1.4 Fast Fourier Transform (FFT) Algorithm

Fast Fourier Transform, or FFT, is any algorithm for computing the \( N \)-point DFT with a computational complexity of \( O(N \log N) \). It is \textit{not} a new transform, but simply an efficient method of calculating the DFT of \( x(n) \).

If we assume that \( N \) is even, we can write the \( N \)-point DFT of \( x(n) \) as

\[
X^{(N)}(k) = \sum_{n \text{ is even: } n=2m, m=0, \cdots, \frac{N}{2}-1} x(n)e^{-j\frac{2\pi k}{N}n} + \sum_{n \text{ is odd: } n=2l+1, l=0, \cdots, \frac{N}{2}-1} x(n)e^{-j\frac{2\pi k}{N}n}
\]

\[
= \sum_{m=0}^{\frac{N}{2}-1} x(2m)e^{-j\frac{2\pi k}{N}2m} + \sum_{l=0}^{\frac{N}{2}-1} x(2l+1)e^{-j\frac{2\pi k}{N}(2l+1)} \tag{1.31}
\]

We make the following substitutions:

\[
x_0(m) = x(2m), \text{ where } m = 0, \cdots, \frac{N}{2} - 1,
\]

\[
x_1(l) = x(2l+1), \text{ where } l = 0, \cdots, \frac{N}{2} - 1.
\]

Rewriting Eq. (1.31), we get

\[
X^{(N)}(k) = \sum_{m=0}^{\frac{N}{2}-1} x_0(m)e^{-j\frac{2\pi k}{N}m} + e^{-j\frac{2\pi k}{N}} \sum_{l=0}^{\frac{N}{2}-1} x_1(l)e^{-j\frac{2\pi k}{N}l}
\]

\[
= X_0^{(\frac{N}{2})}(k) + e^{-j\frac{2\pi k}{N}} X_1^{(\frac{N}{2})}(k), \tag{1.32}
\]

where \( X_0^{(\frac{N}{2})}(k) \) is the \( \frac{N}{2} \)-point DFT of the even-numbered samples of \( x(n) \) and \( X_1^{(\frac{N}{2})}(k) \) is the \( \frac{N}{2} \)-point DFT of the odd-numbered samples of \( x(n) \). Note that both of them are \( \frac{N}{2} \)-periodic discrete-time functions.

We have the following algorithm to compute \( X^{(N)}(k) \) for \( k = 0, \cdots, (N-1) \) :

1. Compute \( X_0^{(\frac{N}{2})}(k) \) for \( k = 0, \cdots, \frac{N}{2} - 1 \).
2. Compute \( X_1^{(\frac{N}{2})}(k) \) for \( k = 0, \cdots, \frac{N}{2} - 1 \).
3. Perform the computation (1.32) with \( N \) complex multiplications and \( N \) complex additions.

Actually, it is possible to use fewer than \( N \) complex multiplications. Let

\[
W_N = e^{-j\frac{2\pi}{N}}.
\]
Sec. 1.4. Fast Fourier Transform (FFT) Algorithm

The following remarks apply to the FFT:

\[ X(0) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N} = x(0) \]

Therefore,

\[ X_N(k) = X_2(k) + W_N^k X_2(k) \]

for \( k = 0, \ldots, N/2 - 1 \),

\[ X_N(k) = X_2(k) - W_N^k X_2(k) \]

for \( k = N/2 \), \( \ldots, N - 1 \).

Then

\[ W_N^k = e^{-j2\pi k/N} = \begin{cases} 1 & \text{if } k \equiv 0 \pmod{N} \\ e^{-j2\pi k/N} & \text{otherwise} \end{cases} \]

Fig. 1.36. The FFT algorithm.

Figure 1.37 illustrates the recursive implementation of the FFT supposing that \( N = 2^M \). There is a total of \( M = \log_2 N \) stages of computation, each requiring \( 3N/2 \) complex operations. Hence, the total computational complexity is \( O(N \log N) \). We see that the process ends at a 1-point DFT.

The following remarks apply to the FFT:

Actually, slightly fewer if we do not count multiplications by \( \pm 1 \) and \( \pm j \).
**Figure 1.37.** The recursive implementation of the FFT supposing that $N = 2^M$. There is a total of $M = \log_2 N$ stages of computation, each requiring $\frac{3}{2}N$ complex operations. Hence, the total computational complexity is $O(N \log N)$.

1. For large $N$, the FFT is much faster than the direct application of the definition of DFT, which is of complexity $O(N^2)$.

2. The particular implementation of the FFT described above is called *decimation-in-time radix-2 FFT*.

3. The number of operations required by an FFT algorithm can be approximated as $CN \log N$, where $C$ is a constant. There are many variations of FFT aimed at reducing this constant—e.g., if $N = 3^M$, it may be better to use a radix-3 FFT.

4. Note that

$$\left\{ \frac{1}{N} \text{DFT}[x^*(n)] \right\}^* = \left\{ \frac{1}{N} \sum_{n=0}^{N-1} x^*(n)e^{-j(\frac{2\pi k}{N})n} \right\}^* = \frac{1}{N} \sum_{n=0}^{N-1} x(n)e^{j(\frac{2\pi k}{N})n}$$

which is the IDFT of $x(n)$. Thus, the FFT can also be used to compute the IDFT.
Example 1.26. The 8-point FFT is depicted in Fig. 1.38. The values of the twiddle factors are:

\[
W_2 = e^{-j\frac{2\pi}{2}} = -1,
\]

\[
W_4 = e^{-j\frac{2\pi}{4}} = -j,
\]

\[
W_8 = e^{-j\frac{2\pi}{8}}.
\]
Figure 1.39. The FFT reduces the number of operations required to calculate the DFT by reducing $A^{(N)}$ to two $A^{(\frac{N}{2})}$ that is only half the size of $A^{(N)}$. This operation is repeated with every recursion until we reach the 1-point DFT.
Recall that the DFT is a matrix multiplication (Fig. 1.35). One stage of the FFT essentially reduces the multiplication by an $N \times N$ matrix to two multiplications by $\frac{N}{2} \times \frac{N}{2}$ matrices. This reduces the number of operations required to calculate the DFT by almost a factor of two (Fig. 1.39).

Another interpretation of FFT involves analyzing the matrix

$$A_{k,L} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-j \frac{2\pi k L}{N}} \\ 1 & -e^{-j \frac{2\pi k L}{N}} \end{pmatrix},$$

where $k$ and $L$ are nonnegative integers such that $k < 2^L$. Note that

$$\langle A_{k,L} x, A_{k,L} y \rangle = (A_{k,L} y)^H (A_{k,L} x) = y^H A_{k,L}^H A_{k,L} x = y^H \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & e^{-j \frac{2\pi k L}{N}} \\ e^{j \frac{2\pi k L}{N}} & -e^{-j \frac{2\pi k L}{N}} \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & \frac{e^{-j \frac{2\pi k L}{N}}}{2} \\ 1 & -e^{-j \frac{2\pi k L}{N}} \end{pmatrix} x = y^H \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} x = y^H x = \langle x, y \rangle,$$

i.e., multiplication by $A_{k,L}$ preserves distances and angles — roughly speaking, it is a rotation or reflection. Continuing the matrix decomposition of Fig. 1.39 further until we get the full FFT, it can be shown that FFT consists of $\frac{N}{2} \log N$ multiplications by $2 \times 2$ matrices of the form $\sqrt{2} A_{k,L}$, each operating on a pair of coordinates. Therefore, FFT breaks down the multiplication by the DFT matrix $A$ into elementary planar transformations.

### 1.4.1 Fast Computation of Convolution

Consider a linear system described by

$$y = Sx,$$  \hspace{1cm} (1.33)

where $x$ is the $N \times 1$ input vector, representing an $N$-periodic input signal; $S$ is an $N \times N$ matrix; and $y$ is the $N \times 1$ output vector, representing an $N$-periodic output signal. What conditions must the matrix $S$ satisfy in order for the system to be time-invariant, i.e., invariant to circular shifts of the input vector?

Note that a circular shift by one sample is

$$\begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ \vdots \\ x(N-1) \end{pmatrix} \rightarrow \begin{pmatrix} x(-1) = x(N - 1) \\ x(0) \\ x(1) \\ \vdots \\ x(N-2) \end{pmatrix}.$$

---

9 The same conclusion can be reached by examining an FFT diagram such as Fig. 1.38.
Let the first column of $S$ be

$$h = \begin{pmatrix} h(0) \\ h(1) \\ h(2) \\ \vdots \\ h(N-1) \end{pmatrix}.$$ 

Note that when

$$x = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

then $y = h$,

and when

$$x = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

then $y$ is the second column of $S$, which therefore, in order for $S$ to be invariant to circular shifts, must be equal to:

$$\begin{pmatrix} h(N-1) \\ h(0) \\ h(1) \\ \vdots \\ h(N-2) \end{pmatrix}.$$ 

Similarly, when

$$x = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix},$$

then $y$ is the third column of $S$, etc.

Thus, the matrix $S$ must have the following structure:

$$S = \begin{pmatrix} h(0) & h(N-1) & h(N-2) & \cdots & h(1) \\ h(1) & h(0) & h(N-1) & \cdots & h(2) \\ h(2) & h(1) & h(0) & \cdots & h(3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ h(N-1) & h(N-2) & h(N-3) & \cdots & h(0) \end{pmatrix}.$$
This is called a *circulant matrix*. We can then write Eq. (1.33) as

\[
y(n) = \sum_{m=0}^{N-1} x(m)h(n - m)
\]

\[
y(n) = \sum_{m=0}^{N-1} x(m)h((n - m) \text{ mod } N) \quad \quad \quad (1.34)
\]

\[
y(n) = x \oplus h(n) = x \otimes h \quad \quad \quad (1.35)
\]

Eq. (1.35) is called a *circular convolution* or a *periodic convolution*. Note that formula (1.34) works even when \( x \) or \( h \) are non-periodic. Observe the following:

- For \( y(0) \), the sum of the indices of \( x \) and \( h \) is always 0 mod \( N \) for every term.

\[
y(0) = x(0)h(0) + x(1)h(N - 1) + x(2)h(N - 2) + \cdots + x(N - 1)h(1)
\]

- For \( y(1) \), the sum of the indices of \( x \) and \( h \) is always 1 mod \( N \) for every term.

\[
y(1) = x(0)h(1) + x(1)h(0) + x(2)h(N - 1) + \cdots + x(N - 1)h(2)
\]

This is true for all \( y(k) \), \( k = 0, 1, \cdots, N - 1 \).

What are the eigenvectors of \( S \)? Let us try

\[
g_k = \begin{pmatrix}
\frac{1}{N} e^{j \frac{2 \pi k}{N} 1} \\
\frac{1}{N} e^{j \frac{2 \pi k}{N} 2} \\
\vdots \\
\frac{1}{N} e^{j \frac{2 \pi k}{N} (N-1)}
\end{pmatrix}, \text{ where } k = 0, 1, \cdots, N - 1.
\]

We have:

\[
y(n) = h(n) \oplus g_k
\]

\[
y(n) = \sum_{m=0}^{N-1} h(m)g_k(n - m)
\]

\[
y(n) = \sum_{m=0}^{N-1} h(m) \frac{1}{N} e^{j \frac{2 \pi k}{N} (n - m)}
\]

\[
y(n) = \left\{ \sum_{m=0}^{N-1} h(m) e^{-j \frac{2 \pi k m}{N}} \right\} \frac{1}{N} e^{j \frac{2 \pi k}{N} n}
\]

\[
y(n) = H(k) \frac{1}{N} e^{j \frac{2 \pi k}{N} n}
\]

DFT of \( h \)
Hence we have that
\[ Sg_k = H(k)g_k \]
where \( g_k \) is the \( k \)-th eigenvector and \( H(k) \) gives the corresponding eigenvalue. Therefore,
\[
S \left( \begin{array}{cccc}
g_0 & g_1 & \cdots & g_{N-1} 
\end{array} \right) = \left( \begin{array}{cccc}
g_0 & g_1 & \cdots & g_{N-1} 
\end{array} \right) \left( \begin{array}{cccc}
H(0) & & & \\
& H(1) & & 0 \\
& & \ddots & \\
0 & & & H(N-1)
\end{array} \right).
\]

Then \( S \) can be written as:
\[
S = B \left( \begin{array}{cccc}
H(0) & & & \\
& H(1) & & 0 \\
& & \ddots & \\
0 & & & H(N-1)
\end{array} \right) A,
\]
where the DFT matrix \( A \) is:
\[
A = NB^H = \left( \begin{array}{c}
g_0^H \\
g_1^H \\
\vdots \\
g_{N-1}^H
\end{array} \right).
\]

Complex exponentials are the eigenvectors of circulant matrices. They diagonalize circulant matrices. Thus, for any \( x \in \mathbb{C}^N \),
\[
Sx = B \left( \begin{array}{cccc}
H(0) & & & \\
& H(1) & & 0 \\
& & \ddots & \\
0 & & & H(N-1)
\end{array} \right) Ax.
\]

Let us compare two algorithms for computing the circular convolution of \( x \) and \( h \).

**Algorithm 1** Directly perform the multiplication \( Sx \). This has computational complexity \( O(N^2) \).

**Algorithm 2**
1. Represent \( x \) in the eigenbasis of \( S \), i.e., the Fourier basis,
\[
X = Ax.
\]
   This step can be done with FFT whose complexity is \( O(N \log N) \).
Step 1 | Step 2 | Step 3
--- | --- | ---
N-point DFT \( x(n) \rightarrow X(k) \) | \( Y(k) = X(k)H(k) \) | N-point IDFT \( Y(k) \rightarrow y(n) = x \circledast h(n) \)
N-point DFT \( h(n) \rightarrow H(k) \) |

Figure 1.40. An illustration of the FFT implementation of the circular convolution.

2. Compute the representation of \( y \) in the eigenbasis of \( S \):

\[
Y = \begin{pmatrix}
H(0) & 0 & \cdots & 0 \\
H(1) & & & \\
0 & & \ddots & \\
& & & H(N-1)
\end{pmatrix} \mathbf{X}.
\]

This computation has complexity \( \mathcal{O}(N) \).

3. Reconstruct \( y \) from its Fourier coefficients:

\( y = B\mathbf{Y} \).

This has complexity \( \mathcal{O}(N \log N) \), if done using the FFT.

This algorithm is summarized in Fig. 1.40. Its total complexity is \( \mathcal{O}(N \log N) \).

(Note that the second algorithm does not necessarily perform better for any matrix.)

**Example 1.27.** This example explores the relationship between the convolution and the circular convolution. Let \( x \) and \( h \) be \( N \)-periodic signals, and let

\[
x_z = \begin{cases} 
x(n), & 0 \leq n \leq N - 1 \\
0, & \text{otherwise}
\end{cases}
\]

\[
h_z = \begin{cases} 
h(n), & 0 \leq n \leq N - 1 \\
0, & \text{otherwise}
\end{cases}
\]

If we let

\[
y_z(n) = x_z * h_z(n)
\]

\[
y(n) = x \circledast h(n)
\]

then \( y(n) \) can be expressed as

\[
y(n) = \begin{cases} 
y_z(n) + y_z(N + n), & n = 0, 1, \cdots, N - 2 \\
y(N - 1), & n = N - 1
\end{cases}
\]
Note that the overlap of $y_z(n)$ and $y_z(N+n)$ causes temporal aliasing in the resulting $y(n)$. This is the main difference between convolution and circular convolution.

$$y_z(n) = x_z * h_z(n)$$

(a) Convolution

$$y(n) = x \otimes h(n)$$

(b) Circular convolution

Fig. 1.41 illustrates the effect of temporal aliasing. To remove or minimize the effect of temporal aliasing, we could zero-pad $x$ and $h$ so that the temporal replicas are spread further apart, and thus, overlapping would not occur.