THE FAST FOURIER TRANSFORM

LONG CHEN

ABSTRACT. Fast Fourier transform (FFT) is a fast algorithm to compute the discrete Fourier transform in $\mathcal{O}(N \log N)$ operations for an array of size $N = 2^J$. It is based on the nice property of the principal root of $x^N = 1$. In addition to the recursive implementation, a non-recursive and in-place implementation, known as butterfly algorithm, is also provided.

Fourier series can be written as

$$u(x) = \sum_{j=0}^{\infty} c_k e^{ikx}, \quad \forall x \in [0, 2\pi]. \tag{1}$$

It transforms a periodic function $u(x)$ to an infinite vector $(c_0, c_1, \ldots, c_N, \ldots)$ composed by Fourier coefficients. The independent variable is changed from x to k.

The significance of Fourier transform is that a function is decomposed into composition of frequencies. Some properties are more transparent viewed in the frequency domain. It opens a new way to study functions and becomes the core of the signal processing.

Another nice property of Fourier transform is that the basis e^{ikx} is friendly to the differential operators. Mathematically it is an eigen-function of the differential operator. The differentiation/integration becomes an algebraic multiplication and thus differential equations are easy to solve in this basis. This is known as spectral methods.

Fourier transform, more precisely the discrete Fourier transform, becomes practical only after faster Fourier transform (FFT) is invented which dramatically reduces the $\mathcal{O}(N^2)$ naive implementation to much faster $\mathcal{O}(N \log N)$ algorithms. FFT is one of the most important algorithms of the 20th century. It is discovered by Cooley and Tukey [1] in 1965 but can be traced back to Gauss 160 years earlier.

1. DISCRETE FOURIER TRANSFORM

The function u is periodic with period 2π and thus the variable x can be viewed as the horizontal axis-angle θ of a unit vector; see Fig. 1. In the discrete case, the transform (1) holds on discrete points. Given an integer N, let $\theta = 2\pi/N, \omega = e^{i\theta}$ and let $x_l = l\theta$, for $l = 0, \ldots, N-1$, be N equal-distributed points on the unit circle. We ask the identity (1) holds at x_l, for $l = 0, \ldots, N-1$, which results in a matrix equation $F\boldsymbol{c} = \boldsymbol{u}$

$$\begin{pmatrix} 1 & 1 & 1\ldots & 1 \\ 1 & \omega & \omega^2\ldots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4\ldots & \omega^{2(N-1)} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)}\ldots & \omega^{(N-1)^2} \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \cdots \\ c_{N-1} \end{pmatrix} = \begin{pmatrix} u_0 \\ u_1 \\ u_2 \\ \cdots \\ u_{N-1} \end{pmatrix}.$$

Date: October 8, 2015.

The matrix F is called Fourier matrix and the (l,k) item of matrix F is $e^{ikx_l} = e^{ikl\theta} = \omega^{lk}$. So we can simply write the matrix $F = (\omega^{lk})_{N\times N}$, for $l,k = 0,1,\ldots,N-1$. The matrix F is symmetric (only transpose no conjugate).

Given a function $u(x)$, more precisely u sampled at certain points x_i, represented by a vector $\boldsymbol{u}$, the Fourier coefficients vector $\boldsymbol{c}$ is given by $F^{-1}\boldsymbol{u}$. So we first figure out F^{-1}.

For a fixed N, all roos of $x^N = 1$ forms a finite cyclic group G. A generator is the principle root $\omega = e^{2\pi i/N}$ and the operation is the multiplication of complex numbers which can be visualized as a rotation of multiple of angle θ. $1 = \omega^0$ is the unit element and the inverse of an element $w \in G$ is its conjugate $\bar{w}$ since $w\,\bar{w} = |w| = 1$ for a unit vector w. If $w = \omega^k = e^{ik\theta}$, then $\bar{w} = e^{-ik\theta} = \omega^{-k}$.

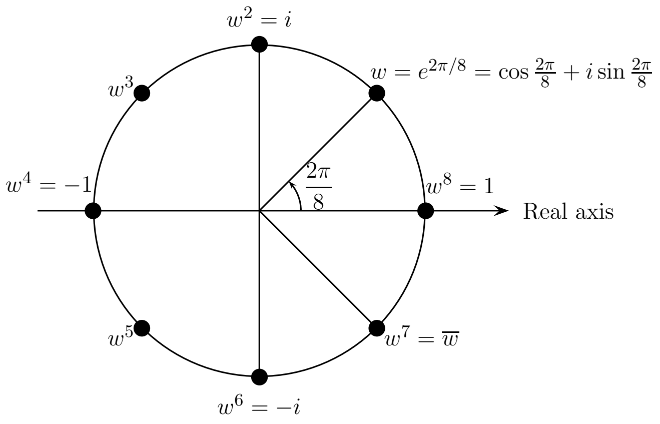

FIGURE 1. default

Lemma 1.1. *The matrix F is orthogonal, i.e., the column vectors of F are mutually orthogonal.*

Proof. Let v_k be the column vectors of F. We compute $v_k \cdot \bar{v}_l$, for $k \neq l$, as

$$1 + W + W^2 + \cdots + W^{N-1} \tag{2}$$

with $W = \omega^k\omega^{-l}$. A key observation is that W is still a root of unity as a power of the generator, i.e., $W^N = \omega^{kN}\omega^{-lN} = 1^k 1^{-l} = 1$. And since $k \neq l$, $W \neq 1$. Then multiply (2) by $1 - W$, we obtain $1 - W^N = 0$ and conclude (2) is zero since $W \neq 1$. □

As a consequence, we obtain the identity $\bar{F}F = NI$ and thus the inverse of F is a scaled $\bar{F}$, i.e. $F^{-1} = \bar{F}/N$. Here we skip the transpose since F is symmetric. Notice that $\bar{F}$ looks just like F: simply change ω to $\bar{\omega}$.

2. FAST FOURIER TRANSFORM

FFT is a fast algorithm for computing Fc or $\bar{F}u$. It is a divide-and-conquer algorithm. To unify the discussion, we consider the computation $y = F_N x$ with $F_N = (w_N^{kj})_{N\times N}$, i.e., for $k = 0,\ldots,N-1$,

$$y_k = \sum_{j=0}^{N-1} w_N^{kj} x_j. \tag{3}$$

Fourier transform corresponds to $w_N = \bar{\omega}_N$ and inverse Fourier transform to $w_N = \omega_N$. Given an even integer N, we use F_N and F_{N_c} with $N_c = N/2$ to denote the transform in different scales.

Algorithm (FFT)

```
(1) Divide $x$ into $x_{\text{even}}$ and $x_{\text{odd}}$.
(2) Compute $y_{\text{even}} = F_{N_c} x_{\text{even}}, \quad y_{\text{odd}} = F_{N_c} x_{\text{odd}}$
(3) Merge $y_{\text{even}}$ and $y_{\text{odd}}$ into $u$.
```

The merge step is not straight-forward. The formulae found by Cooley and Tukey [1] is based on the following properties of the factor w_N which can be verified easily. The key is $w_N^2 = w_{N_c}$.

(1) Symmetry:
$$(w_N^{kj})^* = w_N^{-kj}, \quad w_N^{k(j+N/2)} = -w_N^{kj}.$$
(2) Periodic:
$$w_N^{kj} = w_N^{k(j+N)} = w_N^{(k+N)j}.$$
(3)
$$w_N^{kj} = w_{mN}^{mkj}, \quad w_N^{kj} = w_{N/m}^{kj/m}.$$

Theorem 2.1. *For $k = 0, \dots, N_c - 1$, we have*

$$y(k) = y_{\text{even}}(k) + w_N^k y_{\text{odd}}(k) \tag{4}$$

$$y(k + N_c) = y_{\text{even}}(k) - w_N^k y_{\text{odd}}(k) \tag{5}$$

Proof. We split the sum in the formulae of y_k into even and odd parts:

$$y_k = \sum_{j=0}^{N-1} w_N^{kj} x_j = \sum_{j=0}^{N_c-1} w_N^{k2j} x_{2j} + \sum_{j=0}^{N_c-1} w_N^{k(2j+1)} x_{2j+1}.$$

The first part is simply y_{even} since $w_N^2 = w_{N_c}$. The weight in the second sum can be rewritten as $w_N^{2kj} w_N^k = w_{N_c}^{kj} w_N^k$ and (4) follows. Note that in (4), the index $k = 0, \dots, N_c - 1$ is in the index range of the coarse FT.

We thus need the second formulae (5) for the second half indices. We repeat the sum as

$$y(k + N_c) = \sum_{j=0}^{N-1} w_N^{(k+N_c)j} x_j = \sum_{j=0}^{N_c-1} w_N^{(k+N_c)2j} x_{2j} + \sum_{j=0}^{N_c-1} w_N^{(k+N_c)(2j+1)} x_{2j+1}.$$

The first weight is still $w_{N_c}^{kj}$ as $w_N^{2N_c j} = w_N^{Nj} = 1$. The second weight is $w_{N_c}^{kj} w_N^k w_N^{N_c} = -\omega_{N_c}^{kj} w_N^k$ as $w_N^{N_c} = -1$. Therefore (5) follows. □

A MATLAB code is presented below.

```
function y = fft(x)
N = length(x);
if N == 1   % the coarsest level
    y = x;     return;
end
% divide
y_even = fft(x(1:2:end));
y_odd = fft(x(2:2:end));
% merge
```

```
10  w = exp(-i*2*pi/N);
11  Nc = N/2;
12  k = 1:Nc;
13  tempy_odd = w.^(k-1).*y_odd;
14  y(k) = y_even + tempy_odd;
15  y(j+Nc) = y_even - tempy_odd;
```

The dominant operation in the merge step is the multiplication of complex numbers, i.e. line 13. We thus skip the addition and count the number of multiplication only. We can easily get the recurrence

$$T(N) = 2T(N/2) + N/2$$

from which we obtain $T(N) = 1/2\, N \log_2 N$.

3. Butterfly Algorithm

We discuss a non-recursive and in-place implementation of FFT. The top-to-bottom phase is an even-odd reordering of the input array. An example of $N = 8$ is displayed in the following table.

TABLE 1. Even-odd ordering of different levels

L = 4	0	1	2	3	4	5	6	7
L = 3	0	2	4	6	1	3	5	7
L = 2	0	4	2	6	1	5	3	7
L = 1	0	4	2	6	1	5	3	7

This ordering can be generated by bit reversal permutation. For an index n, written in binary with digits $b_2b_1b_0$ is switched with the index with reversed digits $b_0b_1b_2$; see the figure for an illustration.

0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

FIGURE 2. Bit reversal ordering for $N = 8$.

A MATLAB implementation is presented below.

```
for i = 0:N-1
    ir = bin2dec(flip(dec2bin(i,J)));
    if i<ir
        % all index shifted by one since it starts with 1 not 0
        t = x(i+1);
        x(i+1) = x(ir+1);
```

```
            x(ir+1) = t;
        end
    end
```

The bottom-to-top can be decomposed into pairs of the following butterfly diagram. The weight w_L^j is called twiddle factor. This provides a in-place implementation of FFT.

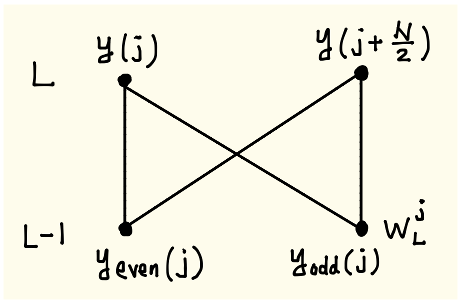

FIGURE 3. Butterfly diagram

Namely we can use the same array to store the updated values. Suppose $N = 2^L$. We use subscript $L, L = 1 : J$, to denote a level index. The two input data of the butterfly diagram is separated by the length $d_L = 2^{L-1}$. The twiddle factor in level L is $w_L = w_{2^L} = w_N^{2^{J-L}}$. The updated formulae is

$$y_L(k) = y_{L-1}(k) + w_L^k y_{L-1}(k + d_L)$$
$$y_L(k + d_L) = y_{L-1}(k) - w_L^k y_{L-1}(k + d_L).$$

We can first copy values $y_{L-1}(k), y_{L-1}(k+d_L)$ to two temporary location and then rewrite with the values of $y_L(k), y_L(k + d_L)$. So only one array of size N is needed. The $N = 8$ case is illustrated in the following figure. Note that the bottom (left) is in the bit-reverse ordering and the top (right) is the natural ordering.

A MATLAB code is presented below.

```
for L = 1:level
    d = 2^(L-1);
    for j = 0:d-1
        p = j*2^(level - L);
        wL = w^p;
        for k = j+1:2^L:N
            a = x(k);
            b = wL*x(k+d);
            x(k) = a + b;
            x(k+d) = a - b;
        end
    end
end
```

4. VARIANTS OF FFT

In the signal process, the input array $x(t)$ is a signal sampled at certain time. The output is in the frequency domain. The FFT presented in the previous section is called decimation-in-time (DIT).

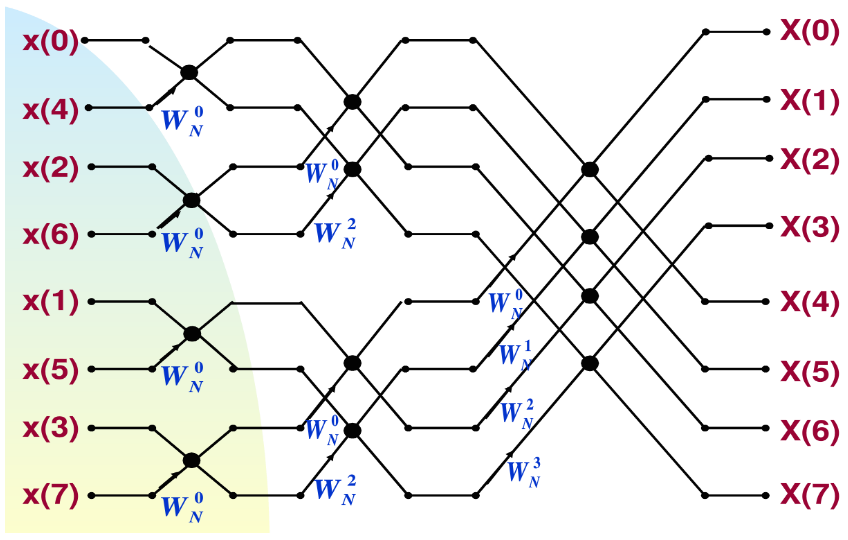

FIGURE 4. Butterfly implementation of FFT

Now we discuss the decimation-in-frequency (DIF) version. The divide step is easier. We simply split the input array $x(0:N-1)$ into two $x_1 = x(0:N_c-1)$ and $x_2 = x(N_c : N-1)$. Then apply the 'butterfly' procedure to x_1 and x_2 to get $\tilde{x}_1$ and $\tilde{x}_2$. After that we apply FFT to shorter arrays $\tilde{x}_1$ and $\tilde{x}_2$ to obtain y_1 and y_2. To merge y_1 and y_2, we need even-odd splitting of y: $y_{\text{even}} = y(0:2:N-2), y_{\text{odd}} = y(1:2:N-1)$.

Exercise 4.1. (1) Derive the formulae from x_1, x_2 to y_{even} and y_{odd}.
(2) Write a pseudocode for DIF (recursive and non-recursive).

When the length N is not of power 2, we can simply add zeros to enlarge the length. We now discuss another variation of FFT adapt to the general factorization $N = N_1N_2$ and usually N_1 is small and called the radix (say between 2 to 8). The FFT presented in the previous section is known radix-2 DIT.

The idea is to reinterpret the 1-D array of length N_1N_2 to a two dimensional matrix of size $N_2 \times N_1$ and then apply the Fourier transform to N_1 arrays of shorter length N_2. The single summation index j is changed to two subscripts (j_1, j_2) and the relation is $j = j_2N_1 + j_1$ for $j_1 = 0, \ldots, N_1, j_2 = 0, \ldots, N_2$. We can split the sum as

$$\begin{aligned}
y(k) = \sum_{j=0}^{N-1} w_N^{kj} x(j) &= \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} w_N^{k(j_1,j_2)} x(j_1, j_2) \\
&= \sum_{j_1=0}^{N_1-1} \sum_{j_2=0}^{N_2-1} w_{N_1N_2}^{kj_2N_1+kj_1} x_{j_1}(j_2) \\
&= \sum_{j_1=0}^{N_1-1} w_N^{kj_1} \sum_{j_2=0}^{N_2-1} w_{N_2}^{kj_2} x_{j_1}(j_2) \\
&= \sum_{j_1=0}^{N_1-1} w_N^{kj_1} \tilde{y}_{j_1}(k).
\end{aligned}$$

Here $\tilde{y}$ is a matrix of size $N_1 \times N_2$ by applying FT to columns of the x interpreted as $N_2 \times N_1$ matrix and then taking transpose.

We further apply the same trick to the index k but in a transpose way, i.e., $k = k_1 N_2 + k_2$. We continue as

$$\begin{aligned} y(k_1, k_2) &= \sum_{j_1=0}^{N_1-1} w_N^{(k_1,k_2)j_1} \tilde{y}_{j_1}(k_1, k_2) \\ &= \sum_{j_1=0}^{N_1-1} w_{N_1 N_2}^{(k_1 N_2 + k_2)j_1} \tilde{y}_{j_1}(k_1, k_2) \\ &= \sum_{j_1=0}^{N_1-1} w_{N_1}^{k_1 j_1} w_N^{k_2 j_1} \tilde{y}_{j_1}(k_1, k_2) \\ &= \sum_{j_1=0}^{N_1-1} w_{N_1}^{k_1 j_1} \tilde{\tilde{y}}_{j_1}(k_1, k_2) \end{aligned}$$

Here $\tilde{\tilde{y}}$ is obtained from $\tilde{y}$ by multiplying N twiddle factors. The last sum is then a Fourier transform of $\tilde{\tilde{y}}$.

So far the sampling is uniform or equivalently the grid of the unit circle is of equidistance. We refer to [2] for FFT on nonuniform sampling. A comprehensive treatment of FFTs can be found in [3].

Exercise 4.2. (1) Use Fourier matrix to find out eigenvectors and eigenvalues of the $N \times N$ circulant matrix

$$C = \begin{pmatrix} c_0 & c_{N-1} & \cdots & c_2 & c_1 \\ c_1 & c_0 & c_{N-1} & \cdots & c_2 \\ \cdots & \cdots & \cdots & & \\ c_{N-2} & \cdots & \cdots & c_0 & c_{N-1} \\ c_{N-1} & c_{N-2} & \cdots & \cdots & c_0 \end{pmatrix}.$$

(2) Design an $\mathcal{O}(N \log N)$ algorithm to compute Cx or $C^{-1}x$.

References

[1] J. W. Cooley and J. W. Tukey. An Algorithm for the Machine Computation of the Complex Fourier Series. *Mathematics of Computation*, 19:297, 1965.

[2] L. Greengard and J.-Y. Lee. Accelerating the Nonuniform Fast Fourier Transform, 2004.

[3] C. Van Loan. *Computational frameworks for the fast Fourier transform*. SIAM, 1992.