Supplementary Notes for ELEN 4810 Lecture 5
Computing Fourier Representations: DFT and FFT

John Wright
Columbia University

October 6, 2025

Disclaimer: These notes are intended to be an accessible introduction to the subject, with no pretense at completeness. In general, you can find more thorough discussions in Oppenheim's book. Please let me know if you find any typos.

Reading suggestions: Oppenheim and Schafer Sections 8.1-8.7, 9.1-9.3

In this lecture, we discuss the Discrete Fourier Transform (DFT), which is a Fourier representation for finite length signals. The main practical importance of this new representation is that (unlike the DTFT), it can be computed very efficiently, for arbitrary inputs. This makes it the primary tool for performing frequency-domain analysis of signals on a computer.

1 Computing Fourier Representations

The discrete-time Fourier transform is an extremely useful tool for analyzing signal processing systems. However, for practical calculation, it has the disadvantage that $X(e^{j\omega})$ is defined on $\omega \in \mathbb{R}$, and hence the full transform cannot be directly calculated. It is also worth noting that the discrete-time Fourier transform is defined via an infinite summation

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] \exp(-j\omega n), \tag{1.1}$$

while, in practice we always work with signals $x[n]$ of finite length.

The *discrete Fourier transform* (DFT) is a Fourier representation for *finite-length* signals

$$x[n], \qquad n = 0, 1, \ldots, N-1. \tag{1.2}$$

We say that such an x has length N.

As with all of the fourier representations that we have introduced thus far, we begin by defining a basis of complex exponentials. Write

$$\varphi_k[n] = \exp\left(j\frac{2\pi k}{N}n\right). \tag{1.3}$$

Notice that φ_k is a complex exponential sequence, and that it is periodic with period N.

In the early lectures, we showed that there are exactly N distinguishable N-periodic complex exponentials, because $\varphi_k[n]$ and $\varphi_{k+rN}[n]$ are equal for any integer r and any N. So, we can restrict our attention to

$$\varphi_0[n], \ldots, \varphi_{N-1}[n]. \tag{1.4}$$

Writing down an inner product for length N signals,

$$\langle x, w \rangle = \sum_{n=0}^{N-1} x[n] w^*[n], \tag{1.5}$$

a quick calculation shows that

$$\langle \varphi_k, \varphi_\ell \rangle = \begin{cases} N & k = \ell \\ 0 & \text{else} \end{cases} \tag{1.6}$$

and so, $\varphi_0, \ldots, \varphi_{N-1}$ are mutually orthogonal.

The length-N DFT of x expresses x in terms of these mutually orthogonal sequences, via

$$\begin{aligned} X[k] &= \langle x, \varphi_k \rangle \\ &= \sum_{n=0}^{N-1} x[n] \exp\left(-j \frac{2\pi k}{N} n\right). \end{aligned} \tag{1.7}$$

Here, $x[n]$ is a discrete time sequence, and $X[k]$ is a discrete frequency representation – it is defined over $k = 0, 1, \ldots, N-1$. As with all of our other fourier representations, the minus sign in the exponent comes from the definition of the complex inner product.

Using (1.6), we can immediately obtain a reconstruction formula,

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \varphi_k[n] \tag{1.8}$$

$$= \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp\left(j \frac{2\pi k}{N} n\right). \tag{1.9}$$

To verify that this is correct, simply plug in the definition of $X[k]$, and then use the fact that the φ_k are orthogonal – I invite you to complete this proof yourself.

Matrix representation. The length-N signals $x[n]$ and $X[k]$ can be viewed as N-dimensional complex vectors. They are related via the linear equation

$$\begin{bmatrix} X[0] \\ X[1] \\ X[2] \\ \vdots \\ X[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & e^{-j\frac{2\pi}{N}} & (e^{-j\frac{2\pi}{N}})^2 & \cdots & (e^{-j\frac{2\pi}{N}})^{N-1} \\ 1 & (e^{-j\frac{2\pi}{N}})^2 & (e^{-j\frac{2\pi}{N}})^4 & \cdots & (e^{-j\frac{2\pi}{N}})^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & (e^{-j\frac{2\pi}{N}})^{(N-1)} & (e^{-j\frac{2\pi}{N}})^{2(N-1)} & \cdots & (e^{-j\frac{2\pi}{N}})^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ x[2] \\ \vdots \\ x[N-1] \end{bmatrix}$$

$$\doteq \boldsymbol{F}\boldsymbol{x}. \tag{1.10}$$

The matrix $\boldsymbol{F} \in \mathbb{C}^{N\times N}$ is called the DFT matrix. If we imagine that it is indexed by $k = 0, 1, \ldots N-1$ and $n = 0, 1, \ldots, N-1$, the matrix elements can be expressed concisely as $\boldsymbol{F}_{kn} = \left(e^{-j\frac{2\pi}{N}}\right)^{kn}$. In the text, this is sometimes expressed by setting $W_N = e^{-j\frac{2\pi}{N}}$, and noting that $\boldsymbol{F}_{kn} = W_N^{kn}$. Notice that $W_N^N = 1$; for this reason, the quantity W_N is referred to as a "root of unity".

Notice that the rows of the matrix $\boldsymbol{F}$ are the conjugate transposes of the basis elements φ_k:

$$\boldsymbol{F} = \begin{bmatrix} \varphi_0^* \\ \varphi_1^* \\ \vdots \\ \varphi_{N-1}^* \end{bmatrix}. \tag{1.11}$$

Moreover,

$$\boldsymbol{F}^* = \left[\begin{array}{c|c|c|c} \varphi_0 & \varphi_1 & \cdots & \varphi_{N-1} \end{array}\right], \tag{1.12}$$

and

$$\boldsymbol{F}\boldsymbol{F}^* = \begin{bmatrix} \langle\varphi_0,\varphi_0\rangle & \langle\varphi_1,\varphi_0\rangle & \cdots & \langle\varphi_{N-1},\varphi_0\rangle \\ \langle\varphi_0,\varphi_1\rangle & \langle\varphi_1,\varphi_1\rangle & \cdots & \langle\varphi_{N-1},\varphi_1\rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle\varphi_0,\varphi_{N-1}\rangle & \langle\varphi_1,\varphi_{N-1}\rangle & \cdots & \langle\varphi_{N-1},\varphi_{N-1}\rangle \end{bmatrix} = N\boldsymbol{I}, \tag{1.13}$$

where $\boldsymbol{I}$ is the $N \times N$ identity matrix. Thus

$$\boldsymbol{F}^*\boldsymbol{F} = \boldsymbol{F}\boldsymbol{F}^* = N\boldsymbol{I}, \tag{1.14}$$

where $\boldsymbol{F}^*$ denotes the *conjugate transpose* of $\boldsymbol{F}$. In the language of linear algebra, $\frac{1}{\sqrt{N}}\boldsymbol{F}$ is a *unitary matrix*.

2 Two interpretations of the DFT

The discrete Fourier transform that we have just introduced can be interpreted in several ways. We give two such interpretations; these and others are discussed in the text (cf. Sections 8.2-8.4).

Sampling the DTFT. The most straightforward interpretation is that the sequence $X[k]$ is a *sampling* of the DTFT $X(e^{j\omega})$ of the signal x. Namely, if we take the convention that $x[n] = 0$ when $n \notin \{0, 1, \ldots, N-1\}$, then

$$X[k] = \sum_{n=-\infty}^{\infty} x[n] \exp\left(-j\frac{2\pi k}{N}n\right) \tag{2.1}$$

$$= X(e^{j\omega})\big|_{\omega=\frac{2\pi k}{N}}. \tag{2.2}$$

So, the DFT just evaluates the DTFT at equally spaced points $\omega = 0, \frac{2\pi}{N}, \ldots, \frac{2\pi(N-1)}{N}$. This seems like a very natural way of approximating the DTFT on a computer!

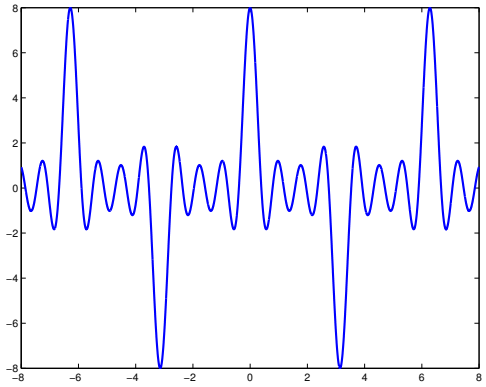

Figure 1: **The periodic sinc** $\sin(Nt)/\sin(t)$, for $N = 8$.

Since the DFT $X[k]$ completely specifies a length-N signal, via the inverse DFT formula (1.9), $X[k]$ also completely specifies the DTFT $X(e^{j\omega})$ of the length n signal. We can write down an explicit formula for $X(e^{j\omega})$ in terms of $X[k]$ as follows:

$$X(e^{j\omega}) = \sum_{n=0}^{N-1}\left(\frac{1}{N}\sum_{k=0}^{N-1}X[k]\exp\left(j\frac{2\pi}{N}kn\right)\right)\exp\left(-j\omega n\right) \tag{2.3}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1}X[k]\sum_{n=0}^{N-1}\exp\left(-j\left(\omega-\frac{2\pi k}{N}\right)n\right) \tag{2.4}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1}X[k]\underbrace{\left\{\exp\left(-\frac{j}{2}\left(\omega-\frac{2\pi k}{N}\right)\right)\right\}}_{\text{Phase shift}}\times\underbrace{\left\{\frac{\sin\left(N\frac{\omega-\frac{2\pi k}{N}}{2}\right)}{\sin\left(\frac{\omega-\frac{2\pi k}{N}}{2}\right)}\right\}}_{\text{Real valued}}. \tag{2.5}$$

Figure 2 plots the "periodic sinc" function $\sin(Nt)/\sin(t)$ for $N = 8$. Two points are worth emphasizing. First, this interpolation formula *only* works for length-N signals. For such signals, the entire DTFT can be reconstructed from knowledge of the (length N) DFT only. Second, the formula itself is not so important – I certainly would not recommend memorizing it. The important point is that the formula exists.

Periodic Extension and Discrete Fourier Series. Assuming that $x[n] = 0$ when $n \notin \{0, 1, \ldots, N-1\}$ is not the only way to extend a finite length signal to all of the integers $\mathbb{Z}$. Sometimes, for reasons we will see below, it is useful to also think about the N-periodic extension $\tilde{x}[n]$, which simply constructs an N-periodic sequence, each period of which corresponds to the finite length signal x: for each $r \in \mathbb{Z}$, and n, set

$$\tilde{x}[n + rN] = x[n]. \tag{2.6}$$

In continuous time, we know that a periodic signal can be represented by a Fourier series. In discrete time, we can view the DFT as offering a *discrete Fourier series* representation for the periodic signal

$\tilde{x}$, by just extending the basis signals φ_k to be defined for all $n \in \mathbb{Z}$. Namely,

$$\tilde{x}[n] = \frac{1}{N} \sum_{k=0}^{N-1} \tilde{X}[k] \exp\left(j \frac{2\pi}{N} kn\right) \tag{2.7}$$

The coefficients $\tilde{X}[k]$ in this representation are just the DFT coefficients $X[k]$.

This interpretation of the DFT may seem artificial – in some sense, the interpretation of the DFT as sampling the DTFT is much more natural. However, we will see that the DFS interpretation actually makes it easier to interpret some of the properties of the DFT, which differ in important ways from the corresponding properties of the DTFT.[1]

3 Properties of the DFT

To discuss the properties of the DFT, we need to work with *modular arithmetic*. For integers $r, N \in \mathbb{Z}$, we define

$$r \mod N \tag{3.1}$$

to be the unique integer $r' \in \{0, 1, \ldots, N-1\}$ of the form $r' = r - kN$ for some $k \in \mathbb{Z}$. For example

$$\begin{aligned} 2 \mod 5 &= 2 \\ 6 \mod 5 &= 1 \\ 5 \mod 5 &= 0. \end{aligned}$$

The text uses the notation $((n))$ and $((n))_N$ for $n \mod N$.

With this noted, we can begin listing properties of the DFT, e.g.,

Proposition 3.1 (Linearity)**.** *The DFT is linear: if X_1 is the DFT of x_1 and X_2 is the DFT of x_2, then $\alpha X_1 + \beta X_2$ is the DFT of $\alpha x_1 + \beta x_2$.*

This follows almost immediately from the definition. Like the DTFT, the DFT has the property that conjugation in time produces a conjugate flip in frequency. We have to be careful about the notion of a flip for a sequence $X[k]$ defined on $k = 0, 1, \ldots, N-1$.

Proposition 3.2 (Conjugation)**.** *If $x[n]$ has DFT $X[k]$, then the conjugate sequence $x^*[n]$ has DFT $X^*[-k \mod N]$.*

This likewise follows directly from the definition – I leave it to you to check. This implies a number of symmetry properties of the DFT of a real signal, which are listed in the textbook.

The situation becomes less straightforward when we think about the analogue of the shift property for the DFT. Note that the N-point DFT is defined over sequences on $0, 1, \ldots, N-1$, and so it is not even clear what a shift $x[n - n_0]$ would mean here. Nevertheless, we can consider the DFT $X[k]$ of x, and ask *what is the length-N signal x' whose DFT is*

$$X'[k] = \exp\left(-j \frac{2\pi k}{N} m\right) X[k] \qquad ? \tag{3.2}$$

The following proposition gives the answer:

[1] There is a third interpretation, in terms of the DTFT of the periodic extension, given in Section 8.4 of the text. You may also find this interpretation useful in interpreting the DFT.

Proposition 3.3. *Suppose that $X[k]$ is the DFT of a length N sequence $x[n]$, and set $X'[k] = \exp\left(-j\frac{2\pi k}{N}m\right) X[k]$. Then $X'[k]$ is the DFT of a sequence $x'[n]$ that is cyclically shifted to the right by m:*

$$x'[n] = x[(n-m) \mod N]. \tag{3.3}$$

Proof. We simply apply the inverse transform:

$$\begin{aligned}
x'[n] &= \frac{1}{N}\sum_{k=0}^{N-1} X'[k] \exp\left(j\frac{2\pi}{N}nk\right) && (3.4)\\
&= \frac{1}{N}\sum_{k=0}^{N-1} \exp\left(j\frac{2\pi}{N}(nk-mk)\right) X[k] && (3.5)\\
&= \frac{1}{N}\sum_{k=0}^{N-1} \exp\left(j\frac{2\pi}{N}(nk-mk)\right) \sum_{\ell=0}^{N-1} x[\ell] \exp\left(-j\frac{2\pi}{N}\ell k\right) && (3.6)\\
&= \sum_{\ell=0}^{N-1} x[\ell] \underbrace{\frac{1}{N}\sum_{k=0}^{N-1} \exp\left(j\frac{2\pi}{N}(n-m-\ell)k\right)}_{=1,\ \text{if } \ell = n-m \mod N,\ 0 \text{ else}} && (3.7)\\
&= x[(n-m) \mod N]. && (3.8)
\end{aligned}$$

□

This is a good example of why it is worth being careful about the DFT, and defining it as a first class object in its own right (rather than just a sampling of the DTFT). Although the DFT *is* just a sampled version of the DTFT, it behaves slightly differently. This example also illustrates some of the value of the interpretation of the DFT as the discrete Fourier series coefficients of a periodic extension. The complicated operation of cyclic shifting can be thought of as a simple linear shift of the periodic extension.

Arguably, the most important property of the DTFT is that convolution in the time domain becomes pointwise multiplication in the frequency domain. This property also has an analogue for the DFT, although again, we will end up doing arithmetic modulo N.

Proposition 3.4. *If $x[n]$ and $h[n]$ are two length N signals, with N-point DFT's $X[k]$ and $H[k]$, respectively, and $Y[k] = X[k]H[k]$, then $Y[k]$ is the DFT of a signal*

$$\begin{aligned}
y[n] &= \sum_{m=0}^{N-1} x[m \mod N] h[n-m \mod N] && (3.9)\\
&= \sum_{m=0}^{N-1} x[m] h[n-m \mod N]. && (3.10)
\end{aligned}$$

We use the notation $y = x \circledast h$ for (3.10), and refer to it as *cyclic convolution*.

Proof. The proof of this identity mimics that of others that we've seen.

$$
\begin{aligned}
y[n] &= \frac{1}{N}\sum_{k=0}^{N-1} H[k]X[k]\exp\left(j\frac{2\pi}{N}kn\right) && (3.11)\\
&= \frac{1}{N}\sum_{k=0}^{N-1}\left(\sum_{\ell=0}^{N-1} h[\ell]\exp\left(-j\frac{2\pi}{N}k\ell\right)\right)\left(\sum_{m=0}^{N-1} x[m]\exp\left(-j\frac{2\pi}{N}km\right)\right)\exp\left(j\frac{2\pi}{N}kn\right) && (3.12)\\
&= \sum_{m=0}^{N-1} x[m]\sum_{\ell} h[\ell]\underbrace{\frac{1}{N}\sum_{k=0}^{N-1}\exp\left(j\frac{2\pi}{N}(n-m-\ell)k\right)}_{=1,\text{ if }\ell=n-m \mod N,\ 0\text{ else.}} && (3.13)\\
&= \sum_{m=0}^{N-1} x[m]h[n-m \mod N] && (3.14)\\
&= (x\circledast h)[n]. && (3.15)
\end{aligned}
$$

□

Cyclic convolution is an even more complicated operation than linear convolution. There are two equivalent ways of thinking about it. Either, we can simply work through the sequence of steps to evaluate (3.10). This consists of flipping h, shifting it to the right by m, applying the $\mod N$ operation to wrap the result to lie on $\{0,1,\ldots,N-1\}$ multiplying with x, and then adding. *Or* we can simply observe that $y[n] = x * \tilde{h}[n]$ for $n \in \{0,1,\ldots,N-1\}$. That is to say, the cyclic convolution can be obtained by performing linear convolution with the N-periodized version $\tilde{h}$ of the length N signal h.

Why should we care about such a crazy operation? Since convolution in time is equivalent to multiplication in frequency, we could try to calculate the linear convolution $x * h$ by computing the DFT of each, multiplying and then computing the inverse DFT. Both the DFT and its inverse have extremely efficient algorithms (which we will describe in the next lecture). Typically, this procedure is much more efficient than simply computing $x * h$ in time domain. Unfortunately, Proposition 3.4 implies that we have to be careful – if we just compute $\mathrm{DFT}_N^{-1}\left(X[k]H[k]\right)$, the result will not be the linear convolution $x * h$, but rather the *cyclic convolution* $x \circledast h$. So, for our purposes, the main reason to care about cyclic convolution is because *it is what you are stuck with if you try to do convolution in the frequency domain using the DFT*.

Since we usually care about computing the linear convolution, the question is how to make sure that the cyclic convolution is equal to the linear convolution. The next proposition addresses this issue:

Proposition 3.5. *Consider two signals $x[n]$ and $h[n]$ of lengths N and N', respectively. The linear convolution $x * h$ has length $M = N + N' - 1$. If we* zero pad *x and h to length M, by adding $M - N$ zeros to the end of x to form $\bar{x}$ and $M - N'$ zeros to the end of h to form $\bar{h}$, then the cyclic convolution is equal to the linear convolution $x * h = \bar{x} \circledast \bar{h}$. Moreover, if $\bar{X}$ and $\bar{H}$ are the length-M DFT of x and h, respectively, then*

$$x * h = \mathrm{DFT}_M^{-1}\left\{\bar{X}\circ\bar{H}\right\}, \tag{3.16}$$

where $\bar{X}\circ\bar{H}[k] = \bar{X}[k]\bar{H}[k]$.

4 Computing the DFT

Our next goal is to efficiently compute the Discrete Fourier Transform

$$X[k] \;=\; \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) \qquad k = 0, \ldots, N-1. \tag{4.1}$$

We denote the output sequence as $X = \mathrm{DFT}_N\{x\}$, where N is the length of the input sequence x. There is a very straightforward algorithm for computing the sequence X: simply use the formula (4.1)! Computing each entry $X[k]$ requires N complex multiplications and $N-1$ complex additions. We can compute the entire sequence X by computing each element sequentially, using a total of N^2 complex multiplications and $N(N-1)$ complex additions. Typical general purpose digital computers do not have dedicated hardware for complex arithmetic, and so, complex operations are performed using real arithmetic and additional logic. For example, a complex multiplication $(a+jb)(c+jd)$ can be computed using four real multiplications and two real additions. A complex addition can be computed using two real additions. So, if we want to be extremely precise, the naive algorithm for (4.1) requires $4N^2$ real multiplications and $2N^2 + 2N(N-1) = 4N^2 - 2N$ real additions. In analyzing algorithms, this kind of precision is typically overkill – we are happy to know qualitatively how the computation grows with N. Because it requires $4N^2$ multiplications, the naive algorithm takes time $O(N^2)$.[2]

5 Fast Fourier Transform Algorithms

Fast Fourier Transform (FFT) algorithms fundamentally improve over the naive algorithm, reducing the computational cost from $O(N^2)$ to $O(N \log N)$. In addition, the hidden constant in the $O(\cdot)$ notation is not too large, making the FFT (and its variants) *the* algorithm of choice for computing the DFT in practice.

The core idea in the FFT algorithm is to reduce the computation of *one length-N DFT* two the computation of *two length-$N/2$ DFT's*, plus an additional $O(N)$ computation to combine them together. This *divide and conquer* strategy can be iterated to reduce the computation of the DFT to successively smaller DFT's (length $N/4$, $N/8$, $N/16$, ...), until one finally arrives at a base case (conceptually, $N = 2$), which can be evaluated by direct computation. For notational simplicity, we describe these procedures under the assumption that N is a power of 2: $N = 2^p$ for some p.

There are two popular strategies for reducing a length-N DFT to two length $N/2$ DFT's, known as *decimation in time* and *decimation in frequency*.

5.1 Decimation in time

The decimation-in-time FFT architecture is a consequence of the following result, which shows that we can compute DFT of an even-length sequence $x[n]$ by combining the DFT's of the even terms $x_e[n] = x[2n]$ and odd terms $x_o[n] = x[2n+1]$:

[2]The "big-O" or Landau notation has the following meaning: we write $f(N) = O(g(N))$ if there exists a constant C independent of N such that for all N, $f(N) \leq Cg(N)$.

Theorem 5.1. *Let $x[n]$ be a length-N sequence, with N an even integer. Let*

$$x_e[n] = x[2n], \qquad n = 0, \ldots, N/2 - 1 \tag{5.1}$$
$$x_o[n] = x[2n+1], \qquad n = 0, \ldots, N/2 - 1. \tag{5.2}$$

Then

$$\mathrm{DFT}_N\{x\}[k] = \mathrm{DFT}_{N/2}\{x_e\}\left[k \bmod \frac{N}{2}\right] + \exp\left(-j\frac{2\pi k}{N}\right)\mathrm{DFT}_{N/2}\{x_o\}\left[k \bmod \frac{N}{2}\right]. \tag{5.3}$$

Proof. We calculate

$$\mathrm{DFT}_N\{x\}[k] = \sum_{n=0}^{N-1} x[n]\exp\left(-j\frac{2\pi kn}{N}\right) \tag{5.4}$$
$$= \sum_{i=0}^{N/2-1} x[2i]\exp\left(-j\frac{2\pi k\times 2i}{N}\right) + \sum_{\ell=0}^{N/2-1} x[2\ell+1]\exp\left(-j\frac{2\pi k\times(2\ell+1)}{N}\right) \tag{5.5}$$
$$= \sum_{i=0}^{N/2-1} x_e[i]\exp\left(-j\frac{2\pi ki}{N/2}\right) + \exp\left(-j\frac{2\pi k}{N}\right)\sum_{\ell=0}^{N/2-1} x_o[\ell]\exp\left(-j\frac{2\pi k\ell}{N/2}\right) \tag{5.6}$$
$$= \mathrm{DFT}_{N/2}\{x_e\}\left[k \bmod \frac{N}{2}\right] + \exp\left(-j\frac{2\pi k}{N}\right)\mathrm{DFT}_{N/2}\{x_o\}\left[k \bmod \frac{N}{2}\right] \tag{5.7}$$

□

This result immediately suggests an architecture for calculating the N-point DFT, which we draw as Figure 2: namely, we compute the $N/2$-point DFT of the event terms, the $N/2$-point DFT of the odd terms, and then combine them together. Each entry $X[k]$ of the N-point DFT of x can be generated by adding an entry of $\mathrm{DFT}_{N/2}\{x_e\}$ with an entry of $\mathrm{DFT}_{N/2}\{x_o\}$, scaled by a complex exponential.

Let T_N be the number of arithmetic operations required to compute an N-point DFT. Using the architecture in Figure 2, the computational complexity is at most $2T_{N/2}+CN$. The first term accounts for the cost of the two $N/2$-point DFT's, while the second term accounts for the cost of combining their outputs to produce $X[k]$. If we were to simply implement the $N/2$-point DFT's using the naive algorithm, we would not have saved all that much – their cost would be proportional to $4\times(N/2)^2$, and so we would save roughly a factor of two. We can try to further improve the complexity by iterating the procedure – expressing each of the $N/2$-point DFT's in terms of two $N/4$-point DFT's, as shown in Figure 3. This leads to a computational cost of at most $4T_{N/4} + 2CN$. It is clear that if $N = 2^p$ for some integer p, we can iterate this procedure $p-1$, arriving at a computational cost of at most $2^{p-1}T_2 + (p-1)CN$. We can compute the two-point DFT directly, in constant time. The architecture for doing this is drawn in Figure 4. Since T_2 is a constant, 2^{p-1} is bounded by N, and $p-1$ is bounded by $\log_2 N$, the overall computational cost of this algorithm is bounded by a constant times $N\log N$. This is a remarkable improvement over the $O(N^2)$ algorithms described above.

Several practical simplifications are possible. First, we can note that for $W_N = \exp\left(-j\frac{2\pi}{N}\right)$, $W_{N/2}^r = W_N^{2r}$. So, we can express all of the complex coefficients in Figure 3 as powers of W_N. This yields Figure 5. In Figure 5, we repeatedly perform a basic operation, pictured in Figure 6(left).

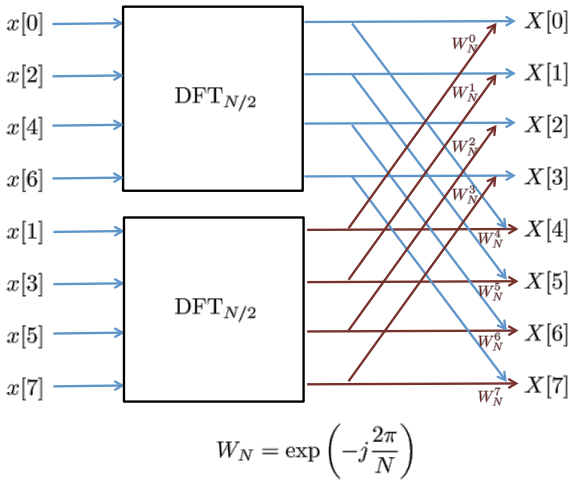

Figure 2: **Recursive DFT architecture suggested by Theorem 5.1 (Decimation in Time)**. In this diagram, arrows leading into other lines represent additions; a scalar component labeling any line represents a scalar multiplication. So, each of the components $X[k]$ of the output length-N DFT is generated by adding an output of the top length $N/2$ DFT to a scaled output of the bottom length $N/2$ DFT.

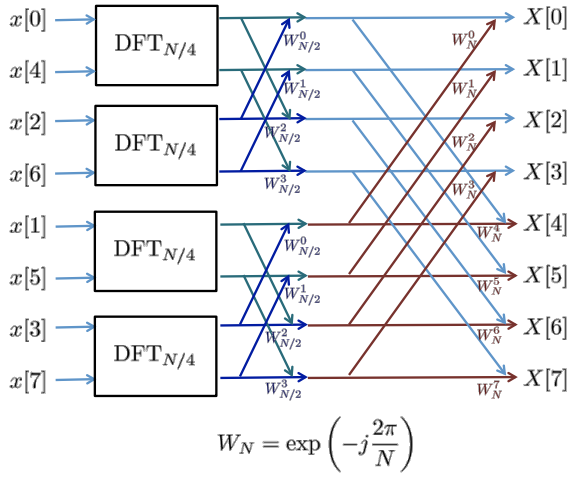

Figure 3: **Recursive DFT architecture**, with two levels of recursion, generated by replacing the $\mathrm{DFT}_{N/2}$ boxes in Figure 1 with two length $N/4$ DFT's, using Theorem 1 (Decimation in Time).

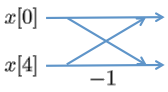

Figure 4: The base of the recursion: a two-point DFT, using two complex additions and a single negation.

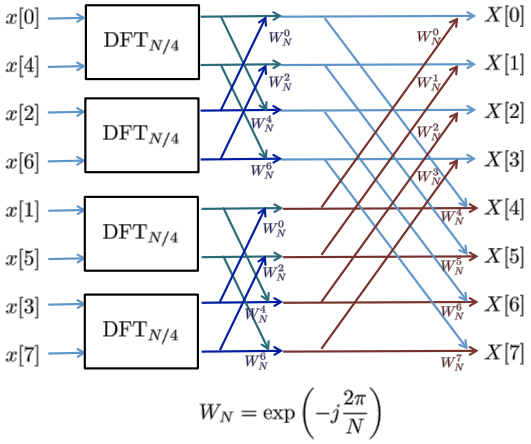

Figure 5: Recursive DFT architecture, with coefficients as powers of W_N.

This operation combines two complex numbers in two different ways, with multiplicative coefficients which are powers of W_N. With a glance at Figure 5 and some mental extrapolation, we can guess that (i) the multiplicative coefficients are always applied to the "bottom" input, and (ii) the multiplicative coefficients are always of the form W_N^r and $W_N^{r+N/2} = -W_N^r$. We can exploit this observation to replace one of the multiplications with a subtraction, yielding the basic structure in Figure 6(right). Substituting this transformation into Figure 5 yields a practical FFT architecture. The text describes additional practical considerations, including indexing the input and operating "in place" with limited memory.

The algorithm described here is a special case of the *Cooley-Tukey FFT*, published by Cooley and Tukey in 1965. It turned out that this algorithm was actually known to (and used by!) Gauss as early as 1805, for interpolation problems in astronomy. The procedure works as described when

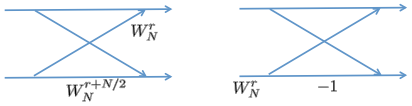

Figure 6: The basic operation in a decimation-in-time FFT (left), and a slight simplification (right), which uses that $W_N^{r+N/2} = -W_N^r$.

$N = 2^p$ for some integer p. Variants are known for other bases than 2 – e.g., $N = 3^p$. A more general version of the Cooley-Tukey algorithm can be applied to general composite integers N: for $N = N_1 N_2$ we can reduce the computation to N_1 DFT's of length N_2. There are also algorithmic variants which apply to N that are not composite (i.e., prime N). These work by expressing the length-N DFT as a convolution, and then computing this convolution in frequency domain using FFT's of different (composite) length. For example, for prime N *Rader's algorithm* reduces the DFT to 3 DFT's of (composite) length $N - 1$. You can consult, e.g., Section 9.6.2 of the text for more information about these approaches.

5.2 Decimation in frequency

The decimation-in-frequency FFT architecture is inspired by the following observation on the DFT of a length-N sequence $x[n]$, for N an even integer. We assume that $x[n]$ is defined for $n = 0, 1, \ldots, N-1$; our goal is to compute the discrete Fourier transform $X[k] = \mathrm{DFT}_N\{x\}[k]$ for $k = 0, 1, \ldots, N-1$. The decimation-in-frequency architecture results from the observation that we can compute the terms for even k and the terms for odd k separately:

Theorem 5.2. *Let*

$$x_0[n] = x[n] + x[n + N/2], \tag{5.8}$$

$$x_1[n] = \Big(x[n] - x[n + N/2]\Big) \exp\left(-j\frac{2\pi n}{N}\right). \tag{5.9}$$

Then

$$\mathrm{DFT}_N\{x\}[k] = \begin{cases} \mathrm{DFT}_{N/2}\{x_0\}[k/2] & k \text{ even} \\ \mathrm{DFT}_{N/2}\{x_1\}\left[\frac{k-1}{2}\right] & k \text{ odd}. \end{cases} \tag{5.10}$$

Proof. For $k = 2i$ an even integer, we calculate

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) \tag{5.11}$$

$$= \sum_{n=0}^{N/2-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) + \sum_{n=N/2}^{N-1} x[n + N/2] \underbrace{\exp\left(-j\frac{2\pi k(n + N/2)}{N}\right)}_{= \exp\left(-j\frac{2\pi kn}{N}\right), \text{ since } \exp\left(-j\frac{2\pi kN/2}{N}\right) = 1 \text{ for } k = 2i.} \tag{5.12}$$

$$= \sum_{n=0}^{N/2-1} (x[n] + x[n + N/2]) \exp\left(-j\frac{2\pi kn}{N}\right) \tag{5.13}$$

$$= \sum_{n=0}^{N/2-1} x_0[n] \exp\left(-j\frac{2\pi in}{N/2}\right) \tag{5.14}$$

$$= \mathrm{DFT}_{N/2}\{x_0\}[i]. \tag{5.15}$$

For $k = 2i + 1$ an odd integer, we similarly calculate

$$\begin{aligned} X[k] &= \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) && (5.16)\\ &= \sum_{n=0}^{N/2-1} x[n] \exp\left(-j\frac{2\pi kn}{N}\right) + x[n+N/2] \exp\left(-j\frac{2\pi k(n+N/2)}{N}\right) && (5.17)\\ &= \sum_{n=0}^{N/2-1} (x[n] - x[n+N/2]) \exp\left(-j\frac{2\pi kn}{N}\right) && (5.18)\\ &= \sum_{n=0}^{N/2-1} (x[n] - x[n+N/2]) \exp\left(-j\frac{2\pi n}{N}\right) \exp\left(-j\frac{2\pi in}{N/2}\right) && (5.19)\\ &= \mathrm{DFT}_{N/2}\left\{x_1\right\}[i], && (5.20) \end{aligned}$$

completing the proof. □

Thus, we can compute the DFT of x by first computing the length $N/2$ DFT's of x_0 and x_1, and then interleaving the results. Iterating this operation again leads to an $O(N \log N)$ time algorithm, also referred to as the FFT, here computed by *decimation in frequency*. Based on the formula, you can easily draw similar diagrams relating the input and the output, as we did for decimation in time.

Decimation in time and decimation in frequency lead to the same asymptotic complexity $O(N \log N)$. The relative advantages and disadvantages between decimation in time and decimation in frequency tend to be platform specific. Most platforms/languages in which you would think to perform digital signal processing already come with highly optimized implementations of the FFT – as such, you are unlikely to ever implement the FFT, except as a learning exercise. For example, Matlab's implementation (`fft`) uses a combination of the decimation-in-time (Cooley-Tukey) algorithm, with Radar's algorithm (alluded to above, and in the text) to handle prime N, and optimized implementations of special base cases. For our purposes, the important point is to understand why the DFT can be computed so efficiently.

6 Inverse DFT via the FFT

The inverse discrete Fourier transform has a very similar structure to the forward transform. In fact, the inverse DFT can be computed directly using the (forward) FFT and some simple manipulation. Since

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] \exp\left(j\frac{2\pi nk}{N}\right), \quad (6.1)$$

we have the identity

$$\begin{aligned} Nx^*[n] &= \sum_{k=0}^{N-1} X^*[k] \exp\left(-j\frac{2\pi nk}{N}\right) && (6.2)\\ &= \mathrm{DFT}_N\left\{X^*\right\}[n]. && (6.3) \end{aligned}$$

Hence, we can evaluate the inverse DFT as

$$x[n] = \frac{1}{N} \left(\mathrm{DFT}_N \left\{ X^* \right\} [n] \right)^* . \tag{6.4}$$

The DFT in this expression can be computed efficiently via the FFT.

As above, the main purpose of this derivation is to convince you that we can compute the inverse DFT efficiently using a single FFT and some additional minor manipulations. There are a variety of ways of relating relating $x[n]$ to a (forward) DFT of a manipulated version of $X[k]$, which may have practical advantages over the above formula. You can consult the text and its references for more details on these alternatives.