Assignment 1: Fast Fourier Transform

Junjia Liu[1]

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

November 3, 2018

Contents

1 Fourier Transform

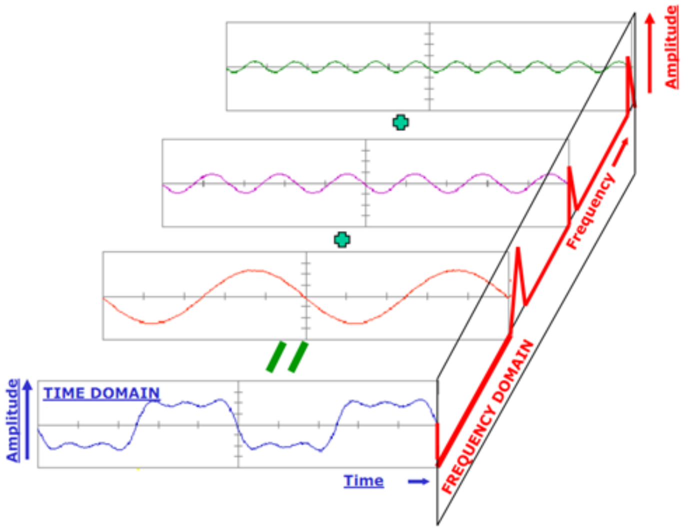
1.1 History of Fourier Transform

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 approximally 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

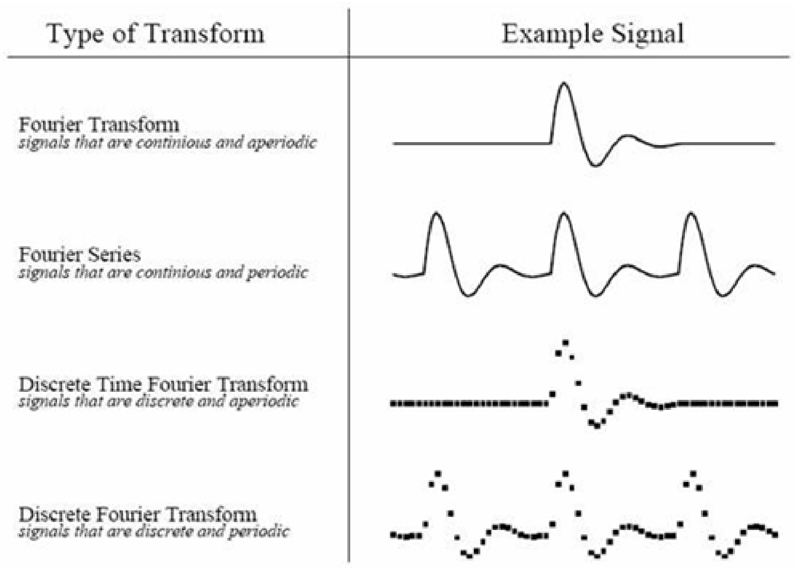
“The FFT is one of the truly great computational developments of this [20 th] century. It has changed the face of science and engineering so much that it is not an exaggeration to say that life as we know it would be very different without the FFT.” (Charles van Loan)

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
signals that are continious and aperiodic
Fourier Series
signals that are continious and periodic
Discrete Time Fourier Transform
signals that are discrete and aperiodic
Discrete Fourier Transform
signals that are discrete and periodic

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:

$$X(k) = \sum_{k=0}^{N-1} x(n) \cdot e^{\frac{-j2\pi nk}{N}}$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{\frac{-j2\pi nk}{N}}$$

For each value of DFT, there are N complex multiplications and $N-1$ complex additions, so if we calculate by using the defination, 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 nth root of unity, where n is a positive integer ($i.e. n = 1, 2, 3, \ldots$), is a number W satisfying the equation

$$W^n = 1$$

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

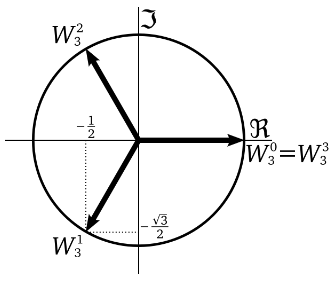


Figure 1: 3rd root of unity

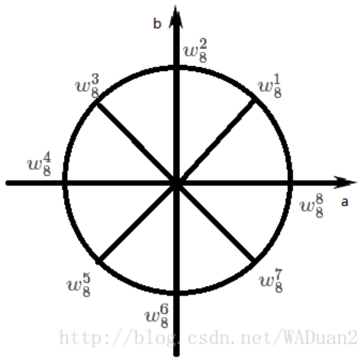


Figure 2: 8th root of unity

Properties of W_N^k:

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

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).

In the case of the radix-2 Cooley–Tukey algorithm, the butterfly is simply a DFT of size-2 that takes two inputs (x0, x1) (corresponding outputs of the two sub-transforms) and gives two outputs (y0, y1) by the formula:

$$y_0 = x_0 + x_1$$

$$y_1 = x_0 - x_1$$

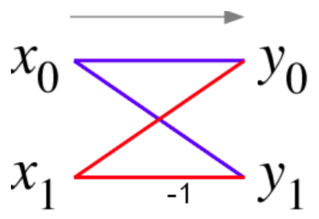


Figure 3: Radix-2 butterfly diagram

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 $W_N^k = e^{-\frac{j2\pi nk}{N}}$ relies on O(nlogn) butterflies of the form:

$$y_0 = x_0 + x_1 W_N^k$$

$$y_1 = x_0 - x_1 W_N^k$$

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 $y_k = \sum_{n=0}^{N-1} W_N^{kn} x_n$ can be divided into two parts,

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

Where $F_{even}(k)$ and $F_{odd}(k)$ is $N/2$-point transformation for the even and odd sequences of $\{x_n\}_0^{N-1}$.[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 $F_{even}(k)$ and $F_{odd}(k)$ are functions with a period of $N/2$,

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

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.

Figure 4 shows an example of the time domain decomposition used in the FFT. In this example, a 16 point signal is decomposed through four separate stages. The first stage breaks the 16 point signal into two signals each consisting of 8 points. The second stage decomposes the data into four signals of 4 points. This pattern continues until there are N signals composed of a single point. An **interlaced decomposition** is used each time a signal is broken in two, that is, the signal is separated into its even and odd numbered samples. [2]

It can be seen that an N-point transformation change into single point require $Log_2 N$ stages. For each stage, there are n points. It easy to find that complexity of this algorithm is $O(n \log n)$.

All in all, the decomposition is just a **reordering** of samples in the signal. Figure 5 shows the achievement of decomposition which using a **bit reversal sorting** algorithm.

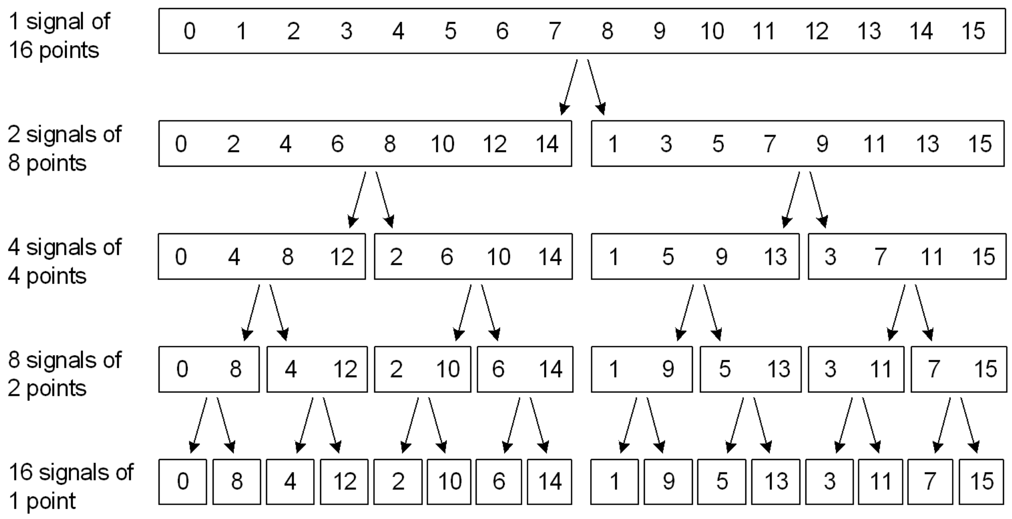


Figure 4: Schematic diagram of FFT with 16-point

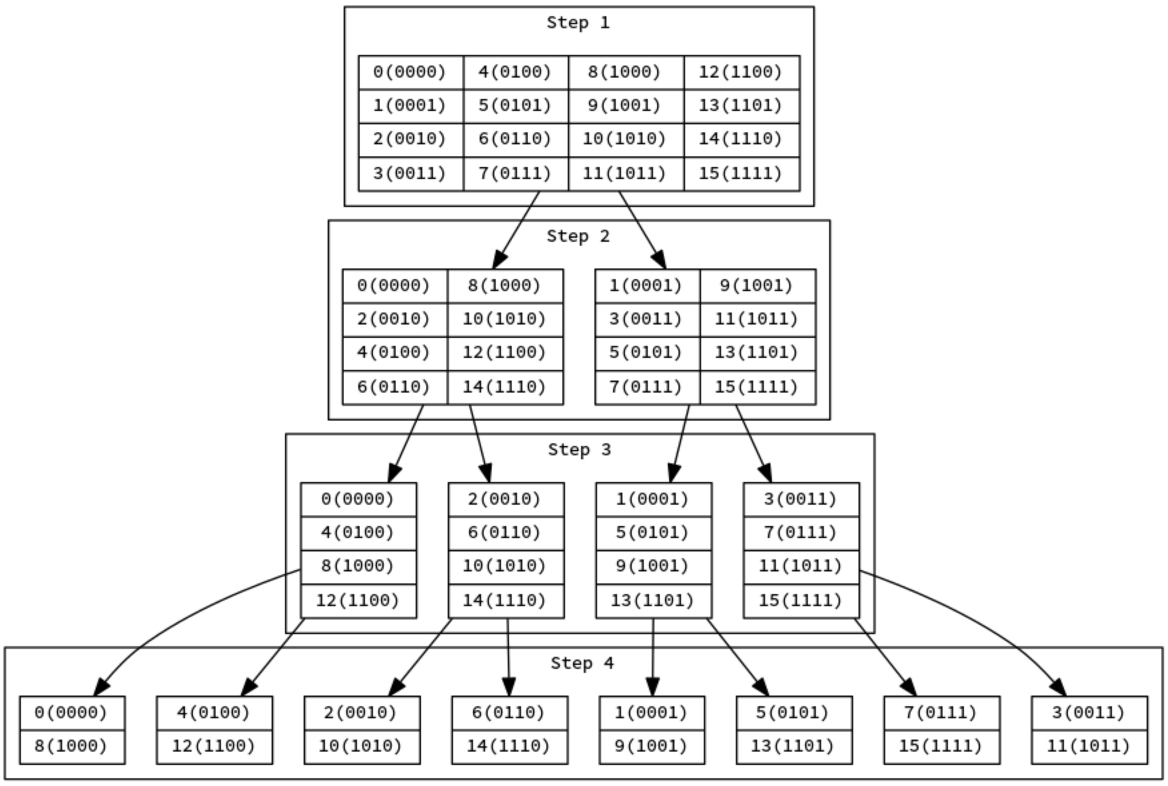


Figure 5: Bit reversal sorting

According to the bit reversal sorting algorithm, FFT algorithm can be implemented by calculating from the resorted samples to original samples. The process is shown in Figure 6.

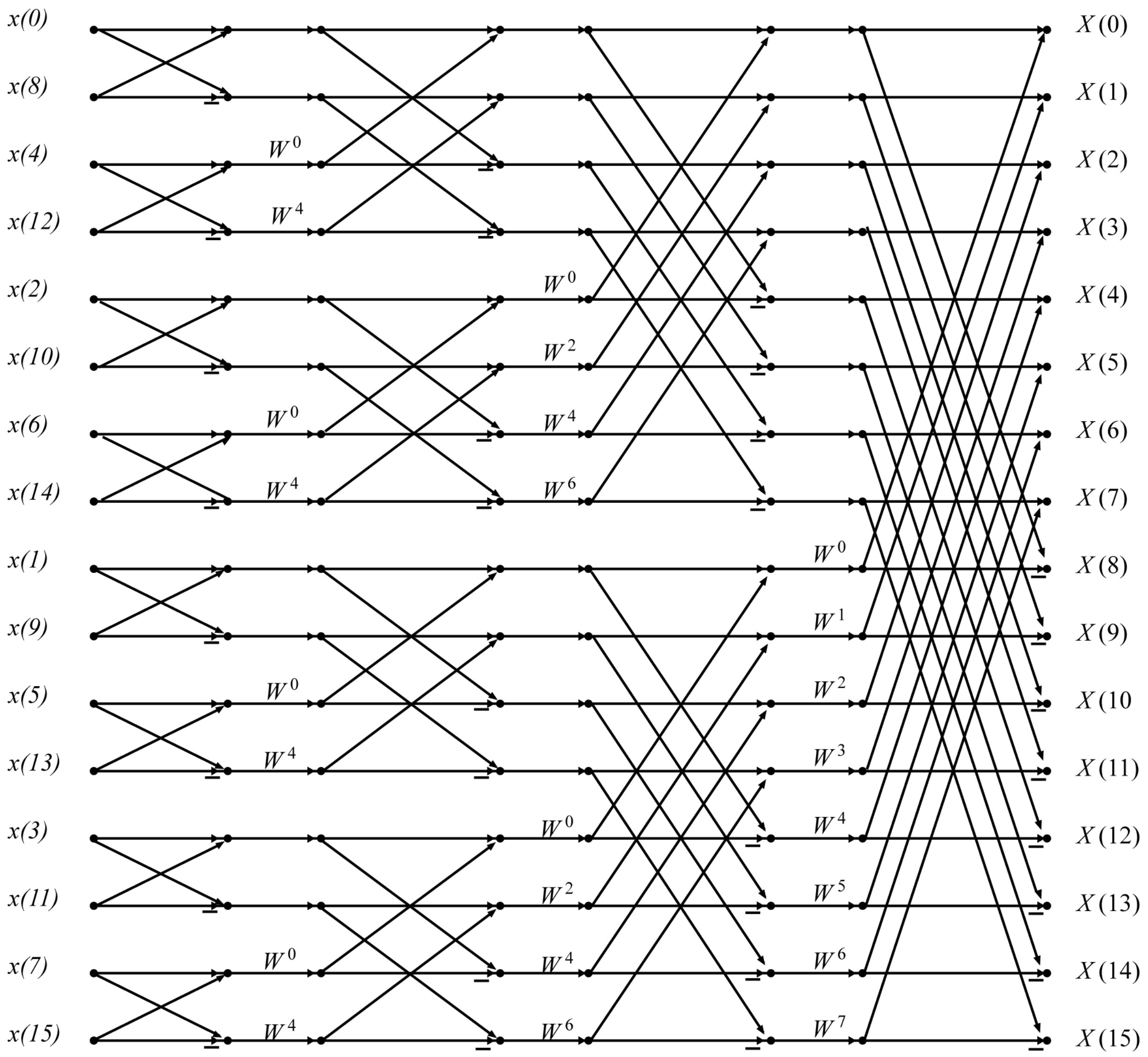


Figure 6: Code implementation of 16-points FFT

2.3 FFT Program

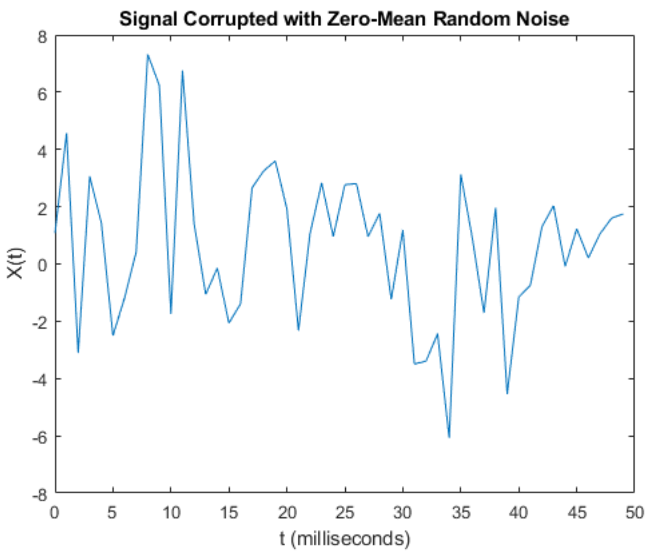
2.3.1 FFT in Matlab

It is easy to achieve FFT algorithm in Matlab.

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

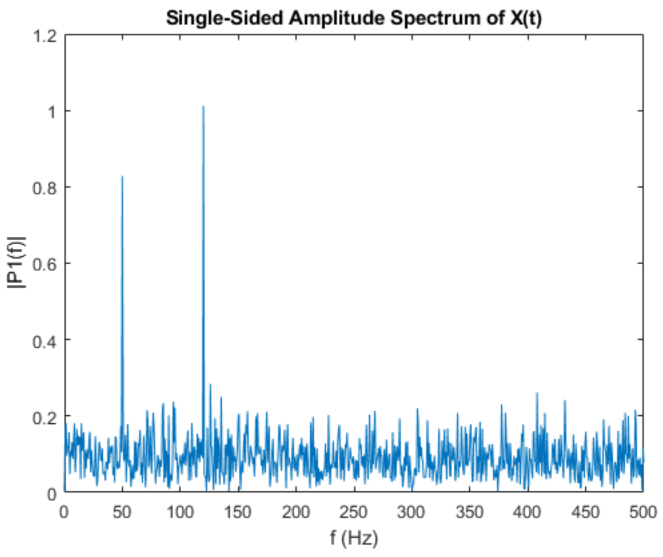
There is a example given by MathWork. Use Fourier transforms to find the frequency components of a signal buried in noise.

```
Fs = 1000;              % Sampling frequency
T = 1/Fs;               % Sampling period
L = 1500;               % Length of signal
t = (0:L-1)*T;          % Time vector

% Form a signal containing a 50 Hz sinusoid of amplitude 0.7
%      and a 120 Hz sinusoid of amplitude 1.
S = 0.7*sin(2*pi*50*t) + sin(2*pi*120*t);

%Corrupt the signal with zero-mean white noise
%          with a variance of 4.
X = S + 2*randn(size(t));

plot(1000*t(1:50),X(1:50))
title('Signal Corrupted with Zero-Mean Random Noise')
xlabel('t(milliseconds)')
ylabel('X(t)')
```



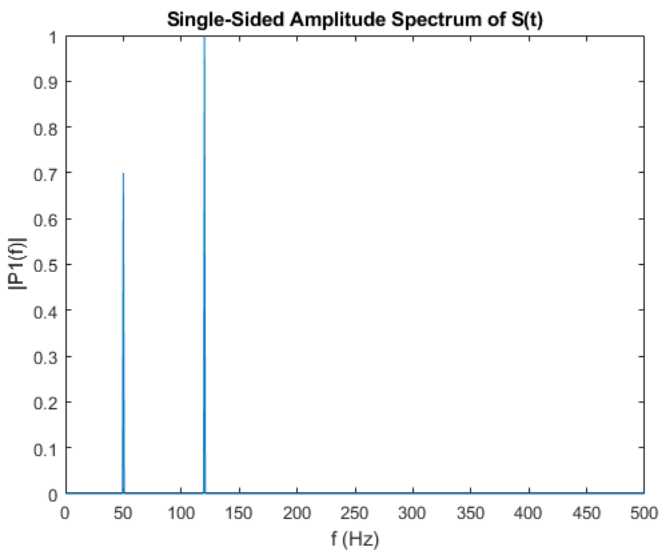
Compute the Fourier transform of the signal.

```
Y = fft(X);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);
f = Fs*(0:(L/2))/L;
plot(f,P1)
title('Single-Sided Amplitude Spectrum of X(t)')
xlabel('f(Hz)')
ylabel('|P1(f)|')
```



Now, take the Fourier transform of the original, uncorrupted signal and retrieve the exact amplitudes, 0.7 and 1.0.

```
Y = fft(S);
P2 = abs(Y/L);
P1 = P2(1:L/2+1);
P1(2:end-1) = 2*P1(2:end-1);

plot(f,P1)
title('Single-Sided Amplitude Spectrum of S(t)')
xlabel('f(Hz)')
ylabel('|P1(f)|')
```



Although Matlab already have the fft function, it is neccessary to achieve fft algorithm by ourselves.

```
function F=my_fft(x)
sz=size(x);
if sz(1)>1&&sz(2)>1||numel(sz)>2
        F=-1;
        return;
end

N=max(sz(1:2));
if N==1
        F=x;
        return;
end
if sz(1)>1
        F=zeros(N,1);
end
if sz(2)>1
        F=zeros(1,N);
end
for k=1:N/2
        [F(k),F(k+N/2)]=my_fft_ele(x,N,k);
end
```

```
function [Fk,Fkn]=my_fft_ele(x,N,k)
    if N==1
        Fk=x;
        Fkn=x;
        return;
    else
        x1=x(1:2:N-1);
        x2=x(2:2:N);
        F1=my_fft_ele(x1,N/2,k);
        F2=my_fft_ele(x2,N/2,k);
        Wkn=exp(-i*2*pi*(k-1)/N);
        Fk=F1+Wkn*F2;
        Fkn=F1-Wkn*F2;
    end
```

2.3.2 FFT in C++

Recursive implementation:

```
void fft(int n, complex<double>* buffer, int offset, int step,
         complex<double>* epsilon)
{
        if(n == 1) return;
        int m = n >> 1;
        fft(m, buffer, offset, step << 1, epsilon);
        fft(m, buffer, offset + step, step << 1, epsilon);
        for(int k = 0; k != m; ++k)
        {
          int pos = 2 * step * k;
          temp[k] = buffer[pos + offset]+ epsilon[k * step]
                        * buffer[pos + offset + step];
          temp[k + m] = buffer[pos + offset] - epsilon[k*step]
                        * buffer[pos + offset + step];
        }

        for(int i = 0; i != n; ++i)
          buffer[i * step + offset] = temp[i];
}

void init_epsilon(int n){
        double pi = acos(-1);
        for(int i = 0; i != n; ++i)
        {
          epsilon[i] = complex<double>(cos(2.0 * pi * i / n),
                              sin(2.0 * pi * i / n));
          arti_epsilon[i] = conj(epsilon[i]);
        }
}
```

Iterative implementation:

```
//Bit reversal sorting
void bit_reverse(int n, complex_t *x) {
        for(int i = 0, j = 0; i != n; ++i)
        {
                if(i > j) swap(x[i], x[j]);
                for(int l = n >> 1; (j ^= l) < l; l >>= 1);
        }
}

//FFT
void transform(int n, complex_t *x, complex_t *w)
{
    bit_reverse(n, x);
    for(int i = 2; i <= n; i <<= 1)
    {
            int m = i >> 1;
            for(int j = 0; j < n; j += i)
            {
                    for(int k = 0; k != m; ++k)
                    {
                            complex_t z = x[j + m + k]*w[n/i*k];
                            x[j + m + k] = x[j + k] - z;
                            x[j + k] += z;
                    }
            }
    }
}
```

2.3.3 Zero padding

There will be a problem when the number of samples is not equal to $2^p, p \in Z$(i.e. the fft program we built by ourselves in section 2.3.1). Becsuse FFT algorithm needs 2^p points so that it can finially separate the DFT into radix-2 Cooley-Tukey algorithm. The solution is padding zero after the incomplete sample set which is used in MATLAB fft function. Thanks to zhanjxcom, we can find that the position we pad zero in the sample set have an influence to the precision of frequency domain.

2.4 Application

Because my major is Robotics, when talking about FFT or FT, Digital Image Processing or Computer Vision must be the first to be occured to me.

2.4.1 Two-dimensional DFT

As we are only concerned with digital images, we will restrict this discussion to the Discrete Fourier Transform (DFT).

For a square image of size $N \times N$, the two-dimensional DFT is given by:

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) e^{-\iota 2\pi (\frac{ki}{N} + \frac{lj}{N})}$$

where $f(a,b)$ is the image in the spatial domain and the exponential term is the basis function corresponding to each point $F(k,l)$ in the Fourier space. The equation can be interpreted as: the value of each point $F(k,l)$ is obtained by multiplying the spatial image with the corresponding base function and summing the result.

In a similar way, the Fourier image can be re-transformed to the spatial domain. The inverse Fourier transform is given by:

$$f(a,b) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{N-1}^{l=0} F(k,l) e^{\iota 2\pi (\frac{ka}{N} + \frac{lb}{N}))}$$

To calculate these formulas, two-dimensional DFT can be separated into two one-dimensional DFT. The spatial domain image is first transformed into an intermediate image using N one-dimensional Fourier Transforms. This intermediate image is then transformed into the final image, again using N one-dimensional Fourier Transforms. FFT is useful to accelerate the calculation.

2.4.2 Application in Digital Image Processing

The Fourier Transform is used to access the geometric characteristics of a spatial domain image. Because the image in the Fourier domain is decomposed into its sinusoidal components, it is easy to examine or process certain frequencies of the image, thus influencing the geometric structure in the spatial domain.

In most implementations the Fourier image is shifted in such a way that the DC-value (i.e. the image mean) $F(0,0)$ is displayed in the center of the image. The further away from the center an image point is, the higher is its corresponding frequency.

We start off by applying the Fourier Transform on the classical Digital Image Processing example —— Lenna.

Figure 7: Lenna

First, change RGB image into grayscale.

```
f=imread('Lenna.jpg');
imshow(f)
f1=rgb2gray(f);
imshow(f1)
```

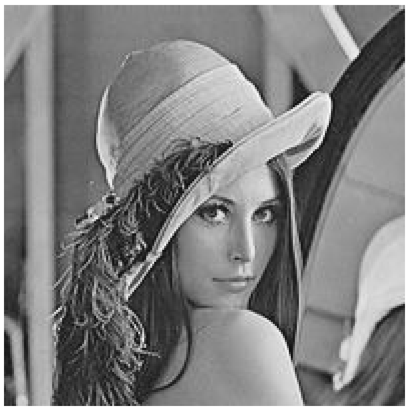
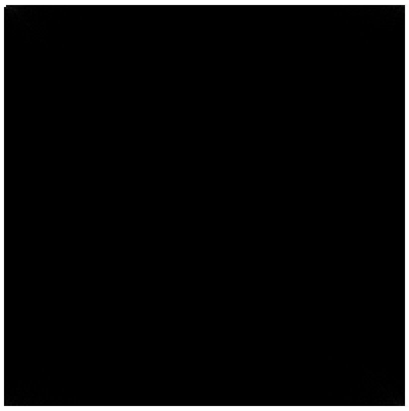


Figure 8: Lenna in grayscale

The magnitude calculated from the complex result is shown in

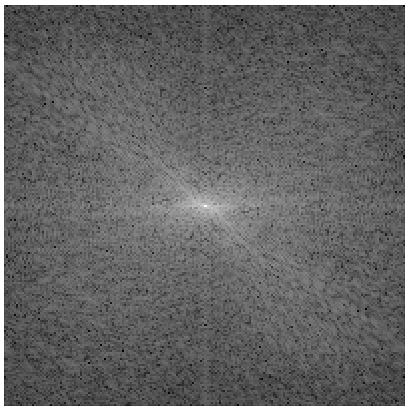
```
F=fft2(f1);
S=abs(F);
imshow(S,  [])
```



We can see that the DC-value is by far the largest component of the image. However, the dynamic range of the Fourier coefficients (i.e. the intensity values in the Fourier image) is too large to be displayed on the screen, therefore all other values appear as black. If we apply a logarithmic transformation to the image we obtain

```
Fc=fftshift(F);
S2=log(1+abs(Fc));
imshow(S2,[])
```

The result shows that the image contains components of all frequencies, but that their magnitude gets smaller for higher frequencies. Hence, low frequencies contain more image in-



formation than the higher ones. The transform image also tells us that there are two dominating directions in the Fourier image, one passing vertically and one horizontally through the center. These originate from the regular patterns in the background of the original image.

Then, we try to use FFT and Gaussian Blur to process this figure.

```
[M,N]=size(f1);
F=fft2(f1);
sig=10;
H=lpfilter('gaussian',M,N,sig);
G=H.*F;
g=real(ifft2(G));
imshow(g,[])
```

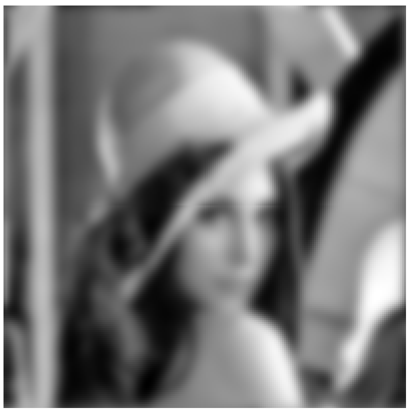


Figure 9: Lenna after Gaussian Blur

Obviously, this is not a perfect processing method. However, it is also not our theme in this article. We have already seen the simple application of FFT in Digital Image Processing.

3 Conculsion

Fourier Transform is a method to transform a signal from time domain into frequency domain. Fast Fourier Transform is a simplify of DFT, which really accelerates the DFT and makes it easier to use. It uses the notions of Root of unit and Butterfly diagram to separate the classical DFT mission into many smaller one and use Bit reversal sorting to reorder the samples of the signal. Besides, Two-dmensional DFT is also used in Digital Image Processing and FFT is used to accelerate this process.

References

[1] Wikipedia contributors. Fast fourier transform — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Fast_Fourier_transform&oldid=864632799, 2018. [Online; accessed 28-October-2018].

[2] Steven W Smith et al. The scientist and engineer's guide to digital signal processing. 1997.