the interval of length P :

$$a_n = \frac{2}{P} \int_P s(x) \cdot \cos(2\pi x \frac{n}{P}) dx \quad b_n = \frac{2}{P} \int_P s(x) \cdot \sin(2\pi x \frac{n}{P}) dx$$

If $s(x)$ is P -periodic, then any interval of that length is sufficient. a_0 and b_0 can be reduced to $a_0 = \frac{2}{P} \int_P s(x) dx$ and $b_0 = 0$. In many texts, $P = 2\pi$ is commonly chosen for the sinusoid functions.

The actual Fourier series is:

$$s_N(x) = \frac{a_0}{2} + \sum_{n=1}^N (a_n \cos(\frac{2\pi nx}{P}) + b_n \sin(\frac{2\pi nx}{P}))$$

Using a trigonometric identity:

$$A_n \cdot \cos(\frac{2\pi nx}{P} - \varphi_n) \equiv \underbrace{A_n \cos(\varphi_n)}_{a_n} \cdot \cos(\frac{2\pi nx}{P}) + \underbrace{A_n \sin(\varphi_n)}_{b_n} \cdot \sin(\frac{2\pi nx}{P}),$$

and definitions:

$$A_n \triangleq \sqrt{a_n^2 + b_n^2}$$

and

$$\varphi \triangleq \arctan^2(b_n, a_n)$$

the sine and cosine pairs can be expressed as a single sinusoid with a phase offset, analogous to the conversion between orthogonal (Cartesian) and polar coordinates:

$$s_N = \frac{A_0}{2} + \sum_{n=1}^N A_n \cdot \cos(\frac{2\pi nx}{P} - \varphi_n)$$

We can also generate complex-valued $s(x)$ using Euler's formula to split the cosine function into complex exponentials. Use an asterisk to denote the complex conjugation, gives us:

$$\begin{aligned} \cos(\frac{2\pi nx}{P} - \varphi_n) &\equiv \frac{1}{2} e^{i(\frac{2\pi nx}{P} - \varphi_n)} + \frac{1}{2} e^{-i(\frac{2\pi nx}{P} - \varphi_n)} \\ &= (\frac{1}{2} e^{-i\varphi_n}) \cdot e^{i\frac{2\pi(+n)x}{P}} + (\frac{1}{2} e^{-i\varphi_n})^* \cdot e^{i\frac{2\pi(-n)x}{P}} \end{aligned}$$

Therefore, with definitions:

$$c_n \triangleq \begin{cases} A_0/2 & = a_0/2 & n = 0 \\ \frac{A_n}{2} e^{-i\varphi_n} & = \frac{1}{2}(a_n - ib_n) & n > 0 \\ c_{|n|}^* & & n < 0 \end{cases} = \frac{1}{P} \int_P s(x) \cdot e^{-i\frac{2\pi nx}{P}} dx$$

the final result is:

$$s_N(x) = \sum_{n=-N}^N c_n \cdot e^{i\frac{2\pi nx}{P}}$$

3.2 Fourier Transform

In the study of Fourier series, complicated but periodic functions are written as the sum of simple waves mathematically, represented by sines and cosines. The Fourier transform is an extension of the Fourier series that results when the period of the represented function is lengthened and allowed to approach infinity.

Fourier transform (FT) is a mathematical transform that decomposes a function (often a function of time, or signal) into constituting frequencies. The Fourier transform of a function of time is a complex-valued function of frequency, whose magnitude (absolute value) represents the amount of that frequency present in the original function, and whose argument is the phase offset of the basic sinusoid in that frequency. The Fourier transform is not limited to functions of time, but the domain of the original function is commonly referred to as the time domain. There is also an inverse Fourier transform that mathematically synthesizes the original function from its frequency domain representation, as proven by the Fourier inversion theorem.

The Fourier transform of a function is denoted as \hat{f} . We define:

$$\hat{f} = \int_{-\infty}^{\infty} f(x)e^{-2\pi i x \xi} dx$$

for real number ξ .

In the expression, the sign in the exponent is negative because it is common in electrical engineering to represent by $f(x) = e^{2\pi i \xi_0 x}$, with initial phase zero and frequency ξ_0 .

If we inverse this expression, let the independent variable x represents time, the transform variable ξ represents frequency. Under suitable conditions, f is determined by \hat{f} via the inverse transform:

$$f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi)e^{2\pi i x \xi} d\xi$$

for any real number x .

This expression is known as the Fourier inversion theorem. The functions f and \hat{f} are often referred to as a Fourier integral pair, or Fourier transform pair.

3.3 Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is the equivalent of the continuous Fourier Transform for signals known only at N instants separated by sample times T (i.e. a finite sequence of data).

Let $f(t)$ be the continuous signal which is the source of the data. Let N samples be denoted $f_0, f_1, f_2, \dots, f_k, \dots, f_{N-1}$.

The Fourier Transform of the original signal, $f(t)$, would be

$$F(i\omega) = \int_{-\infty}^{\infty} e^{-i\omega t} dt$$

We could regard each sample $f[k]$ as an impulse having an area $f[k]$. Then, since the integrand exists only at the sample points:

$$\begin{aligned} F(i\omega) &= \int_0^{(N-1)T} f(t)e^{-i\omega t} dt \\ &= f_0 e^{-i\omega 0} + f_1 e^{i\omega T} + \dots + f_j e^{-i\omega jT} + \dots + f_{N-1} e^{i\omega(N-1)T} \end{aligned}$$

i.e.

$$F(i\omega) = \sum_{j=0}^{N-1} f_j e^{-i\omega kT}$$

For continuous Fourier transform, it could be evaluated over a finite interval (usually the fundamental period T_0) rather than from $-\infty$ to $+\infty$ if the waveform was periodic. Similarly, since there are only a finite number of input data points, the DFT treats the data as if it were periodic (i.e. $f(N)$ to $f(2N - 1)$ is the same as $f(0)$ to $f(N - 1)$.)

Since the operation treats the data as if it were periodic, we evaluate the DFT equation for the fundamental frequency (one cycle per sequence, $\frac{1}{NT}$ Hz, $\frac{2\pi}{NT}$ rad/sec.) and its harmonics. Set:

$$\omega = 0, \frac{2\pi}{NT}, \frac{2\pi}{NT} \times 2, \dots, \frac{2\pi}{NT}, \dots, \frac{2\pi}{NT} \times (N - 1)$$

or, in general

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j e^{-\frac{i \cdot 2\pi jk}{N}} \quad (k = 0 : N - 1)$$

\hat{f}_k is the DFT of the sequence f_k .

We may write this equation in matrix form as:

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \hat{f}_2 \\ \vdots \\ \hat{f}_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W & W^2 & W^3 & \dots & W^{N-1} \\ 1 & W^2 & W^4 & W^6 & \dots & W^{N-2} \\ 1 & W^3 & W^6 & W^9 & \dots & W^{N-3} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \dots \\ 1 & W^{N-1} & W^{N-2} & W^{N-3} & \dots & W \end{bmatrix} = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

where $W = \exp(-i2\pi/N)$ and $W = W^{2N}$ etc.= 1.

3.4 From Discrete Fourier Transform to Fast Fourier Transform

Through DFT we are able to get a summation of sinusoidal and cosinusoidal functions from a group of data points as input signal samples. The sinusoidal and cosinusoidal summations are from the exponential terms by Euler's Formula. The time complexity of DFT is $O(n^2)$, tremendous computations are required when the input contains a large number of data. For example, if $N = 10000$, we may need more than 100 million operations. Although our computers can finish 100 million operations around one second, if we can reduce the computational complexity to $O(n \log_2 n)$ then only about 133000 operations are needed and this can be completed within milliseconds. With such a reduction in computational complexity, we can save thousands of times to get the same result. When we want to do computations of a very large scale, the difference will be even more obvious.

To achieve FFT, first consider the aforementioned expression of DFT's result:

$$\hat{f}_k = \sum_{j=0}^{N-1} f_j e^{-\frac{i \cdot 2\pi jk}{N}}$$

To make things clearer, now we use F and x instead of \hat{f} and f :

$$F_k = \sum_{j=0}^{N-1} x_j \cdot e^{-\frac{i \cdot 2\pi jk}{N}}$$

where k stands for the number of frequency bins and n stands for the number of input signal samples.

First, divide the original summation into two series with even and odd indexes respectively.

$$\begin{aligned}
F_k &= \sum_{j=0}^{N-1} x_j \cdot e^{-\frac{i \cdot 2\pi j k}{N}} = \text{even indexed terms} + \text{odd indexed terms} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-\frac{i \cdot 2\pi k(2m)}{N}} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k(2m+1)}{N}} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-\frac{i \cdot 2\pi k m}{N/2}} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k(m+1/2)}{N/2}}
\end{aligned}$$

Note that, by opening the bracket in the exponential term, the odd-indexed series can be written as

$$\sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k(m+1/2)}{N/2}} = \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k m}{N/2}} \cdot e^{-\frac{i \cdot \pi k}{N/2}} = e^{-\frac{i \cdot \pi k}{N/2}} \cdot \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k m}{N/2}}$$

Denote $e^{-\frac{i \cdot \pi k}{N/2}}$ as c_k , the original summation becomes

$$F_k = \sum_{j=0}^{N-1} x_j \cdot e^{-\frac{i \cdot 2\pi j k}{N}} = \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-\frac{i \cdot 2\pi k m}{N/2}} + c_k \cdot \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-\frac{i \cdot 2\pi k m}{N/2}}$$

Since the periodicity of trigonometric functions

$$\begin{aligned}
\cos\left(-\frac{2\pi k(m+N/2)}{N/2}\right) &= \cos\left(-\frac{2\pi k m}{N/2} - 2\pi k m\right) = \cos\left(-\frac{2\pi k m}{N/2}\right) \\
\text{and } \sin\left(-\frac{2\pi k(m+N/2)}{N/2}\right) &= \sin\left(-\frac{2\pi k m}{N/2} - 2\pi k m\right) = \sin\left(-\frac{2\pi k m}{N/2}\right)
\end{aligned}$$

indicates that

$$e^{-\frac{i \cdot 2\pi k(m+N/2)}{N/2}} = e^{-\frac{i \cdot 2\pi k m}{N/2}}$$

and similarly,

$$c_{k+\frac{N}{2}} = e^{-\frac{i \cdot \pi(k+N/2)}{N/2}} = e^{i\left(-\frac{\pi k}{N/2} - \pi\right)} = -c_k$$

we can conclude that when $k > N/2$, all the terms in summation repeat besides c_k , that is

$$\begin{aligned}
\text{even indexed terms (with } k) &= \text{even indexed terms (with } k + \frac{N}{2}) \\
\text{odd indexed terms (with } k) &= - \text{odd indexed terms (with } k + \frac{N}{2})
\end{aligned}$$

Then, these two series can be split into their own odd and even indexed series until it cannot be split anymore. This asks our input number of data points to be some power of 2, because every time we split the original half-summation series are split into 2 smaller ones. During this process, we are able to reduce the operations needed to a large extent. We do not calculate N times for N points at each of the N F_k s, instead, we calculate n times for each split. Suppose $N = 2^p$, where p is some non-negative number, then the computational complexity is just $Np = N \log_2 N$. With an input of N data points $\{(x_i, y_i)\}_0^{N-1}$, we can achieve an FFT algorithm as the attached pseudo-code (see FFT Pseudo-code in Appendix).

3.5 Extension: Butterfly Diagram

Based on the rationale we have discussed in the previous section, here we want to talk about the actual implementation. The way we derive DFT to FFT by splitting a long sequence into some smaller sequences is called the Cooley-Tukey Algorithm. The Cooley-Tukey Algorithm traces the locations of the elements in the previous DFT matrix and they can be presented in a nice diagram called Butterfly Diagram.

Denote $e^{-\frac{i \cdot 2\pi}{n}}$ as W_n , $W_n^k = e^{-\frac{i \cdot 2\pi k}{n}}$. Recall what we mentioned in the DFT section that we can use W to show the changes in each stage. First, let's look at the most basic transform with 2 input data points. From the previous sections, we have that

$$\begin{aligned} F_0 &= x_0 + W_2^0 x_1 \\ F_1 &= x_0 - W_2^0 x_1 \end{aligned}$$

which can be described in the following butterfly diagram Figure 1. The input data points x_0 , x_1 are on the left and the frequency bins summed of trigonometric functions are on the right. Each arrow carries a computation operation, where the element that arrow starts from is multiplied by the multiplier W_n^k or $-W_n^k$ it reaches. Through these arrows, we can clearly trace the operations during the transform process. The shape of this diagram presents is just like a butterfly.

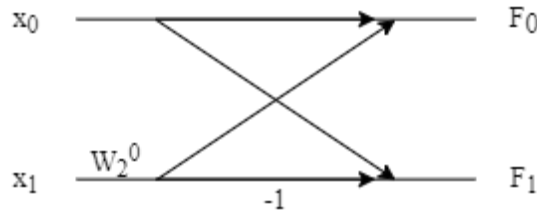


Figure 1: 2-Input Butterfly Diagram

In cases with more input data points, the butterfly diagrams become complex. On the left are the input data points reordered according to their indexes and on the right are the frequency bins. At the first stage, they are composed of pairs of 2-input diagrams, where the upper line represents the multiplier W_2^0 and the lower line represents the multiplier $-W_2^0$. The next stage is formed by taking two non-adjacent lines with one line in between as a new 2-input diagram and use two new 2-input diagrams form a 4-input diagram. The i th stage is formed by taking any two non-adjacent lines with $2^{i-1} - 1$ lines in between as a new 2-input diagram, and use 2^{i-1} new 2-input diagrams to form a 2^i -input diagram. Take an 8-input diagram as an example:

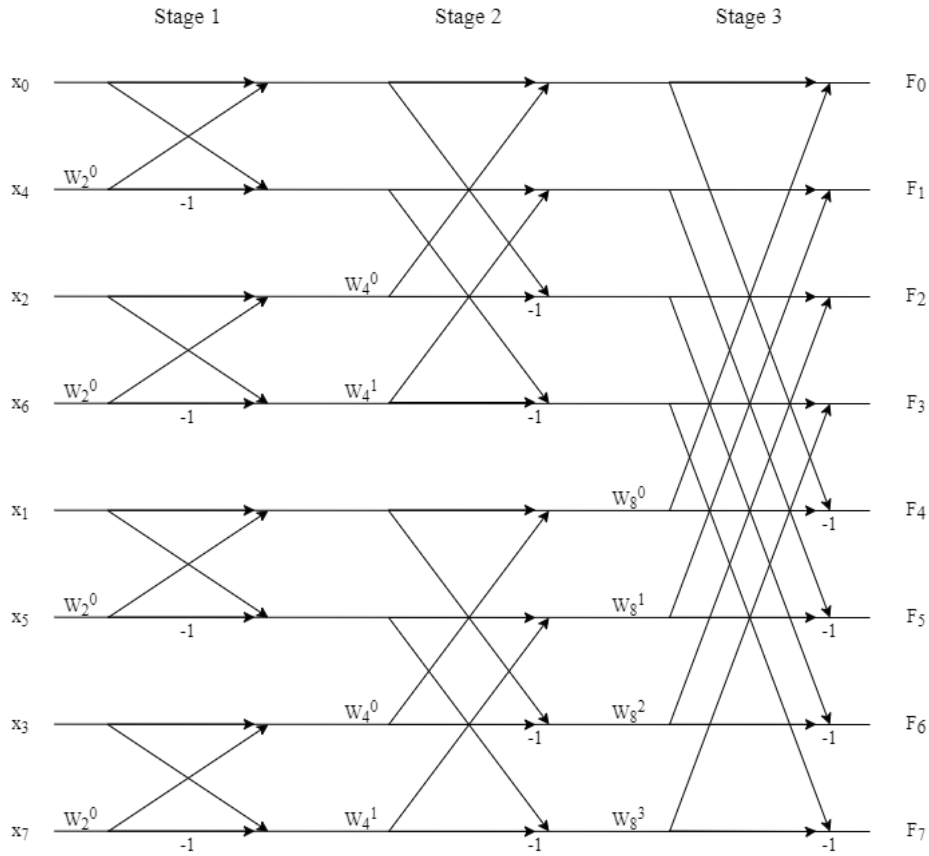


Figure 2: 8-Input Butterfly Diagram

We start from x_0 to x_7 to get the first round of results, then use these results to achieve second-round operations, and at last get to the expression of each F_k .

In the method provided in our textbook, the expressions of coefficients are achieved by directly tracing back from each F_k , so it has many complex loops. However, the butterfly diagram has shown that recursion and iteration are powerful to solve this problem. Take advantage of that, we can code directly by implementing this process, since we can apply similar operations to each group of smaller subsequences during different transform stages. Instead of using the pseudo-code as mentioned in textbook for programming, we will tend to this optimized method with recursions (see FFT Julia code in Appendix).

4 Applications

Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. In 1994, Gilbert Strang described the FFT as "the most important numerical algorithm of our lifetime" [13]. In this section, we will mainly introduce four of the applications of Fast Fourier transform: large polynomial and large integer multiplication, signal decomposition, signal filtering, and image processing.

4.1 Large Polynomial and Large Integer Multiplication

The traditional "left-shifting-and-adding" or the Russian Peasant multiplication takes $O(n^2)$ runtime. However, the FFT-based algorithm can reduce it to $O(n \log n)$ time [3]. Let $X = P(B)$ and $Y = Q(B)$ be two large integers, then they can also be expressed as two polynomials of base $z = B$:

$$P(z) = \sum_{j=0}^{n-1} x_j z^j, Q(z) = \sum_{j=0}^{n-1} y_j z^j$$

To compute PQ in time $O(n \log n)$, first compute FFT of P and Q

$$\begin{aligned} X^* &= (x_0^*, x_1^*, \dots, x_{2n-1}^*) = FFT_{2n}(x_0, x_1, \dots, x_{n-1}, 0, \dots, 0) \\ Y^* &= (y_0^*, y_1^*, \dots, y_{2n-1}^*) = FFT_{2n}(y_0, y_1, \dots, y_{n-1}, 0, \dots, 0) \end{aligned}$$

Then multiply term by term of X^* and Y^* to obtain Z^*

$$Z^* = (z_0^*, z_1^*, \dots, z_{2n-1}^*), \text{ where } z_i^* = x_i^* y_i^*$$

Compute the inverse Fourier transform to obtain the final answer Z

$$Z = (z_0, z_1, \dots, z_{2n-1}), \text{ where } z_i = \frac{1}{2n} \sum_{k=0}^{2n-1} z_k^* w_{2n}^{-ki}$$

4.2 Signal Decomposition

As is introduced before, FFT decomposes an input signal to trigonometric functions of different frequencies. To implement this process, take the following function as an example:

$$\sum_{n=1}^5 n \sin(n\omega t), \quad \omega = 10 \times 2\pi$$

The signal is shown in Fig. 3(a), and the spectrum of FFT is shown in Fig. 3(b). It is apparent that frequency 10, 20, 30, 40, 50 Hz has amplitudes much larger than others.

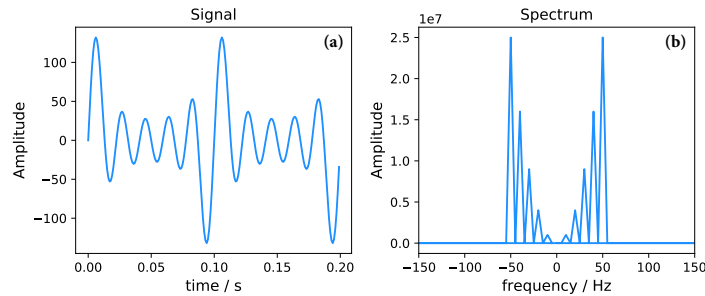


Figure 3: FFT of Sine Summation Function

FFT decomposing is important because it allows signals to be examined one sample at a time. Likewise, systems are characterized by how they respond to impulses. By understanding how the system responds to impulses, the output of the system can be calculated for any given input.

4.3 Signal Filtering

A Fourier filter is a filtering function based on the manipulation of specific frequency components of the signal. It works by performing Fourier transform on the signal, then attenuating or amplifying a specific frequency, and finally inversely transforming the result [3]. In many scientific measurements, such as spectroscopy and chromatography, the signal is a relatively smooth shape, which can be represented by a much smaller number of Fourier components [14].

If we add a random noise of amplitude 20 to the signal in section 5.2, we can get the signal in Fig. 4(a). After performing FFT on it, we can see that there appear some high-frequency components in the spectrum (Fig. 4(b)):

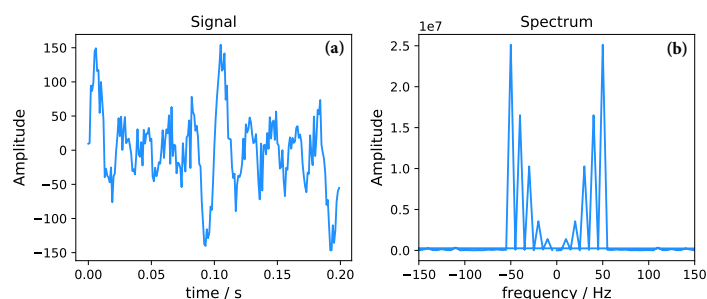


Figure 4: FFT of Signal with Noise

After simply removing those high frequency components and perform inverse Fourier Transform, we can get a clean filtered signal:

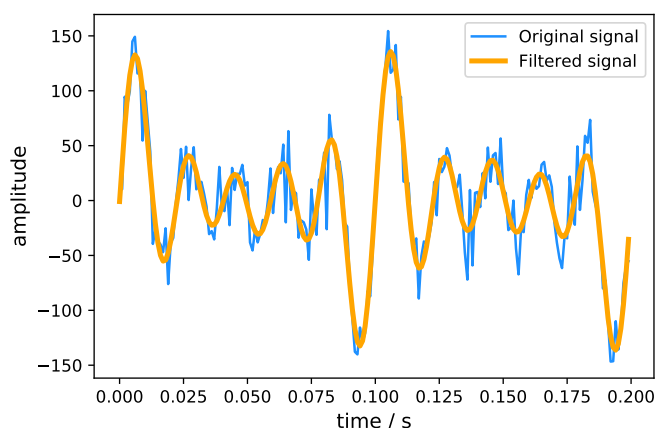


Figure 5: Filtered Signal

This is actually not an optimal way of creating a filter because such a brutal cut-off in frequency space does not control distortion on the signal [15]. However, this can give us an intuition into how signal filtering works to remove noise.

4.4 Image Processing

The Fourier transform can be generalized to higher dimensions. For example, many signals $f(x, y)$ are functions of 2D space defined over an x-y plane. The spectrum of a periodic and discrete 2-D signal [16] can be expressed as

$$F[k, l] = \frac{1}{\sqrt{MN}} \sum_{n=0}^{N-1} \sum_{m=0}^{M-1} f[m, n] e^{-j2\pi(\frac{mk}{M} + \frac{nl}{N})} \quad (1)$$

where k and l are spatial frequencies in x and y directions, $M =$ and N are the numbers of samples in x and y directions in both spatial and spatial frequency domains, respectively. Gray-scale images are typical examples of 2D signals. Here we input a gray-scale DKU logo to 2-dimensional FFT and get the results in Fig. 6.

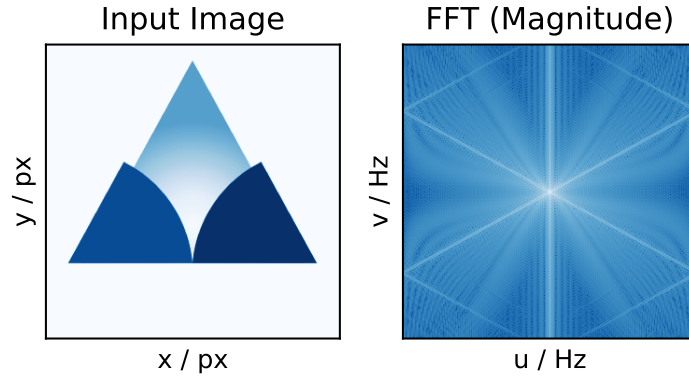


Figure 6: FFT of DKU Logo

Same as one-dimensional FFT, the spectrum will peak at spatial frequencies of repeated texture. In image processing, Fourier transform is used to access the geometric features of the spatial domain image. Since the image in the Fourier domain is decomposed into sinusoidal components, it is easy to check or process certain frequencies of the image, which affects the geometric structure in the spatial domain [16].

5 Conclusion

Fast Fourier transform offers us a powerful method to compute discrete Fourier transform, helping in converting the time domain into the frequency domain in high efficiency. To derive FFT from DFT, the idea is to break down a large DFT sequence of size 2^p into many smaller ones of size 2^q ($q \leq p$), according to their indexes. To better visualize this process, we can create a butterfly diagram, in which the small butterflies(the 2-input diagrams) form portions of computation at each stage. Based on the analysis of the butterfly diagram, we adopted a recursive coding method which allows us to achieve the same results in fewer lines of code.

Fast Fourier transforms are widely used for applications in engineering, music, science, and mathematics. In this article, we mainly introduce four applications: large polynomial multiplication, signal decomposition, signal filtering, and image processing. For each of the applications, we explain the intuition behind and some practical examples are also given.

Till now, the number of input data N is required to be some power of 2 (i.e. $2 \bmod N = 0$), so that its DFT sequence can be split. Thus, we believe that future research could be done to investigate more situations when the number of input data points N is not some power of 2.

References

- [1] I. N. Sneddon, Use of Integral Transforms. New York: McGraw-Hill, 1972.
- [2] gandalf61. Motivation on Using Fourier Series to Solve Heat Equation.
<https://math.stackexchange.com/q/2898235>
- [3] K. L. Stratos. The Fast Fourier Transform and Its Applications.
<http://www.cs.columbia.edu/~stratos/research/fft.pdf>
- [4] S. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, 1999.
- [5] J. Cooley and J. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathetics of Computation*, **19**(90):297–301, 1965.
- [6] J. Dongarra and F. Sullivan. The top ten algorithms of the century. *Computing in Science and Engineering*, 2000.
- [7] F. Bottin, S. Leroux, A. Knyazev, and G. Zérah. Large-scale ab initio calculations based on three levels of parallelization. **Computational Materials Science**, **42**(2), 329-336, 2008.
- [8] M. C. Pease. An adaptation of the fast Fourier transform for parallel processing. *Journal of the ACM*, **15**(2), 252-264, 1968.
- [9] G. Bergland. Fast Fourier transform hardware implementations—An overview. *IEEE Transactions on Audio and Electroacoustics*, **17**(2), 104-108, 1969.
- [10] G. Bergland. A parallel implementation of the fast Fourier transform algorithm. *IEEE Transactions on Computers*, **100**(4), 366-370, 1972
- [11] R. Singleton. An algorithm for computing the mixed radix fast fourier transform. *IEEE Transactions on Audio and Electroacoustics*, **17**(2):93–103, 1969.
- [12] R. Yavne. An economical method for calculating the discrete fourier transform. *American Federation of information Processing Societies Joint Computer Conference*, 115–125, 1968.
- [13] G. Strang (May–June 1994). Wavelets. *American Scientist*. **82** (3): 250–255. JSTOR 29775194
- [14] Fourier Filter.
<https://terpconnect.umd.edu/~toh/spectrum/FourierFilter.html>
- [15] Sphinx-Gallery. Plotting and manipulating FFTs for filtering.
https://scipy-lectures.org/intro/scipy/auto_examples/plot_fftpack.html
- [16] Two-dimensional Fourier Transform.
http://fourier.eng.hmc.edu/e101/lectures/Image_Processing/node6.html

6 Appendix

FFT Pesudo-code

```
1: function FFT( $m, p, y_0, y_1, \dots, y_{2m-1}$ )
2:   Set  $M = m, q = p, \zeta = e^{\frac{\pi i}{m}}$ 
3:   for  $j = 0, 1, \dots, 2m - 1$  do
4:      $c_j = y_j$ 
5:   end for
6:   for  $j = 1, 2, \dots, M$  do
7:      $\xi_j = \zeta^j, \xi_{j+m} = -\xi_j$ 
8:   end for
9:   Set  $K = 0, \xi_0 = 1$ 
10:  for  $L = 1, 2, \dots, p + 1$  do
11:    while  $K < 2m - 1$  do
12:      for  $j = 1, 2, \dots, M$  do
13:         $K = k_p \cdot 2^p + k_{p-1} \cdot 2^{p-1} + \dots + k_1 \cdot 2 + k_0$ 
14:        Set  $K_1 = \frac{K}{2^q} = k_p \cdot 2^{p-q} + \dots + k_{q+1} \cdot 2 + k_q$ 
15:        Set  $K_2 = k_q \cdot 2^p + k_{q+1} \cdot 2^{p-1} + \dots + k_p \cdot 2^q$ 
16:        Set  $\eta = c_{K+M} \xi_{K_2}$ 
17:        Set  $c_{K+M} = c_K - \eta$ 
18:        Set  $c_K = c_K + \eta$ 
19:        Set  $K = K + 1$ 
20:      end for
21:      Set  $K = K + M$ 
22:    end while
23:    Set  $K = 0, M = \frac{M}{2}, q = q - 1$ 
24:  end for
25:  while  $K < 2m - 1$  do
26:     $K = k_p \cdot 2^p + k_{p-1} \cdot 2^{p-1} + \dots + k_1 \cdot 2 + k_0$ 
27:    Set  $j = k_0 \cdot 2^p + k_1 \cdot 2^{p-1} + \dots + k_{p-1} \cdot 2 + k_p$ 
28:    if  $j > K$  then
29:       $temp = c_j, c_j = c_k, c_k = temp$ 
30:       $K = K + 1$ 
31:    Set  $a_0 = c_0/m, a_m = Re(e^{-i\pi j} c_j/m)$ 
32:    for  $j = 1, \dots, m-1$  do Set  $a_j = Re(e^{-i\pi j} c_j/m), b_j = (e^{-i\pi j} c_j/m)$ 
33:    end for
34:  return ( $c_0, \dots, c_{2m-1}, a_0, \dots, a_m, b_1, \dots, b_{m-1}$ )
35:
```

FFT Julia code

fft (generic function with 1 method)

```
• function fft(y)
•     y1 = Any[]; y2 = Any[]
•     n = length(y)
•     if n == 1
•         return y
•     end
•     wn(n) = exp(-2*π*im/n)
•     y_even = fft(y[1:2:end])
•     y_odd = fft(y[2:2:end])
•     w = 1
•     for k in 1:Int(n/2)
•         push!(y1, y_even[k] + w*y_odd[k])
•         push!(y2, y_even[k] - w*y_odd[k])
•         w = w*wn(n)
•     end
•     return vcat(y1,y2)
• end
```

FFT Python code

```
import numpy as np
import cv2
from matplotlib import pyplot as plt

# Seed the random number generator
np.random.seed(1234)
time_step = 0.001

time_vec = np.arange(0, 0.2, time_step)
sig = (10 * np.sin(2 * 10 * np.pi * time_vec) + 20 * np.sin(2 * 20 * np.pi * time_vec)
      + 30 * np.sin(2 * 30 * np.pi * time_vec) + 40 * np.sin(2 * 40 * np.pi *
      time_vec)
      + 50 * np.sin(2 * 50 * np.pi * time_vec) + 20 * np.random.randn(time_vec.size))

# The FFT of the signal
sig_fft = fftpack.fft(sig)

# And the power (sig_fft is of complex dtype)
power = np.abs(sig_fft)**2

# The corresponding frequencies
sample_freq = fftpack.fftfreq(sig.size, d=time_step)

# Find the peak frequency: we can focus on only the positive frequencies
pos_mask = np.where(sample_freq > 0)
freqs = sample_freq[pos_mask]
peak_freq = freqs[power[pos_mask].argmax()]

# Check that it does indeed correspond to the frequency that we generate
# the signal with
np.allclose(peak_freq, 1./period)

plt.subplots(figsize=(8,3.5))
plt.subplot(1,2,1)
plt.plot(time_vec, sig, c='dodgerblue', label='Original signal')
plt.xlabel('time / s',fontsize=12)
plt.ylabel('Amplitude',fontsize=12)
plt.title('Signal')
plt.subplot(1,2,2)
plt.plot(sample_freq, power, c='dodgerblue')
plt.xlim(-150,150)
plt.title('Spectrum')
plt.xlabel('frequency / Hz',fontsize=12)
plt.ylabel('Amplitude',fontsize=12)
plt.tight_layout()

ksize = 50
ksize = ksize*2+1
sigma = 15
fil = cv2.imread('dku.png')
fil = cv2.cvtColor(fil, cv2.COLOR_BGR2GRAY)
# fil = cv2.getGaussianKernel(ksize,0)
# fil = fil * fil.T

fil_fft = cv2.dft(np.float32(fil),flags = cv2.DFT_COMPLEX_OUTPUT)
```

```
fil_shift = np.fft.fftshift(fil_fft)

magnitude_fil = 20*np.log(cv2.magnitude(fil_shift[:, :, 0], fil_shift[:, :, 1]))

plt.subplot(131), plt.imshow(fil, cmap='Blues_r')
plt.xlabel('x / px'), plt.ylabel('y / px')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(132), plt.imshow(magnitude_fil, cmap='Blues_r')
plt.xlabel('u / Hz'), plt.ylabel('v / Hz')
plt.title('FFT (Magnitude)'), plt.xticks([]), plt.yticks([])
plt.tight_layout()
```