A Brief Introduction to the FFT

Keith M. Chugg

April 1, 2021

The Fast Fourier Transform (FFT) is an algorithm for computing the Discrete Fourier Transform (DFT) more efficiently than directly using the DFT definition. The FFT is one of the most widely used algorithms in (electrical) engineering. It is widely implemented in software packages such as Matlab and Python's Numpy. Furthermore, there are many digital circuit implementations of the FFT used in a wide array of applications spanning imaging systems to WiFi modems.

1 Background and Definitions

The DFT is an expansion of a signal $x[n]$, defined on $\mathcal{Z}_N = \{0, 1, \ldots N-1\}$, in terms of the orthogonal basis signals

$$e_k[n] = \frac{1}{N} e^{j2\pi \frac{k}{N} n} \qquad k \in \mathcal{Z}_N,\ n \in \mathcal{Z}_N \tag{1}$$

where the N frequencies $\nu_k = k/N$ for $k \in \mathcal{Z}_N$ yield complex exponentials with period N. There are various forms for the DFT, depending on the convention for scaling the complex exponentials in (1) – *i.e.,* the factor of $1/N$ is most common in the engineering literature and software packages. If the factor of $1/N$ is replaced by 1, the DFT is often referred to as the Discrete Time Fourier Series [1]. If the factor of $1/N$ is replaced by $1/\sqrt{N}$, the transform is sometimes called the Normalized DFT – in this case the basis is orthonormal – and is sometimes used in the math literature. The basic results herein apply with a scaling factor regardless of these conventions.

For the scaling convention in (1), the DFT is

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n} = \sum_{n=0}^{N-1} x[n] W_N^{kn} \qquad k \in \mathcal{Z}_N \tag{2a}$$

$$W_N = e^{-j\frac{2\pi}{N}} \tag{2b}$$

where the W_N notation is widely adopted in the FFT literature and $W = W_N$ will be adopted when the context is clear. It is clear from (2a) that the DFT can be viewed as a matrix-vector multiplication. Specifically, the matrix

$$\mathbf{F}_N = \begin{bmatrix} W^0 & W^0 & W^0 & \cdots & W^0 & W^0 \\ W^0 & W^1 & W^2 & \cdots & W^{N-2} & W^{N-1} \\ W^0 & W^2 & W^4 & \cdots & W^{2(N-2)} & W^{2(N-1)} \\ \vdots & & & \ddots & & \vdots \\ W^0 & W^{N-2} & W^{2(N-2)} & \cdots & W^{(N-2)(N-2)} & W^{(N-2)(N-1)} \\ W^0 & W^{N-1} & W^{2(N-1)} & \cdots & W^{(N-1)(N-2)} & W^{(N-1)(N-1)} \end{bmatrix} \tag{3}$$

so that the element of $\mathbf{F}_N$ at row k and column n is

$$[\mathbf{F}_N]_{k,n} = W_N^{kn} = W_N^{(kn \bmod N)} \tag{4}$$

where the last equality follows from the definition of W_N.

With this definition, we have that (2a) can be expressed in matrix-vector form as

$$\underset{(N\times 1)}{\mathbf{X}} = \underset{(N\times N)}{\mathbf{F}_N}\underset{(N\times 1)}{\mathbf{x}} \tag{5}$$

where $\mathbf{X}$ is the vector of DFT coefficients $\{X[k]\}$ and $\mathbf{x}$ is the vector of time samples $\{x[n]\}$, ordered from 0 down to $N-1$.

An important observation is that performing the DFT computation in (2a), or equivalently the matrix-vector multiplication in (5), is equivalent to computing N dot products each of size N. It follows that the number of multiply accumulate (MAC) operations scales as N^2.

2 Fast Fourier Transform

In this section we will outline a method for computing the DFT, the FFT, with a number of MAC operations that scale as $N\log_2 N$. There are many variants of the FFT, so our goal is just to convey the main idea and provide a simple example.

3 An FFT Example for $N = 4$

The basic idea of the FFT is to compute the size N DFT in terms of two size $N/2$ DFTs and then apply this concept repeatedly until one is computing only $N = 2$ DFTs directly. Thus, the 2-point DFT is important to consider first. For $N = 2$, we have $W_2 = e^{-j\pi} = -1$ and

$$X[0] = x[0] + x[1] \tag{6}$$

$$X[1] = x[0] - x[1] \tag{7}$$

$$\mathbf{F}_2 = \begin{bmatrix} +1 & +1 \\ +1 & -1 \end{bmatrix} \tag{8}$$

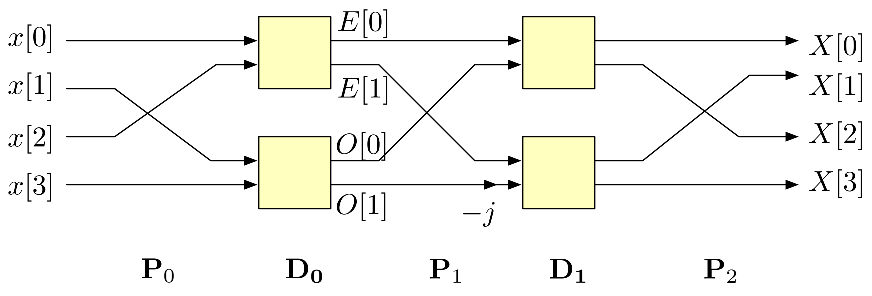
Note that the 2-point DFT comprises only a "DC term ($\nu_0 = 0$)" (the sum of all $x[n]$) and the highest frequency term ($\nu_1 = 0.5$), which is the difference between the two time samples.

Let us now consider an $N = 4$ DFT and see if we can compute this using 2-point DFTs. For $N = 4$ we have

$$W_4 = e^{-j\frac{2\pi}{4}} = (-j) \tag{9}$$

It follows that the DFT matrix is

$$\mathbf{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & (-j)^2 & (-j)^3 \\ 1 & (-j)^2 & (-j)^4 & (-j)^6 \\ 1 & (-j)^3 & (-j)^6 & (-j)^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & +j \\ 1 & -1 & 1 & -1 \\ 1 & +j & -1 & -j \end{bmatrix} \tag{10}$$



$$\mathbf{F}_4 = \mathbf{P}_2\mathbf{D}_1\mathbf{P}_1\mathbf{D}_0\mathbf{P}_0$$

Figure 1: An $N = 4$ FFT is obtained by computing the 4-point DFT using two stages, each computing two, $N = 2$ point DFTs with permutations between computational stages. Yellow boxes are 2-point DFTs.

where the $W_N^{kn} = W_N^{(kn \bmod N)}$ identity has been applied. Using this, it is straightforward to compute the $N = 4$ DFT as

$$\begin{aligned}
X[0] &= x[0] + x[1] + x[2] + x[3] && \text{(11a)}\\
&= (x[0] + x[2]) + (x[1] + x[3]) && \text{(11b)}\\
&= E[0] + O[0] && \text{(11c)}\\
X[1] &= x[0] - jx[1] - x[2] + jx[3] && \text{(11d)}\\
&= (x[0] - x[2]) - j(x[1] - x[3]) && \text{(11e)}\\
&= E[1] + (-j)O[1] && \text{(11f)}\\
X[2] &= x[0] - x[1] + x[2] + x[3] && \text{(11g)}\\
&= (x[0] + x[2]) - (x[1] + x[3]) && \text{(11h)}\\
&= E[0] - O[0] && \text{(11i)}\\
X[3] &= x[0] + jx[1] - x[2] - jx[3] && \text{(11j)}\\
&= (x[0] - x[2]) + j(x[1] - x[3]) && \text{(11k)}\\
&= E[1] - (-j)O[1] && \text{(11l)}
\end{aligned}$$

where we have introduced "even" and "odd" signals:

$$\begin{aligned}
e[0] &= x[0] & e[1] &= x[2] && \text{(12a)}\\
o[0] &= x[1] & o[1] &= x[3] && \text{(12b)}
\end{aligned}$$

and we have defined $E[k]$ and $O[k]$ as the 2-point DFTs of $e[n]$ and $o[n]$, respectively, as is implied in (11). This processing, where $E[k]$ and $O[k]$ are computed, then combined to obtain $X[k]$, is shown in Fig. 1. This is a 4-point FFT to compute a 4-point DFT.

3.1 The FFT as a DFT Matrix Factorization ($N = 4$ Example)

Below each stage of the processing in Fig. 1, a matrix label has been assigned – this denotes the equivalent matrix-vector multiplication for each stage in the processing flow diagram. Specifically,

$\{\mathbf{P}_i\}$ are (4×4) permutation matrices and $\{\mathbf{D}_i\}$ are (4×4) matrices that are block diagonal with (2×2) blocks. These matrices can be read off of the digram and imply a factorization of the DFT matrix $\mathbf{F}_4$ as also shown in Fig. 1. Specifically, we have

$$\mathbf{P}_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13a}$$

$$\mathbf{D}_0 = \begin{bmatrix} +1 & +1 & 0 & 0 \\ +1 & -1 & 0 & 0 \\ 0 & 0 & +1 & +1 \\ 0 & 0 & +1 & -1 \end{bmatrix} \tag{13b}$$

$$\mathbf{P}_1 = \mathbf{P}_0 \tag{13c}$$

$$\mathbf{D}_1 = \begin{bmatrix} +1 & +1 & 0 & 0 \\ +1 & -1 & 0 & 0 \\ 0 & 0 & +1 & +1 \\ 0 & 0 & +j & -j \end{bmatrix} \tag{13d}$$

$$\mathbf{P}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{13e}$$

where the scaling of $O[1]$ by $-j$ has been absorbed into $\mathbf{D}_1$. It is straightforward to verify that

$$\mathbf{F}_4 = \mathbf{P}_2\mathbf{D}_1\mathbf{P}_1\mathbf{D}_0\mathbf{P}_0 \tag{14}$$

Thus, the FFT can be viewed as factorization of the DFT matrix into a sequence of permutation and block diagonal matrices.

4 Generalizing to $N = 2^q$

To get some insight into how the $N = 4$ may generalize, consider that the key step was to separate $x[n]$ into the two subsequences $e[m] = x[2m]$ and $o[m] = x[2m + 1]$ for $m \in \mathcal{Z}_M$ where $M = N/2$. This approach is known as *decimation in time* and leads directly to an FFT algorithm for N a power of 2. Specifically, the DFT of $x[n]$ can be computed by combining the $N/2$-point DFTs for the even and odd subsequences

$$\begin{aligned}
X[k] = \mathbb{DFT}_N\left\{x[n]\right\} &= \sum_{n=0}^{N-1} x[n] e^{-j2\pi \frac{k}{N} n} && (15)\\
&= \sum_{m=0}^{M-1} x[2m] e^{-j2\pi \frac{k}{N}(2m)} + \sum_{m=0}^{M-1} x[2m+1] e^{-j2\pi \frac{k}{N}(2m+1)} && (16)\\
&= \sum_{m=0}^{M-1} x[2m] e^{-j2\pi \frac{k}{M} m} + e^{-j2\pi \frac{k}{N}} \sum_{m=0}^{M-1} x[2m+1] e^{-j2\pi \frac{k}{M} m} && (17)\\
&= \mathbb{DFT}_M\left\{e[n]\right\} + W_N^k \mathbb{DFT}_M\left\{o[n]\right\} && (18)\\
&= E[k] + W_N^k O[k] && (19)\\
& && (20)
\end{aligned}$$

Note that $E[k]$ and $O[k]$ should be interpreted as periodic in k with period $M = N/2$ since $k \in \mathcal{Z}_N$. This means that that the pair $(E[k], O[k])$ are combined twice with a different "twiddle factor" W_N^k to produce the N-point DFT of $x[n]$. Specifically, for $k \in \mathcal{Z}_M$

$$X[k] = E[k] + W_N^k O[k] \quad (21a)$$

$$X[k+M] = E[k] - W_N^k O[k] \quad (21b)$$

since $E[k]$ and $O[k]$ are periodic with period M and

$$W_N^{k+N/2} = e^{-j2\pi \frac{k}{N}} e^{-j\pi} = -W_N^k. \quad (22)$$

The two equations in (21) can be viewed as a 2-point DFT of $E[k]$ and $W_N^k O[k]$. Thus, the N-point DFT can be obtained by first computing $N/2$-point DFTs over the even and odd subsequences and then combining the results using $N/2$ size 2 DFTs. In fact, the expression in (11) is this exact relationship with $W_4 = -j$. This concept is illustrated in Fig. 2 where this is applied to the $N = 8$ case.

Note that if the green boxes in Fig. 2, which are 4-point DFTs, are replaced by the $N = 4$ FFT diagram in Fig. 1, we obtain an $N = 8$ FFT. This is shown in Fig. 3. This comprises 3 computation stages, each having four 2-point DFTs.

When N is a power of 2, this process can be repeated recursively as we have shown for $N = 8$. For example, if we consider $N = 16$, we can use (21) to compute this DFT using two 8-point DFTs and a combining stage of eight, 2-point DFTs. Then, we could replace the 8-point DFTs with the the 8-point FFT in Fig. 3, yielding a 16-point FFT. In this manner, when N is a power of 2, the FFT can be constructed and it consists of $\log_2(N)$ stages, each comprising $N/2$ DFT units. Thus, in contrast to the direct computation of the DFT via matrix multiplication, which has complexity scaling as N^2, the FFT has complexity that scales as $N\log_2(N)$. Thus, the important result from the FFT is that

$$\frac{\text{Computational Complexity of Direct-DFT Computation}}{\text{Computational Complexity of FFT Computation}} \sim \frac{N^2}{N\log_2(N)} = \frac{N}{\log_2(N)}$$

For large values of N, this difference is very significant. For example, for $N = 1024$, this ratio is about 100. For $N = 2^{16}$ this ratio is 4096.

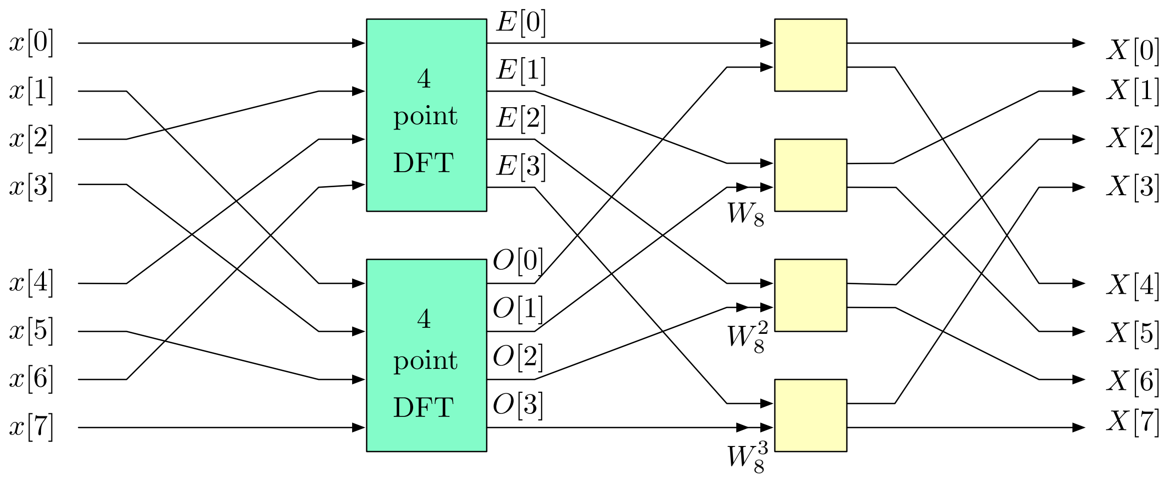


Figure 2: An $N = 8$ point DFT can be computed using two stages. In the first stage, two 4-point DFTs are computed over the even and odd indices, respectively. In the second stage, four, 2-point DFTs are computed. Yellow boxes are 2-point DFTs. Replacing the $N = 4$ DFT (green) boxes by the $N = 4$ FFT in Fig. 1 yields an $N = 8$ FFT.

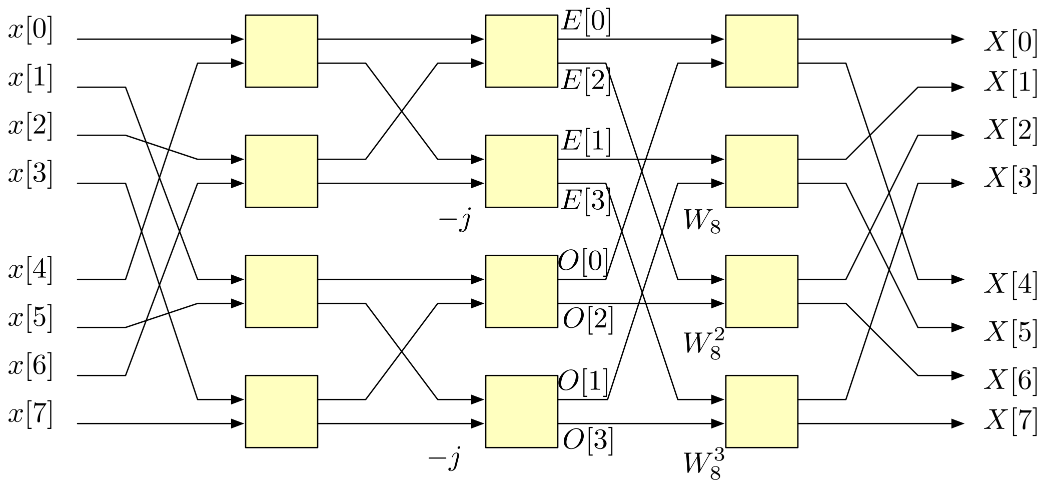


Figure 3: An $N = 8$ point FFT obtained by replacing the $N = 4$ DFT (green) boxes in Fig. 2 by the $N = 4$ FFT in Fig. 1. Yellow boxes are 2-point DFTs. Note that this now has $3 = \log_2(8)$ computational stages, each with $4 = N/2$, 2-point DFTs.

5 Variations and Notes

Since the DFT is self-dual, the inverse DFT can be computed using an FFT architecture very similar to that used for the FFT. In fact, another popular architecture is to develop a fast transform for the inverse DFT by starting with $X[k]$ and decimating in frequency. This yields a computational graph that is essentially the transpose (reverse) of that in Fig. 3. Another interesting variant yields a computational graph for which the permutations between all stages are each the same perfect shuffle permutation. This regular structure may be especially attractive for digital hardware implementations.

The FFTs considered in this handout are all "radix-2" FFTs, meaning that they continue to simplify down to 2-point DFT computations. This can be generalized to radix-4 architectures where the basic DFT blocks are 4-point DFTs. This may be useful if one has a parallel processing resource that can compute a 4-point DFT matrix multiply in one clock cycle. This can be generalized to other values for the FFT radix, but the radix-2 architecture is by far the most widely employed.

Considering N to be a power of 2 simplifies the exposition and directly yields the decomposition down to 2-point DFTs. Most software packages, however, support any value of N. The book by Porat [2] is a good reference for FFT development when N is not a power of 2. The matrix factorization approach can yield interesting insights into the FFT. Poularikis and Seely's book [3] develops the FFT using matrix factorizations and the traditional equation-based approach and is a good starting reference. This book also covers the fast Hadamard transform (FHT) which has a development very similar to that of the FFT. In fact, there are other generalization of the FFT that involve the selection of an orthogonal basis so that the change-of-coordinate matrix (*e.g.,* $\mathbf{F}_N$ in our case) can be factored into a sequence of permutation and block diagonal matrices. A popular reference for the FFT is Oppenheim and Schafer [4] and their notation and diagram conventions are widely adopted in academia and industry.

References

[1] A. V. Oppenheim, A. S. Willsky, and I. T. Young, *Signals and Systems.* Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

[2] B. Porat, *A course in digital signal processing.* Wiley, 1997.

[3] D. Poularikas and S. L. Seely, *Signals and Systems.* Boston, MA: B/C Engineering, 1985.

[4] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing.* Englewood Cliffs, New Jersey: Prentice-Hall, 1975.