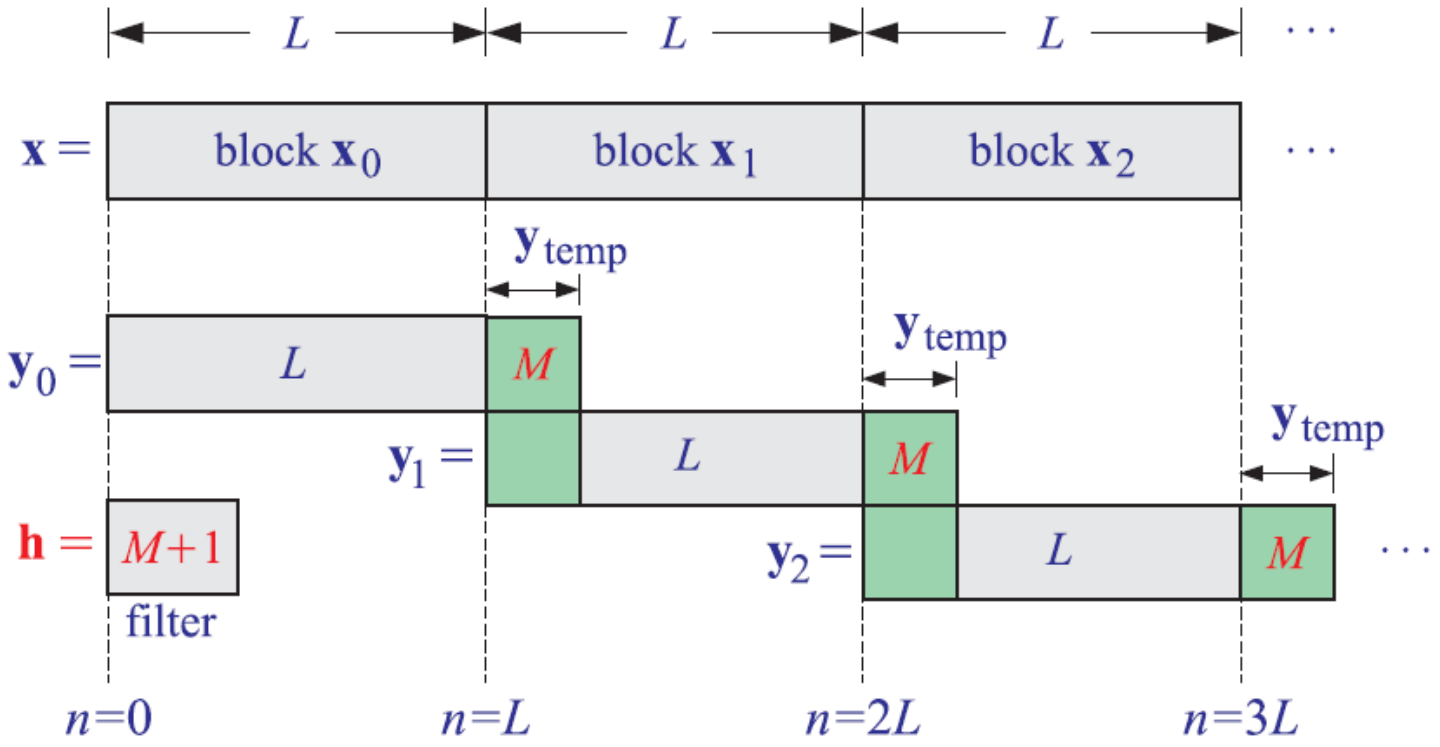
DSA – March 1, 2021

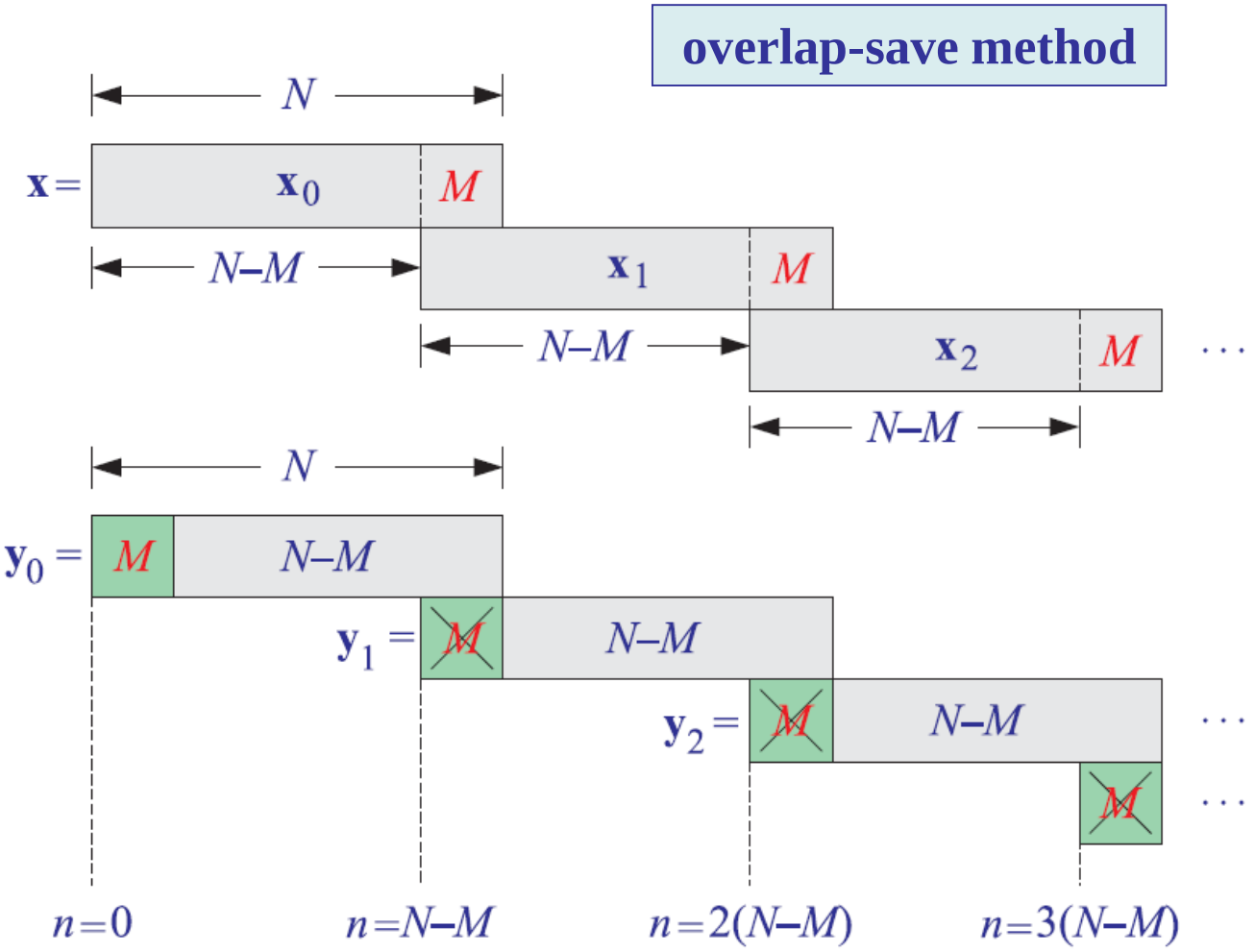
Topics: DTFT, DFT, physical vs. computational resolution, inverse DFT, mod-*N* wrapping, DFS, DFT matrix, FFT algorithm, circular convolution, overlap-add and overlap-save methods of fast convolution with examples, digital matched filters.

overlap-add method



DSA – March 1, 2021

DTFT, DFT, physical vs. computational resolution, inverse DFT, mod-*N* wrapping, DFS, DFT matrix, FFT algorithm, circular convolution, overlap-add and overlap-save methods of fast convolution with examples, digital matched filters



DTFT
frequency resolution and windowing

I2SP – Ch.9
O&S – Ch.10

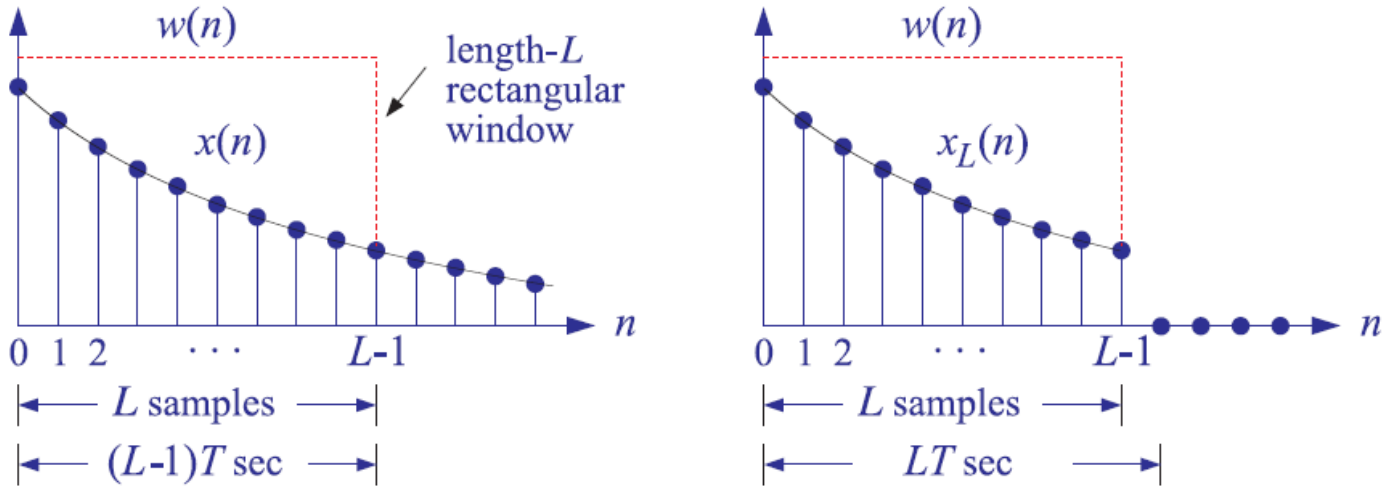
The discrete Fourier transform (DFT) and its fast implementation, the fast Fourier transform (FFT), have three major uses in DSP:

(a) the numerical *computation* of the frequency spectrum of a signal

(b) the efficient implementation of *convolution* by the FFT

(c) the *coding* of waveforms, such as speech or pictures, for efficient transmission and storage

Even though $\widehat{X}(f)$ is the closest approximation to $X(f)$ that we can achieve by DSP, it is still not computable because generally it requires an infinite number of samples $x(nT)$, $-\infty < n < \infty$. To make it computable, we must make a second approximation to $X(f)$, keeping only a finite number of samples, say, $x(nT)$, $0 \leq n \leq L-1$. This *time-windowing* process is illustrated below.



In terms of the time samples $x(nT)$, the original sampled spectrum $\widehat{X}(f)$ and its time-windowed version $\widehat{X}_L(f)$ are given by:

$$\widehat{X}(f) = \sum_{n=-\infty}^{\infty} x(nT)e^{-2\pi jfnT}$$

(DTFT)

$$\widehat{X}_L(f) = \sum_{n=0}^{L-1} x(nT)e^{-2\pi jfnT}$$

We may take the duration of the data record to be, in seconds and in samples:

$$\boxed{T_L = LT \quad \Rightarrow \quad L = \frac{T_L}{T} = f_s T_L}$$

The windowed signal may be thought of as an infinite signal which is zero outside the range of the window and agrees with the original one within the window. Defining the *rectangular window* of length L:

$$w(n) = \begin{cases} 1, & \text{if } 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

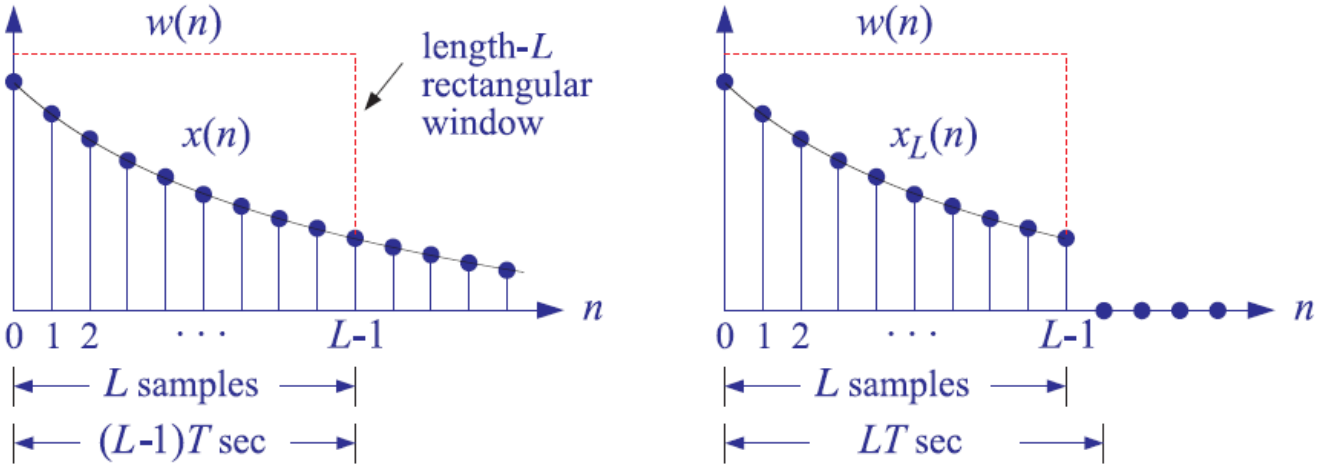
then, the windowed signal can be expressed as follows:

$$x_L(n) = x(n)w(n) = \begin{cases} x(n), & \text{if } 0 \le n \le L-1 \\ 0, & \text{otherwise} \end{cases}$$

In terms of digital frequency, $\omega = 2\pi f / f_s$, we may denote the DTFTs,

$$\boxed{\omega = \frac{2\pi f}{f_s}}$$

$$\boxed{\begin{aligned} X(\omega) &= \sum_{n=-\infty}^{\infty} x(n)e^{-j\omega n} \\ X_L(\omega) &= \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = \sum_{n=-\infty}^{\infty} x_L(n)e^{-j\omega n} \end{aligned}} \quad \text{(DTFT)}$$



As the length L of the data window increases, the windowed signal $x_L(n)$ becomes a better approximation of $x(n)$, and thus, $X_L(\omega)$, a better approximation of $X(\omega)$. In general, the windowing process has two major effects:

1. It reduces the *frequency resolution* of the computed spectrum, in the sense that the smallest resolvable frequency difference is limited by the length of the data record, that is, $\Delta f = 1/T_L$. This is the well-known "uncertainty principle."

2. It introduces *spurious* high-frequency components into the spectrum, which are caused by the sharp clipping of the signal $x(n)$ at the left and right ends of the rectangular window. This effect is referred to as "frequency leakage."

Both effects can be understood by deriving the precise connection of the windowed spectrum $X_L(\omega)$ to the unwindowed one $X(\omega)$. Using the property that the Fourier transform of the *product* of two time functions is the *convolution* of their Fourier transforms, we obtain the frequency-domain version of, $x_L(n) = x(n)w(n)$,

$$\boxed{X_L(\omega) = \int_{-\pi}^{\pi} X(\omega')W(\omega - \omega')\,\frac{d\omega'}{2\pi}}$$

where $W(\omega)$ is the DTFT of the rectangular window $w(n)$, that is,

$$W(\omega) = \sum_{n=0}^{L-1} w(n)e^{-j\omega n}$$

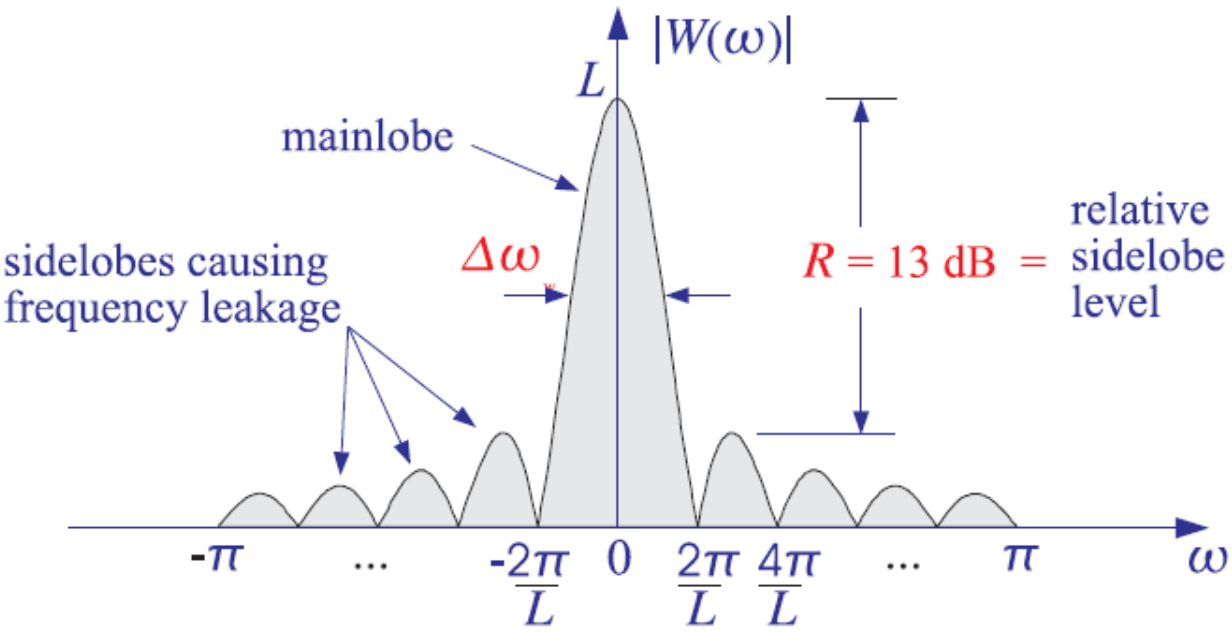
It can be thought of as the evaluation of the z-transform on the unit circle at $z = e^{j\omega}$. Setting $w(n) = 1$ in the sum, we find:

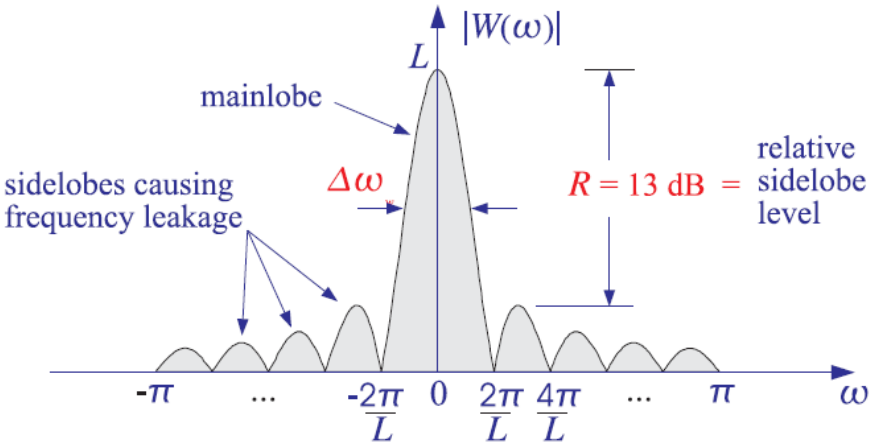
$$W(z) = \sum_{n=0}^{L-1} w(n)z^{-n} = \sum_{n=0}^{L-1} z^{-n} = \frac{1 - z^{-L}}{1 - z^{-1}}$$

Setting $z = e^{j\omega}$, we find for $W(\omega)$:

$$\boxed{W(\omega) = \frac{1 - e^{-jL\omega}}{1 - e^{-j\omega}} = \frac{\sin(\omega L/2)}{\sin(\omega/2)}\,e^{-j\omega(L-1)/2}}$$

$$W(\omega) = \frac{1 - e^{-jL\omega}}{1 - e^{-j\omega}} = \frac{\sin(\omega L/2)}{\sin(\omega/2)}\, e^{-j\omega(L-1)/2}$$





The mainlobe peak at DC dominates the spectrum, because $w(n)$ is essentially a DC signal, except when it cuts off at its endpoints. The higher frequency components that have "leaked" away from DC and lie under the sidelobes represent the sharp transitions of $w(n)$ at the endpoints.

The *width* of the mainlobe can be defined in different ways. For example, we may take it to be the width of the base, $4\pi/L$, or, take it to be the 3-dB width, that is, where $|W(\omega)|^2$ drops by $1/2$. For simplicity, we will define it to be *half* the base width, that is, in units of radians per sample:

$$\boxed{\Delta\omega_w = \frac{2\pi}{L}} \qquad \text{(rectangular window width)}$$

In units of Hz, it is defined through $\Delta\omega_w = 2\pi\Delta f_w/f_s$. Thus,

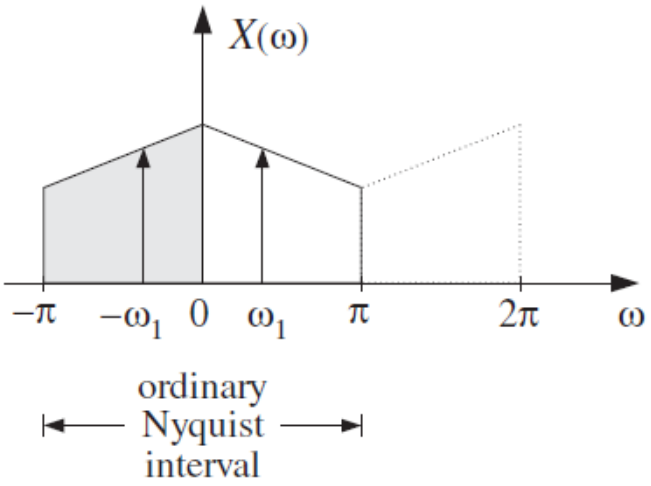
$$\boxed{\Delta f_w = \frac{f_s}{L} = \frac{1}{LT} = \frac{1}{T_L}}$$

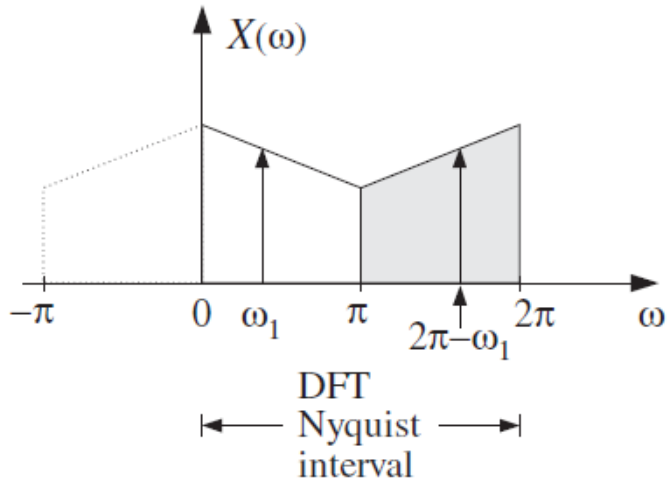
DTFT Computation

We discuss some computational aspects of the DTFT. Consider a length-L signal $x(n)$, $n = 0, 1, \ldots, L-1$, which may have been pre-windowed by a length-L non-rectangular window. Its DTFT can be written in the simplified notation:

$$\boxed{X(\omega) = \sum_{n=0}^{L-1} x(n) e^{-j\omega n}} \quad \text{(DTFT of length-}L\text{ signal)} \qquad (1)$$

It may be computed at any desired value of ω in the Nyquist interval $-\pi \leq \omega \leq \pi$. It is customary in the context of developing computational algorithms to take advantage of the periodicity of $X(\omega)$ (with period 2π) and map the conventional symmetric Nyquist interval $-\pi \leq \omega \leq \pi$ onto the right-sided one $0 \leq \omega \leq 2\pi$, referred to as the *DFT Nyquist interval*.





The positive-frequency subinterval $0 \leq \omega \leq \pi$ remains unchanged, but the negative-frequency one, $-\pi \leq \omega \leq 0$, gets mapped onto the second half of the DFT Nyquist interval, $\pi \leq \omega \leq 2\pi$.

For example, a cosinusoidal signal $\cos(\omega_1 n)$ with two spectral peaks at $\pm\omega_1$ will be represented by the two shifted peaks:

$$\{\omega_1,\ -\omega_1\} \Leftrightarrow \{\omega_1,\ 2\pi - \omega_1\}$$

The DTFT can also be thought of as the evaluation of the z-transform of the sequence $x(n)$ on the unit circle:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = \sum_{n=0}^{L-1} x(n)z^{-n}\Big|_{z=e^{j\omega}} = X(z)\Big|_{z=e^{j\omega}} \tag{2}$$

Thus, $X(\omega)$ can be computed by evaluating the polynomial $X(z)$ at $z = e^{j\omega}$. Hörner's rule of synthetic division is an efficient computational algorithm:

for each complex z do:
    $X = 0$
    for $n = L-1$ down to $n = 0$ do:
        $X = x(n) + z^{-1}X$

(Hörner's rule) (3)

Upon exit, X is the desired value of $X(z)$. To see how the iterations build up the z-transform, we iterate them for the case $L = 4$. Starting with $X = 0$ at $n = L - 1 = 3$, we have:

$$\begin{aligned} X &= x_3 + z^{-1}X = x_3 \\ X &= x_2 + z^{-1}X = x_2 + z^{-1}x_3 \\ X &= x_1 + z^{-1}X = x_1 + z^{-1}x_2 + z^{-2}x_3 \\ X &= x_0 + z^{-1}X = x_0 + z^{-1}x_1 + z^{-2}x_2 + z^{-3}x_3 = X(z) \end{aligned} \tag{4}$$

The built-in MATLAB function **polyval** uses Hörner's algorithm for polynomial evaluation. In turn, the latter is used in the function **freqz** to evaluate the DTFT at any range of ωs, for example, over the range, $\omega_a \leq \omega < \omega_b$.

$$\omega_k = \omega_a + k\,\frac{\omega_b - \omega_a}{N} = \omega_a + k\Delta\omega_{\text{bin}}\,, \quad k = 0, 1, \ldots N-1$$

where $\Delta\omega_{\text{bin}}$ is the *bin width*, that is, the spacing of the frequencies ω_k:

$$\Delta\omega_{\text{bin}} = \frac{\omega_b - \omega_a}{N}$$

The usage of **freqz** for DTFT computation is as follows:

```
% x = ...      % define length-L input signal
% w = ...      % vector of digital frequencies [rads/sample]

X = freqz(x,1,w);   % vector of DTFT values, same length as w
```

DFT

The *N-point DFT of a length-L signal* is defined to be the DTFT of the signal evaluated at N equally-spaced frequencies over the right-sided Nyquist interval, $0 \leq \omega \leq 2\pi$. The DFT frequencies are defined in radians per sample as follows:

$$\boxed{\omega_k = \frac{2\pi k}{N}}, \quad k = 0, 1, \ldots, N-1 \tag{5}$$

or, in Hz

$$\boxed{f_k = \frac{k f_s}{N}}, \quad k = 0, 1, \ldots, N-1 \tag{6}$$

Thus, the N-point DFT will be, for $k = 0, 1, \ldots, N-1$:

$$\boxed{X(\omega_k) = \sum_{n=0}^{L-1} x(n) e^{-j\omega_k n}} \quad (N\text{-point DFT of length-}L\text{ signal}) \tag{7}$$

The N-dimensional complex DFT array $X_k = X(\omega_k)$, $k = 0, 1, \ldots, N-1$ can be computed in a variety of ways:

(a) using the **freqz** function

(b) by the FFT algorithm, provided $N \geq L$

(c) in matrix form using the $N \times L$ DFT matrix, discussed below.

```
%  x = ...                  % define length-L input signal

k = 0:N-1;                  % DFT index
om = 2*pi*k/N;              % DFT frequencies
X = freqz(x,1,om);          % N-point DFT

X = fft(x,N);               % correct only if N>=L

X = A * x                   % A = NxL DFT matrix, x = Lx1 column
```

simplified notation for N-point DFT:

$$\boxed{X_k = \sum_{n=0}^{L-1} x(n) e^{-2\pi jkn/N}} \quad k = 0, 1, \ldots, N-1$$

The value at $k = N$, corresponding to $\omega_N = 2\pi$, is not computed because by periodicity it equals the value at $\omega_0 = 0$, that is, $X(\omega_N) = X(\omega_0)$. The bin-width, i.e., the spacing of the DFT frequencies is in rads/sample or Hz,

$$\Delta\omega_{\text{bin}} = \frac{2\pi}{N} \qquad \text{or,} \qquad \Delta f_{\text{bin}} = \frac{f_s}{N} \tag{8}$$

The standard DFT has its N frequencies distributed evenly over the full Nyquist interval, $[0, 2\pi)$, as shown below, but one can also use equally-spaced frequencies of any desired subinterval,

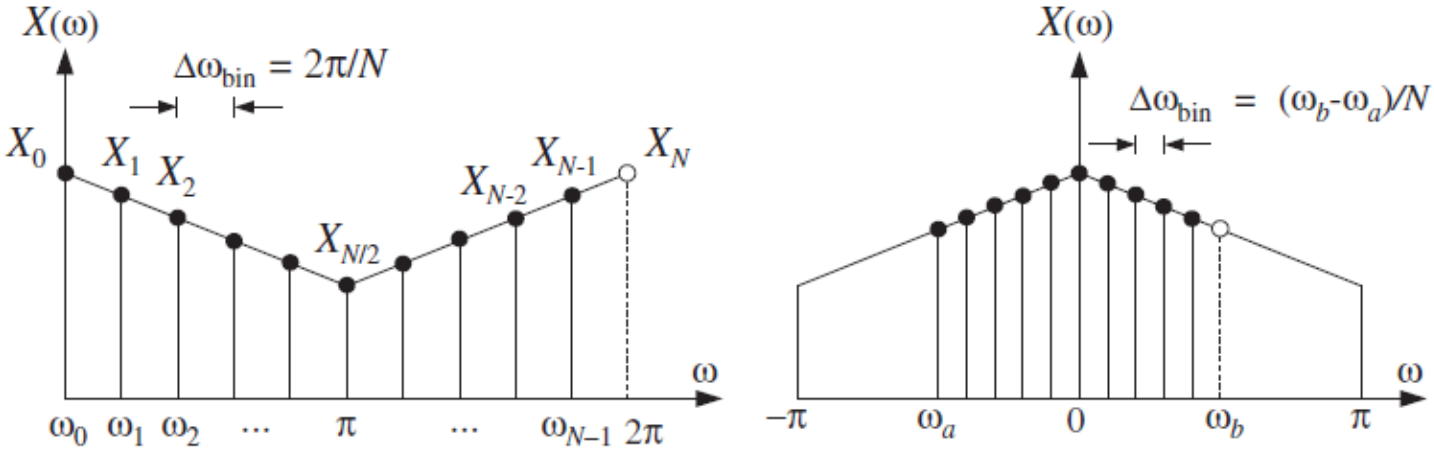


Fig. 1 N-point DTFTs over $[0, 2\pi)$ and over subinterval $[\omega_a, \omega_b)$, for $N = 10$.

The N computed values $X(\omega_k)$ can also be thought of as the evaluation of the z-transform $X(z)$ at the following z-points on the unit circle:

$$X(\omega_k) = X(z_k) = \sum_{n=0}^{L-1} x(n) z_k^{-n} \tag{9}$$

$$z_k = e^{j\omega_k} = e^{2\pi jk/N}, \qquad k = 0, 1, \ldots, N-1 \tag{10}$$

These are recognized as the Nth *roots of unity*, that is, the N solutions of the equation $z^N = 1$. They are evenly spaced around the unit circle at relative angle increments of $2\pi/N$, as shown in Fig. 2.

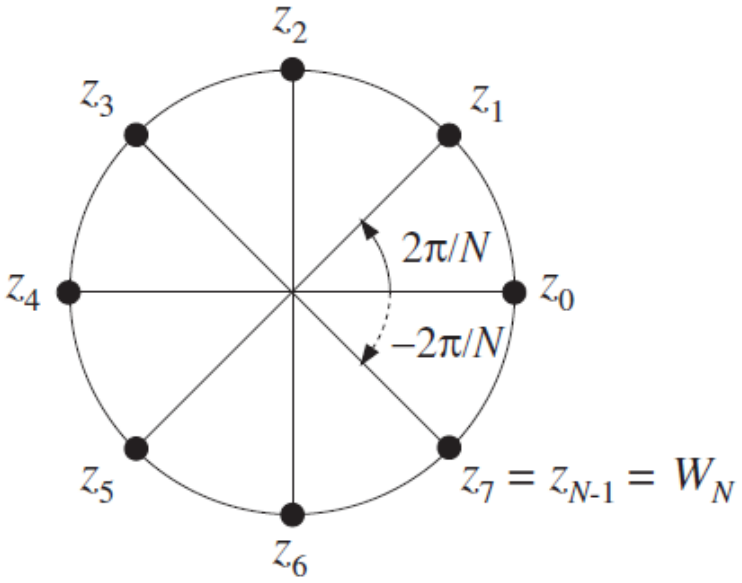


Fig. 2 Nth roots of unity, for $N = 8$.

Note also that the periodicity of $X(\omega)$ with period 2π is reflected in the periodicity of the DFT $X_k = X(\omega_k)$ in the index k with period N. This follows from:

$$\omega_{k+N} = \frac{2\pi(k+N)}{N} = \frac{2\pi k}{N} + 2\pi = \omega_k + 2\pi$$

which implies:

$$X_{k+N} = X(\omega_{k+N}) = X(\omega_k + 2\pi) = X(\omega_k) = X_k$$

Also, if the time signal $x(n)$ is real-valued, the Hermitian property of the DTFT can be combined with its 2π periodicity to give,

$$X^*(\omega) = X(-\omega) = X(2\pi - \omega)$$

and for the DFT,

$$X^*(\omega_k) = X(2\pi - \omega_k)$$

or, in terms of the DFT index,

$$\boxed{X_k^* = X_{N-k}} \quad k = 0, 1, \ldots, N-1$$

noting also that $X_0^* = X_N = X_0$, i.e., X_0 is real-valued.

Having computed an Npoint FFT, X_k, $k = 0, 1, \ldots, N-1$, it should be remembered that only the first $N/2$ outputs correspond to non-negative frequencies, that is,

$$X_k = X(\omega_k)\,, \quad k = 0, 1, \ldots, \frac{N}{2} - 1$$

$$\omega_k = \frac{2\pi k}{N}$$

whereas the remaining $N/2 - 1$ outputs get mapped to negative frequencies by the periodicity and conjugation conditions,

$$X_k = X^*_{-k} = X^*_{N-k} = X^*(\omega_{N-k}) = X^*(-\omega_k),, \quad k = \frac{N}{2}, \ldots, N-1$$

$$\omega_{N-k} = \frac{2\pi(N-k)}{N} = 2\pi - \omega_k$$

The MATLAB function **fftshift** can be used to recenter the computed DFT/FFT to the symmetric Nyquist interval, or the symmetric index interval,

$$-\pi \le \omega_k < \pi\,, \quad -\frac{N}{2} \le k \le \frac{N}{2} - 1$$

with usage,

```
X_shifted = fftshift(X);
```

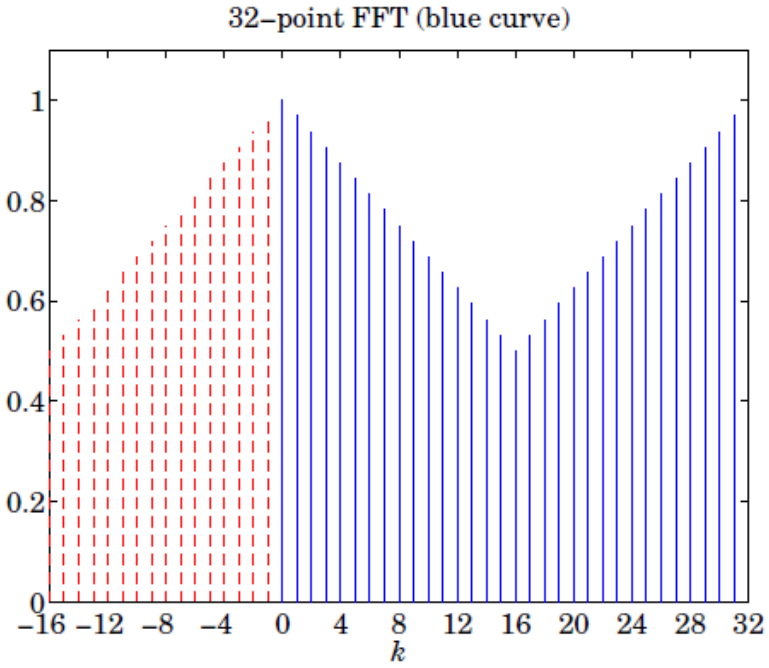
FFT shift example

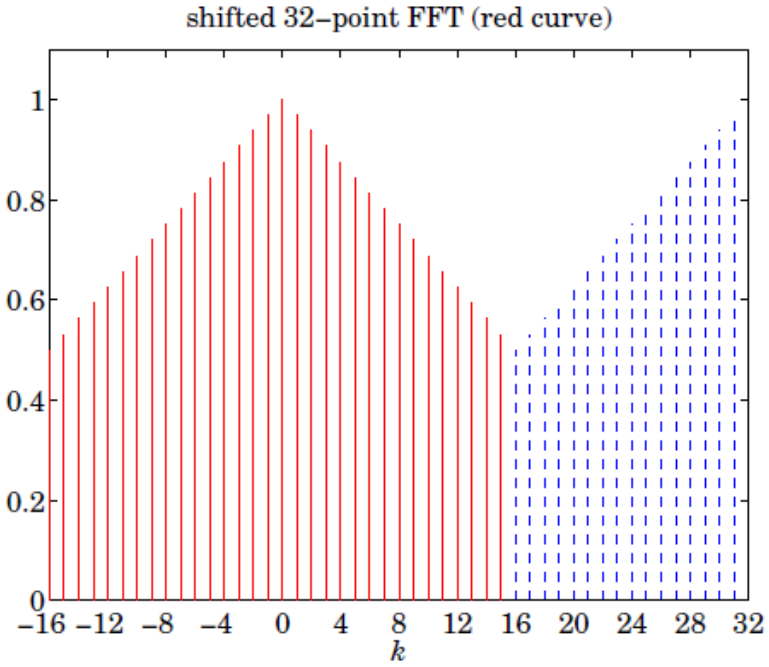
```
N = 32;

n = 0:N-1;
k = 0:N-1;

A = (1-k/N) .* (k<=N/2) + (k/N) .* (k>N/2);

[nn,kk] = meshgrid(n,k)

ww = 2*pi*kk/N;
AA = (1-kk/N) .* (kk<=N/2) + (kk/N) .* (kk>N/2);

x = real(sum(AA .* exp(j*ww.*nn))/N);

X = real(fft(x,N));

Xs = fftshift(X);
ks = -N/2 : N/2-1;

figure; stem(ks,Xs,'r--','marker','none'); hold on
stem(k,X,'b-','marker','none');

figure; stem(k,X,'b--','marker','none'); hold on
stem(ks,Xs,'r-','marker','none');
```

construct input $x(n)$

right-sided FFT, X_k

symmetric FFT, X_k





$$X_k = \frac{1}{32}\begin{bmatrix} 32 \\ 31 \\ 30 \\ 29 \\ 28 \\ 27 \\ 26 \\ 25 \\ 24 \\ 23 \\ 22 \\ 21 \\ 20 \\ 19 \\ 18 \\ 17 \\ 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \\ 26 \\ 27 \\ 28 \\ 29 \\ 30 \\ 31 \end{bmatrix}$$

$$\text{shifted} = \frac{1}{32}\begin{bmatrix} 16 \\ 17 \\ 18 \\ 19 \\ 20 \\ 21 \\ 22 \\ 23 \\ 24 \\ 25 \\ 26 \\ 27 \\ 28 \\ 29 \\ 30 \\ 31 \\ 32 \\ 31 \\ 30 \\ 29 \\ 28 \\ 27 \\ 26 \\ 25 \\ 24 \\ 23 \\ 22 \\ 21 \\ 20 \\ 19 \\ 18 \\ 17 \end{bmatrix} \leftarrow$$

Zero Padding

In principle, the two lengths L and N can be specified *independently* of each other: L is the number of *time samples* in the data record and can even be infinite; N is the number of *frequencies* at which we choose to evaluate the DTFT.

Most discussions of the DFT assume that $L = N$. The reason for this will be discussed later. If $L < N$, we can pad $N - L$ zeros at the end of the data record to make it of length N. If $L > N$, we may reduce the data record to length N by *wrapping* it modulo-N—a process to be discussed later.

Padding any number of zeros at the *end* of a signal has no effect on its DTFT. For example, padding D zeros will result into a length-$(L+D)$ signal:

$$\mathbf{x} = [x_0, x_1, \ldots, x_{L-1}]$$
$$\mathbf{x}_D = [x_0, x_1, \ldots, x_{L-1}, \underbrace{0, 0, \ldots, 0}_{D \text{ zeros}}]$$

with the corresponding DTFTs remaining the same, because $x_D(n) = 0$ for $L \leq n \leq L + D - 1$,

$$X_D(\omega) = \sum_{n=0}^{L+D-1} x_D(n)e^{-j\omega n} = \sum_{n=0}^{L-1} x_D(n)e^{-j\omega n} + \sum_{n=L}^{L+D-1} x_D(n)e^{-j\omega n}$$
$$= \sum_{n=0}^{L-1} x(n)e^{-j\omega n} = X(\omega)$$

Therefore, their evaluation at their N-point DFT frequencies will also be the same: $X_D(\omega_k) = X(\omega_k)$. We note also that padding the D zeros to the *front* of the signal will be equivalent to a delay by D samples, which in the z-domain corresponds to multiplication by z^{-D} and in the frequency domain by $e^{-j\omega D}$. Therefore, the signals:

$$\begin{aligned} \mathbf{x} &= [x_0, x_1, \ldots, x_{L-1}] \\ \mathbf{x}_D &= [\underbrace{0, 0, \ldots, 0}_{D \text{ zeros}}, x_0, x_1, \ldots, x_{L-1}] \end{aligned} \tag{11}$$

will have DTFTs and DFTs:

$$\begin{aligned} X_D(\omega) &= e^{-j\omega D} X(\omega) \\ X_D(\omega_k) &= e^{-j\omega_k D} X(\omega_k), \quad k = 0, 1, \ldots, N-1 \end{aligned} \tag{12}$$

Physical versus Computational Resolution

The bin width, $\Delta f_{\text{bin}} = f_s/N$, represents the spacing between the DFT frequencies at which the DTFT is computed and must not be confused with the frequency resolution width, $\Delta f = f_s/L$, which refers to the minimum resolvable frequency separation between two sinusoidal components (assuming a rectangular window).

To avoid confusion, we will refer to, $\Delta f = f_s/L$, as the *physical* frequency resolution and to, $\Delta f_{\text{bin}} = f_s/N$, as the *computational* frequency resolution.

The interplay between physical and computational resolution is illustrated in Fig. 3 for the triple sinusoidal signal consisting of three equal-strength sinusoids of frequencies $f_1 = 2$ kHz, $f_2 = 2.5$ kHz, and $f_3 = 3$ kHz, where t is in milliseconds and sampled at a rate of 10 kHz,

$$x(t) = \cos(2\pi f_1 t) + \cos(2\pi f_2 t) + \cos(2\pi f_3 t)$$

$$x(t_n) = \cos(2\pi f_1 t_n) + \cos(2\pi f_2 t_n) + \cos(2\pi f_3 t_n)\,, \quad t_n = nT$$

or, in equivalent notation, for a total of L samples, $n = 0, 1, \ldots, L-1$,

$$x(n) = \cos\left(\frac{2\pi f_1}{f_s}\, n\right) + \cos\left(\frac{2\pi f_2}{f_s}\, n\right) + \cos\left(\frac{2\pi f_2}{f_s}\, n\right)$$

The $N = 32$ and $N = 64$ point DFTs of the rectangularly windowed signals of lengths $L = 10$ and $L = 20$ are shown together with their full DTFTs (computed here as 256-point DFTs).

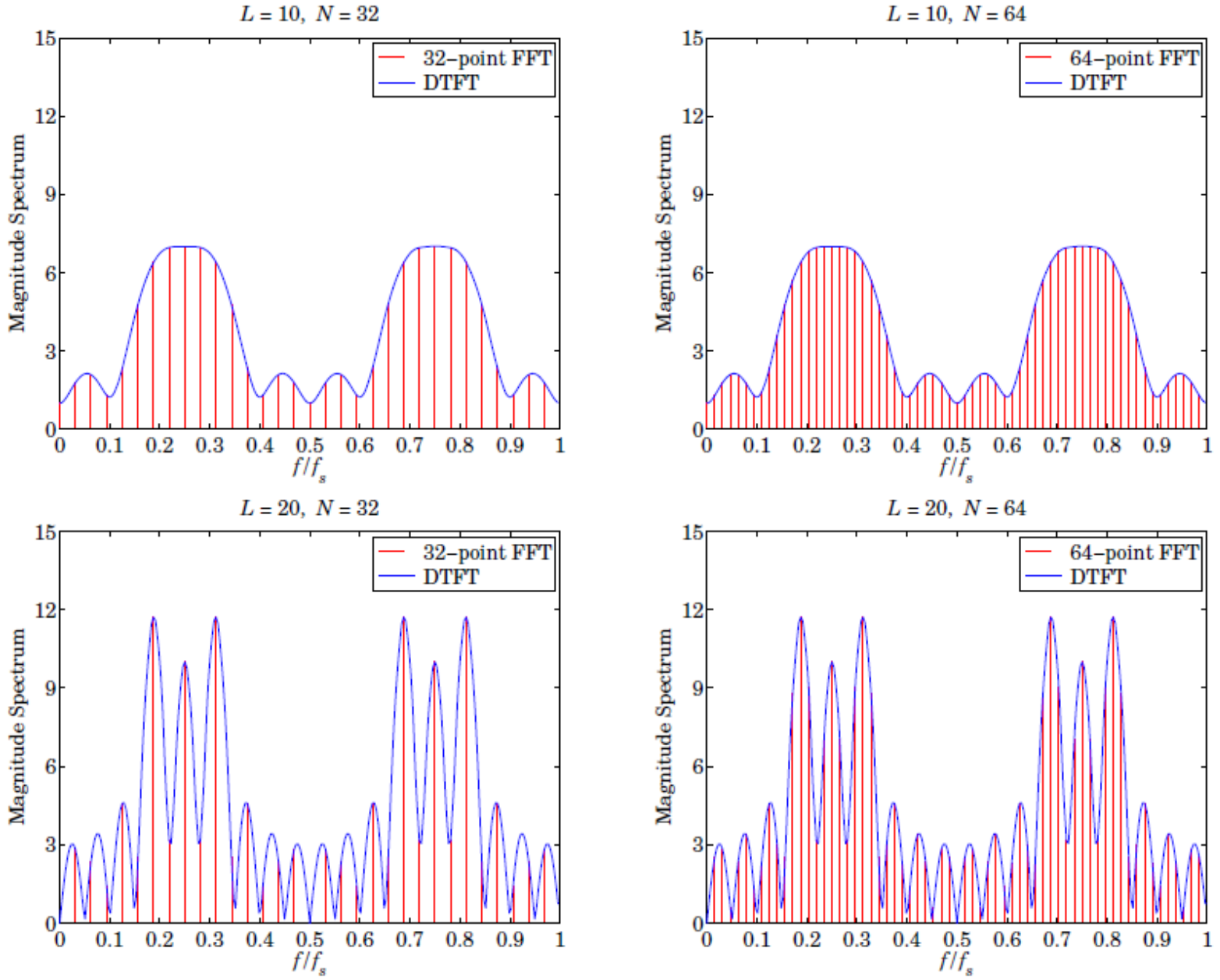


Fig. 3 Physical versus computational resolution in DTFT computation.

It is evident from these graphs that if the length L of the signal is not large enough to provide sufficient physical resolution, then there is no point increasing the length N of the DFT—that would only put more points on the wrong curve.

Another issue related to physical and computational resolution is the question of how accurately the DFT represents the peaks in the spectrum. For each sinusoid that is present in the signal, say, at frequency f_0, the DTFT will exhibit a mainlobe peak arising from the shifted window $W(f - f_0)$.

When we evaluate the N-point DFT, we would like the peak at f_0 to *coincide* with one of the N DFT frequencies (6). This will happen if there is an integer $0 \leq k_0 \leq N - 1$, such that

$$f_0 = f_{k_0} = \frac{k_0 f_s}{N} \quad \Rightarrow \quad \boxed{k_0 = N\frac{f_0}{f_s}} \tag{13}$$

Similarly, the peak at the negative frequency, $-f_0$, or at the equivalent shifted one, $f_s - f_0$, will correspond to the integer, $-k_0$, or to the shifted one $N - k_0$:

$$-f_0 = -k_0\frac{f_s}{N} \quad \Rightarrow \quad f_s - f_0 = f_s - k_0\frac{f_s}{N} = (N - k_0)\frac{f_s}{N}$$

In summary, for each sinusoid with peaks at $\pm f_0$, we would like our DFT to show these peaks at the integers:

$$\{f_0, -f_0\} \quad \Rightarrow \quad \{f_0, f_s - f_0\} \quad \Rightarrow \quad \{k_0, N - k_0\}$$

In general, this is not possible because k_0 computed from Eq. (13) is not an integer, and the DFT will miss the exact peaks. However, for large N, we may *round* k_0 to the nearest integer and use the corresponding DFT frequency as an *estimate* of the actual peak frequency.

A pitfall of using the DFT can be seen in the lower two graphs of Fig. 3, where it appears that the DFT correctly identifies the three peaks in the spectrum, for both $N = 32$ and $N = 64$.

However, this is misleading for two reasons: First, it is a numerical accident in this example that the mainlobe maxima coincide with the DFT frequencies. Second, it can be seen in the figure that these maxima correspond to the *wrong* frequencies and not to the correct ones, which are:

$$\frac{f_1}{f_s} = 0.20, \quad \frac{f_2}{f_s} = 0.25, \quad \frac{f_3}{f_s} = 0.30 \tag{14}$$

This phenomenon, whereby the maxima of the peaks in the spectrum do not quite correspond to the correct frequencies, is called *biasing* and is caused by the lack of adequate *physical resolution*, especially when the sinusoidal frequencies are too close to each other and the sum of terms $W(f - f_0)$ interact strongly.

Using Eq. (13), we can calculate the DFT indices k and $N - k$ to which the true frequencies (14) correspond. For $N = 32$, we have:

$$k_1 = N\frac{f_1}{f_s} = 32 \cdot 0.20 = 6.4, \quad N - k_1 = 25.6$$

$$k_2 = N\frac{f_2}{f_s} = 32 \cdot 0.25 = 8, \quad N - k_2 = 24$$

$$k_3 = N\frac{f_3}{f_s} = 32 \cdot 0.30 = 9.6, \quad N - k_3 = 22.4$$

Similarly, for $N = 64$, we find:

$$k_1 = N\frac{f_1}{f_s} = 64 \cdot 0.20 = 12.8, \quad N - k_1 = 51.2$$

$$k_2 = N\frac{f_2}{f_s} = 64 \cdot 0.25 = 16, \quad N - k_2 = 48$$

$$k_3 = N\frac{f_3}{f_s} = 64 \cdot 0.30 = 19.2, \quad N - k_3 = 44.8$$

Only the middle one at f_2 corresponds to an integer, and therefore, coincides with a DFT value. The other two are missed by the DFT. We may round k_1 and k_3 to their nearest integers and then compute the corresponding DFT frequencies. We find for $N = 32$:

$$k_1 = 6.4 \quad \Rightarrow \quad k_1 = 6 \quad \Rightarrow \quad \frac{f_1}{f_s} = \frac{k_1}{N} = 0.1875$$

$$k_3 = 9.6 \quad \Rightarrow \quad k_3 = 10 \quad \Rightarrow \quad \frac{f_3}{f_s} = \frac{k_3}{N} = 0.3125$$

and for $N = 64$:

$$k_1 = 12.8 \quad \Rightarrow \quad k_1 = 13 \quad \Rightarrow \quad \frac{f_1}{f_s} = \frac{k_1}{N} = 0.203125$$

$$k_3 = 19.2 \quad \Rightarrow \quad k_3 = 19 \quad \Rightarrow \quad \frac{f_3}{f_s} = \frac{k_3}{N} = 0.296875$$

The *rounding error* in the frequencies remains less than $f_s/2N$. It decreases with increasing DFT length N. The *biasing error*, on the other hand, can only be decreased by increasing the data length L.

Figure 4 shows the spectrum of the same signal but with length $L = 100$ samples. Biasing is virtually eliminated with the peak maxima at the correct frequencies. The spectrum is plotted versus the DFT index k, which is proportional to the frequency f via the mapping (6), or,

$$\boxed{k = N\frac{f}{f_s}} \quad \text{(frequency in units of the DFT index)} \tag{15}$$

The Nyquist interval $0 \leq f \leq f_s$ corresponds to the index interval $0 \leq k \leq N$. The N-point DFT is at the integer values $k = 0, 1, \ldots, N-1$.

For plotting purposes, the graph of the spectrum over the full interval $0 \leq k \leq N$ has been split into two side-by-side graphs covering the half-intervals: $0 \leq k \leq N/2$ and $N/2 \leq k \leq N$.

In the upper two graphs having $N = 32$, the DFT misses the f_1 and f_3 peaks completely (the peak positions are indicated by the arrows). The actual peaks are so narrow that they fit completely within the computational resolution width Δf_{bin}.

In the lower two graphs having $N = 64$, the DFT still misses these peaks, but less so. Further doubling of N will interpolate half-way between the frequencies of the 64-point case resulting in a better approximation.

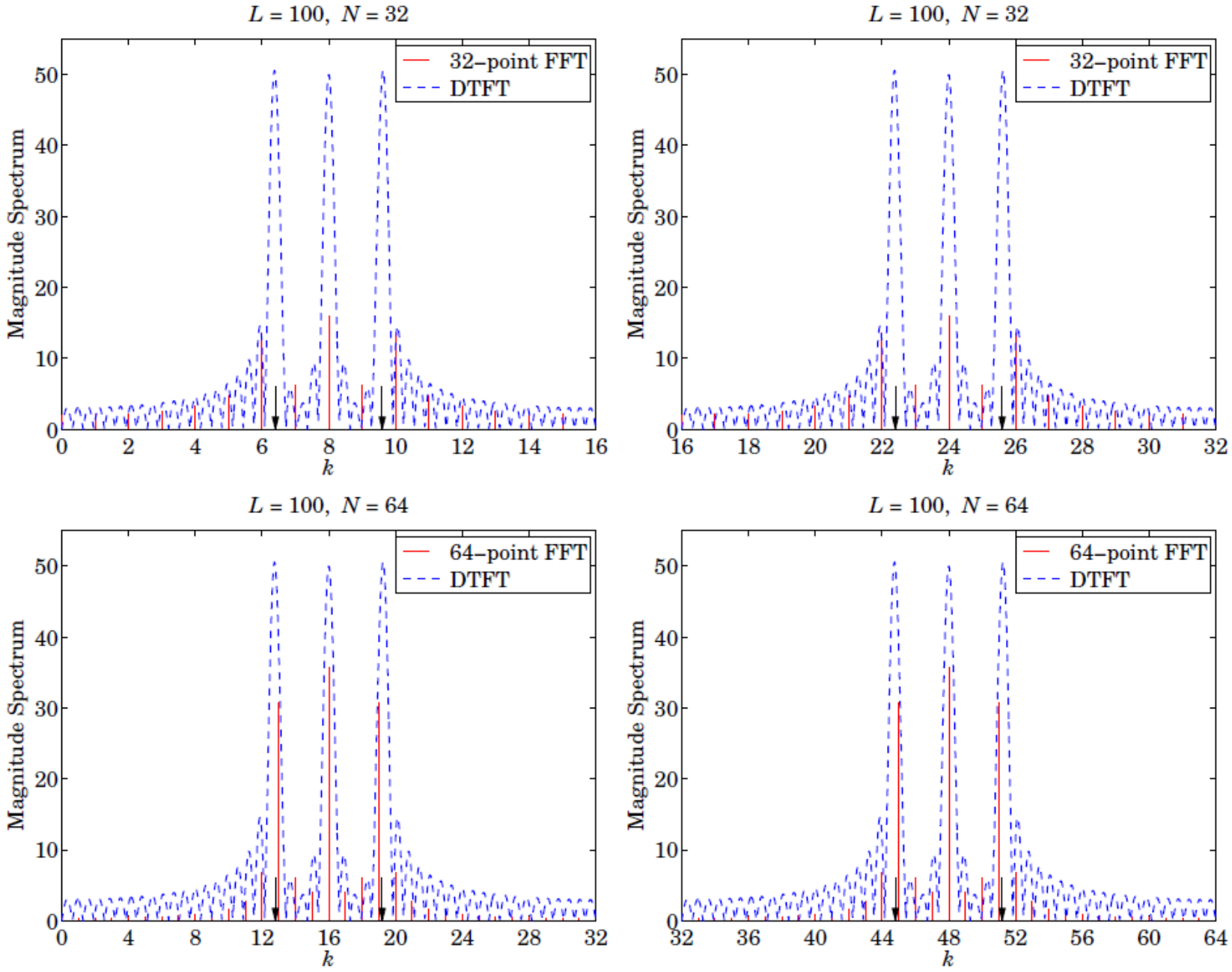


Fig. 4 DFT can miss peaks in the spectrum.

Matrix Form of DFT

The N-point DFT (7) can be thought of as a linear *matrix transformation* of the L-dimensional vector of time data into an N-dimensional vector of frequency data:

$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{L-1} \end{bmatrix} \quad \xrightarrow{\text{DFT}} \quad \mathbf{X} = \begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N-1} \end{bmatrix}$$

with DFT components by $X_k = X(\omega_k)$, $k = 0, 1, \ldots, N-1$. The linear transformation is implemented by an $N \times L$ matrix A, to be referred to as the *DFT matrix*, and can be written compactly as follows:

$$\boxed{\mathbf{X} = \text{DFT}(\mathbf{x}) = A\mathbf{x}} \quad \text{(matrix form of DFT)} \tag{16}$$

or, component-wise:

$$\boxed{X_k = \sum_{n=0}^{L-1} A_{kn} x_n} \quad k = 0, 1, \ldots, N-1 \tag{17}$$

The matrix elements A_{kn} are defined from Eq. (7):

$$A_{kn} = e^{-j\omega_k n} = e^{-2\pi jkn/N} = W_N^{kn}, \qquad \begin{matrix} k = 0, 1, \ldots, N-1 \\ n = 0, 1, \ldots, L-1 \end{matrix} \tag{18}$$

For convenience, we defined the so-called *twiddle factor*, W_N, as the complex number:

$$W_N = e^{-2\pi j/N} \tag{19}$$

Thus, the DFT matrix for an N-point DFT is built from the powers of W_N. Note that the first row $(k = 0)$ and first column $(n = 0)$ of A are always unity:

$$A_{0n} = 1, \quad 0 \le n \le L-1 \qquad \text{and} \qquad A_{k0} = 1, \quad 0 \le k \le N-1$$

The matrix A can be built from its second row $(k = 1)$, consisting of the successive powers of W_N:

$$A_{1n} = W_N^n, \qquad n = 0, 1, \ldots, L-1$$

It follows from the definition that the kth row is obtained by raising the second row to the kth power—element by element:

$$A_{kn} = W_N^{kn} = (W_N^n)^k = A_{1n}^k$$

Some examples of twiddle factors, DFT matrices, and DFTs are as follows: For $L = N$ and $N = 2, 4, 8$, we have:

$$\begin{aligned} W_2 &= e^{-2\pi j/2} = e^{-\pi j} = -1 \\ W_4 &= e^{-2\pi j/4} = e^{-\pi j/2} = \cos(\pi/2) - j\sin(\pi/2) = -j \\ W_8 &= e^{-2\pi j/8} = e^{-\pi j/4} = \cos(\pi/4) - j\sin(\pi/4) = \frac{1-j}{\sqrt{2}} \end{aligned} \tag{20}$$

The corresponding 2-point and 4-point DFT matrices are:

$$A = \begin{bmatrix} 1 & 1 \\ 1 & W_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \tag{21}$$

The 2-point and 4-point DFTs of a length-2 and a length-4 signal will be:

$$\begin{bmatrix} X_0 \\ X_1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} x_0 + x_1 \\ x_0 - x_1 \end{bmatrix}$$

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{22}$$

Thus, the 2-point DFT is formed by taking the sum and difference of the two time samples. We will see later that the 2-point DFT is a convenient starting point for the merging operation in performing the FFT by hand.

The twiddle factor W_N satisfies, $W_N^N = 1$, and therefore it is one of the Nth roots of unity; indeed, in the notation of Eq. (10), it is the root $W_N = z_{N-1}$ and is shown in Fig. 2. Actually, all the successive powers W_N^k, $k = 0, 1, \ldots, N-1$ are Nth roots of unity, but in *reverse order* (i.e., clockwise) than the z_k of Eq. (10):

$$W_N^k = e^{-2\pi jk/N} = z_{-k} = z_k^{-1}, \quad k = 0, 1, \ldots, N-1 \tag{23}$$

Figure 5 shows W_N and its successive powers for the values $N = 2, 4, 8$. Because $W_N^N = 1$, the exponents in W_N^{kn} can be reduced modulo-N, that is, we may replace them by $W_N^{(nk) \bmod (N)}$.

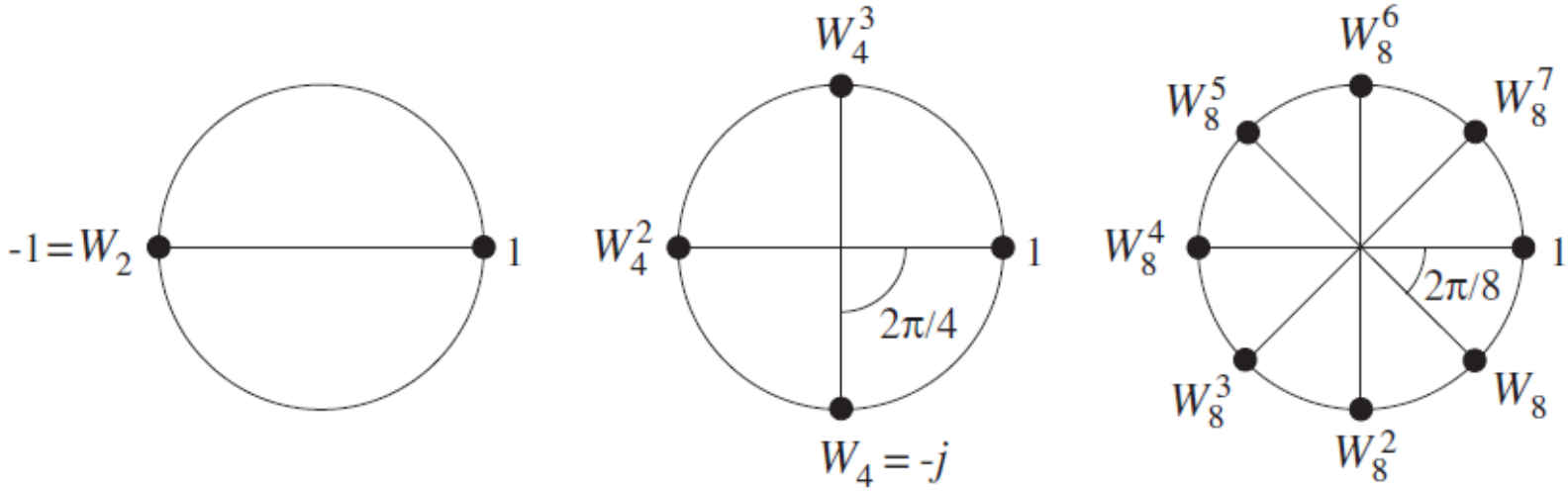


Fig. 5 Twiddle factor lookup tables for $N = 2, 4, 8$.

For example, using the property $W_4^4 = 1$, we may reduce all the powers of W_4 in the 4-point DFT matrix of Eq. (21) to one of the four powers W_4^k, $k = 0, 1, 2, 3$ and write it as

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & W_4^4 & W_4^6 \\ 1 & W_4^3 & W_4^6 & W_4^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & W_4 & W_4^2 & W_4^3 \\ 1 & W_4^2 & 1 & W_4^2 \\ 1 & W_4^3 & W_4^2 & W_4 \end{bmatrix}$$

The entries in A can be read off from the circular lookup table of powers of W_4 in Fig. 5, giving

$$W_4 = -j, \quad W_4^2 = -1, \quad W_4^3 = j$$

and for $N = 8$,

$$W_8 = \frac{1-j}{\sqrt{2}}, \quad W_8^2 = -j, \quad W_8^3 = \frac{-1-j}{\sqrt{2}}, \quad W_8^4 = -1$$

$$W_8^5 = \frac{-1+j}{\sqrt{2}}, \quad W_8^6 = j, \quad W_8^7 = \frac{1+j}{\sqrt{2}}$$

Modulo-N Reduction

The *modulo-N reduction* or *wrapping* of a signal plays a fundamental part in the theory of the DFT. It is defined by dividing the signal $\mathbf{x}$ into contiguous non-overlapping blocks of length N, wrapping the blocks around to be time-aligned with the first block, and adding them up. The process is illustrated in Fig. 6. The resulting wrapped block $\tilde{\mathbf{x}}$ has length N.

The length L of the signal $\mathbf{x}$ could be finite or infinite. If L is not an integral multiple of N, then the last sub-block will have length less than N; in this case, we may pad enough zeros at the end of the last block to increase its length to N.

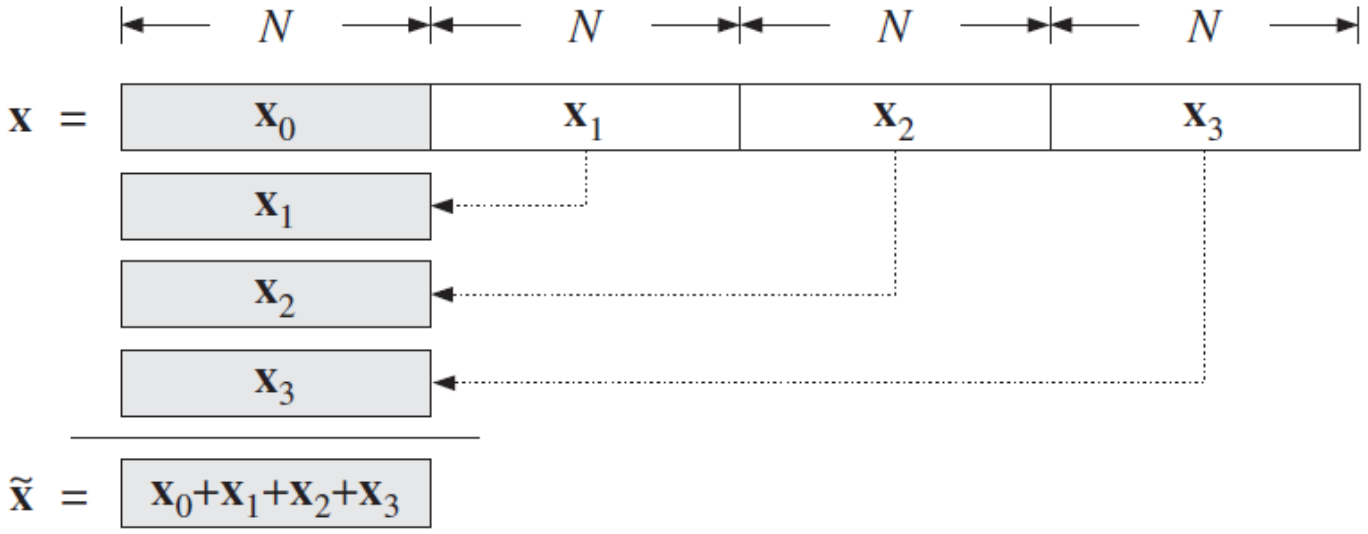


Fig. 6 Modulo-N reduction of a signal.

The wrapping process can also be thought of as *partitioning* the signal vector $\mathbf{x}$ into N-dimensional sub-vectors and adding them up. For example, if $L = 4N$, the signal $\mathbf{x}$ will consist of four length-N sub-vectors:

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} \quad \Rightarrow \quad \widetilde{\mathbf{x}} = \mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3 \tag{24}$$

Example. Determine the mod-4 and mod-3 reductions of the length-8 signal vector:

$$\mathbf{x} = [1,\ 2,\ -2,\ 3,\ 4,\ -2,\ -1,\ 1]^T$$

For the $N = 4$ case, we may divide $\mathbf{x}$ into two length-4 sub-blocks to get:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \\ -2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ -2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

Similarly, for $N = 3$ we divide $\mathbf{x}$ into length-3 blocks:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} 1 \\ 2 \\ -2 \end{bmatrix} + \begin{bmatrix} 3 \\ 4 \\ -2 \end{bmatrix} + \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 7 \\ -4 \end{bmatrix}$$

where we padded a zero at the end of the third sub-block. □

We may express the sub-block components in terms of the time samples of the signal $x(n)$, $0 \leq n \leq L-1$, as follows. For $m = 0, 1, \ldots$

$$x_m(n) = x(mN + n), \quad n = 0, 1, \ldots, N-1 \tag{25}$$

Thus, the mth sub-block occupies the time interval $\big[mN,\ (m+1)N\big)$. The wrapped vector $\widetilde{\mathbf{x}}$ will be in this notation:

$$\begin{aligned} \widetilde{x}(n) &= x_0(n) + x_1(n) + x_2(n) + x_3(n) + \cdots \\ &= x(n) + x(N+n) + x(2N+n) + x(3N+n) + \cdots \end{aligned} \tag{26}$$

for $n = 0, 1, \ldots, N-1$, or, more compactly

$$\boxed{\widetilde{x}(n) = \sum_{m=0}^{\infty} x(mN+n)}, \qquad n = 0, 1, \ldots, N-1 \tag{27}$$

This expression can be used to define $\widetilde{x}(n)$ for all n, not just $0 \leq n \leq N-1$. The resulting double-sided infinite signal is the so-called *periodic extension* of the signal $x(n)$ with period N. More generally, it is defined by

$$\widetilde{x}(n) = \sum_{m=-\infty}^{\infty} x(mN+n), \qquad -\infty < n < \infty \tag{28}$$

The signal $\widetilde{x}(n)$ is periodic in n with period N, that is, $\widetilde{x}(n+N) = \widetilde{x}(n)$. The definition (27) evaluates only one basic period $0 \le n \le N-1$ of $\widetilde{x}(n)$, which is all that is needed in the DFT.

The periodic extension interpretation of mod-N reduction is shown in Fig. 7. The terms $x(n+N)$, $x(n+2N)$, and $x(n+3N)$ of Eq. (26) can be thought as the time-advanced or left-shifted versions of $x(n)$ by N, $2N$, and $3N$ time samples. The successive sub-blocks of $x(n)$ get time-aligned one under the other over the basic period $0 \le n \le N-1$, thus, their sum is the wrapped signal $\widetilde{\mathbf{x}}$.

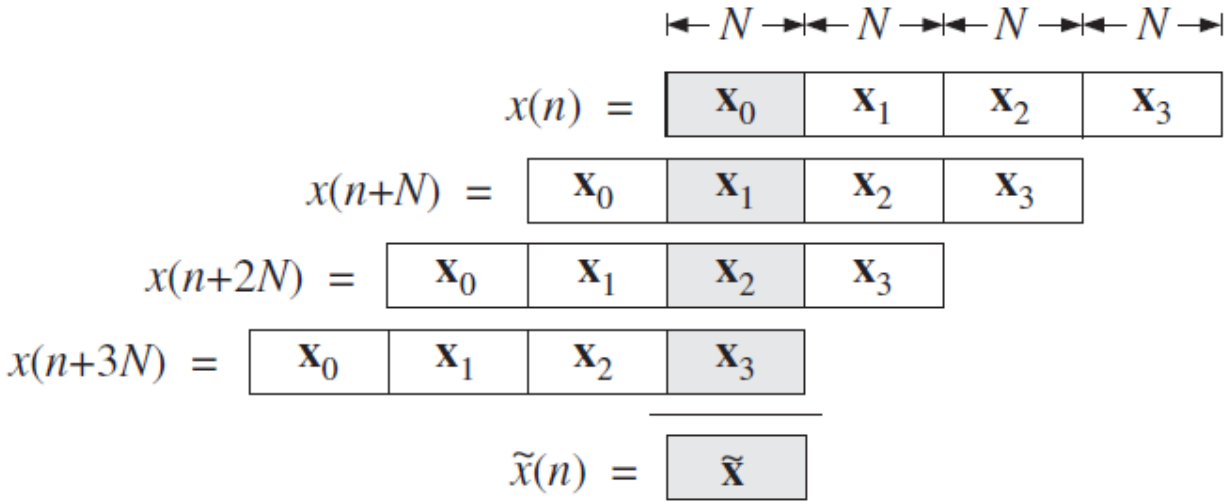


Fig. 7 Periodic extension interpretation of mod-N reduction of a signal.

The connection of the mod-N reduction to the DFT is the theorem that the length-N wrapped signal $\widetilde{\mathbf{x}}$ has the *same* N-point DFT as the original unwrapped signal $\mathbf{x}$, that is,

$$\boxed{\widetilde{X}_k = X_k \quad \text{or,} \quad \widetilde{X}(\omega_k) = X(\omega_k)}, \qquad k = 0, 1, \ldots, N-1 \tag{29}$$

where $\widetilde{X}_k = \widetilde{X}(\omega_k)$ is the N-point DFT of the length-N signal $\widetilde{x}(n)$:

$$\widetilde{X}_k = \widetilde{X}(\omega_k) = \sum_{n=0}^{N-1} \widetilde{x}(n) e^{-j\omega_k n}, \qquad k = 0, 1, \ldots, N-1 \tag{30}$$

In the notation of Eq. (17), we may write:

$$\widetilde{X}_k = \sum_{n=0}^{N-1} W_N^{kn}\, \widetilde{x}(n) = \sum_{n=0}^{N-1} \widetilde{A}_{kn}\, \widetilde{x}(n) \tag{31}$$

where $\widetilde{A}$ is the DFT matrix defined as in Eq. (18):

$$\widetilde{A}_{kn} = W_N^{kn}, \qquad 0 \le k \le N-1, \quad 0 \le n \le N-1 \tag{32}$$

The DFT matrices A and $\widetilde{A}$ have the same definition, except they differ in their dimensions, which are $N \times L$ and $N \times N$, respectively. We can write the DFT of $\widetilde{\mathbf{x}}$ in the compact matrix form:

$$\widetilde{\mathbf{X}} = \text{DFT}(\widetilde{\mathbf{x}}) = \widetilde{A}\,\widetilde{\mathbf{x}} \tag{33}$$

Thus, the above theorem can be stated in vector form:

$$\widetilde{\mathbf{X}} = \mathbf{X} = A\,\mathbf{x} = \widetilde{A}\,\widetilde{\mathbf{x}} \tag{34}$$

Symbolically, we will write $\text{DFT}(\widetilde{\mathbf{x}}) = \text{DFT}(\mathbf{x})$ to denote Eqs. (29) or (34). The above theorem can be proved in many ways. In matrix form, it follows from the property that the $N \times N$ submatrices of the full $N \times L$ DFT matrix A are all *equal* to the DFT matrix $\widetilde{A}$.

These submatrices are formed by grouping the first N columns of A into the first submatrix, the next N columns into the second submatrix, and so on. The matrix elements of the mth submatrix will be:

$$A_{k,mN+n} = W_N^{k(mN+n)} = W_N^{mkN} W_N^{kn}$$

Using the property $W_N^N = 1$, it follows that $W_N^{kmN} = 1$, and therefore:

$$A_{k,mN+n} = W_N^{kn} = A_{kn} = \widetilde{A}_{kn}, \qquad 0 \leq k, n \leq N-1$$

Thus, in general, A is partitioned in the form:

$$A = [\widetilde{A},\ \widetilde{A},\ \widetilde{A}, \ldots] \tag{35}$$

As an example, consider the case $L = 8$, $N = 4$. The 4×8 DFT matrix A can be partitioned into two 4×4 identical submatrices, which are equal to $\widetilde{A}$. Using $W_4 = e^{-2\pi j/4} = -j$, we have:

$$
\begin{aligned}
A &= \left[\begin{array}{cccc|cccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & W_4 & W_4^2 & W_4^3 & W_4^4 & W_4^5 & W_4^6 & W_4^7 \\
1 & W_4^2 & W_4^4 & W_4^6 & W_4^8 & W_4^{10} & W_4^{12} & W_4^{14} \\
1 & W_4^3 & W_4^6 & W_4^9 & W_4^{12} & W_4^{15} & W_4^{18} & W_4^{21}
\end{array}\right] \\
&= \left[\begin{array}{cccc|cccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & W_4 & W_4^2 & W_4^3 & 1 & W_4 & W_4^2 & W_4^3 \\
1 & W_4^2 & W_4^4 & W_4^6 & 1 & W_4^2 & W_4^4 & W_4^6 \\
1 & W_4^3 & W_4^6 & W_4^9 & 1 & W_4^3 & W_4^6 & W_4^9
\end{array}\right] \\
&= \left[\begin{array}{cccc|cccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & j & -1 & -j & 1 & j & -1 & -j
\end{array}\right] = [\widetilde{A}, \widetilde{A}]
\end{aligned}
\tag{36}
$$

where in the second submatrix, we partially reduced the powers of W_4 modulo-4.

The proof of the theorem follows now as a simple consequence of this partitioning property. For example, we have for the N-point DFT of Eq. (24):

$$\mathbf{X} = A\,\mathbf{x} = [\widetilde{A},\ \widetilde{A},\ \widetilde{A},\ \widetilde{A}] \begin{bmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix} = \widetilde{A}\,\mathbf{x}_0 + \widetilde{A}\,\mathbf{x}_1 + \widetilde{A}\,\mathbf{x}_3 + \widetilde{A}\,\mathbf{x}_3$$

$$= \widetilde{A}\,(\mathbf{x}_0 + \mathbf{x}_1 + \mathbf{x}_2 + \mathbf{x}_3) = \widetilde{A}\,\widetilde{\mathbf{x}} = \widetilde{\mathbf{X}}$$

Figure 8 illustrates the relative dimensions of these operations. The DFT (33) of $\widetilde{\mathbf{x}}$ requires N^2 complex multiplications, whereas that of $\mathbf{x}$ requires NL. Thus, if $L > N$, it is more efficient to first wrap the signal mod-N and then take its DFT.

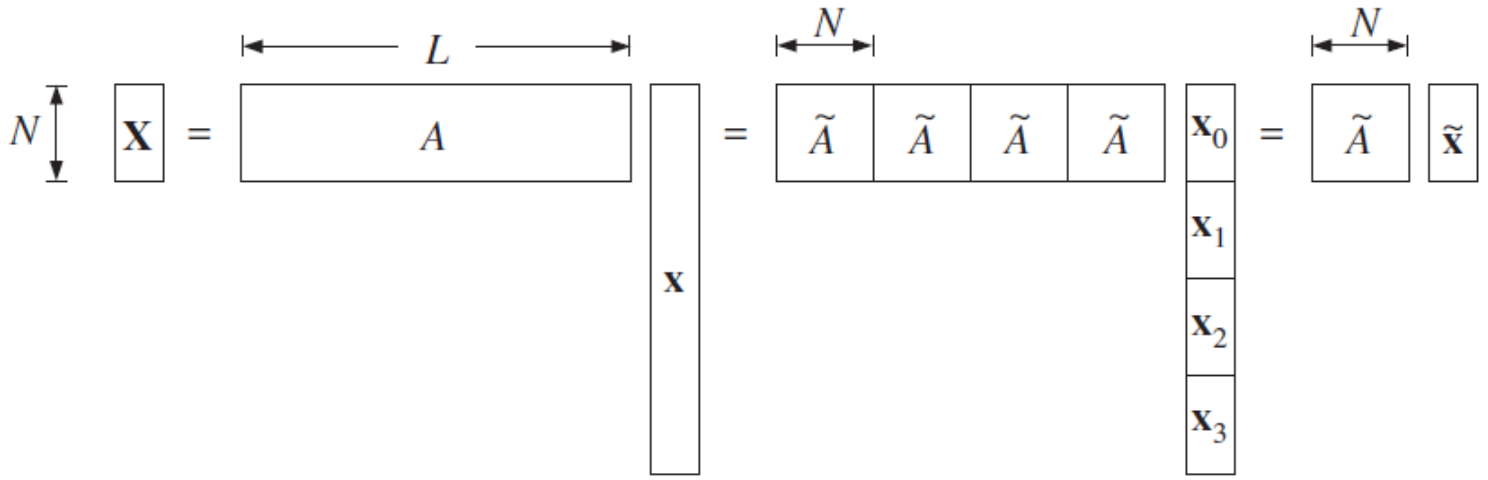


Fig. 8 N-point DFTs of the full and wrapped signals are equal.

Example. Compute the 4-point DFT of the length-8 signal

$$\mathbf{x} = [1,\ 2,\ -2,\ 3,\ 4,\ -2,\ -1,\ 1]^T$$

in two ways: (a) working with the full unwrapped vector $\mathbf{x}$ and (b) computing the DFT of its mod-4 reduction.

Solution: The 4×8 DFT matrix was worked out above, resulting in:

$$\mathbf{X} = A\,\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -2 \\ 3 \\ 4 \\ -2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix}$$

The same DFT can be computed by the DFT matrix $\widetilde{A}$ acting on the mod-4 wrapped signal $\widetilde{\mathbf{x}}$ that we determined previously:

$$\widetilde{\mathbf{X}} = \widetilde{A}\,\widetilde{\mathbf{x}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix}$$

The two methods give identical results. □

Example. The length L of the signal $\mathbf{x}$ can be infinite, as long as the signal is stable, so that the sum (27) converges. To illustrate the theorem (29) or (34), consider the causal signal $x(n) = a^n u(n)$, where $|a| < 1$.
To compute its N-point DFT, we determine its z-transform and evaluate it at the Nth root of unity points $z_k = e^{j\omega_k} = e^{2\pi jk/N}$. This gives:

$$X(z) = \frac{1}{1 - az^{-1}} \quad \Rightarrow \quad X_k = X(z_k) = \frac{1}{1 - az_k^{-1}}, \quad k = 0, 1, \ldots, N-1$$

Next, we compute its mod-N reduction by the sum (27):

$$\widetilde{x}(n) = \sum_{m=0}^{\infty} x(mN + n) = \sum_{m=0}^{\infty} a^{mN} a^n = \frac{a^n}{1 - a^N}, \quad n = 0, 1, \ldots, N-1$$

where we used the geometric series sum. Computing its z-transform, we find:

$$\widetilde{X}(z) = \sum_{n=0}^{N-1} \widetilde{x}(n) z^{-n} = \frac{1}{1 - a^N} \sum_{n=0}^{N-1} a^n z^{-n} = \frac{1 - a^N z^{-N}}{(1 - a^N)(1 - az^{-1})}$$

Evaluating it at $z = z_k$ and using the property that $z_k^N = 1$, we find

$$\widetilde{X}_k = \frac{1 - a^N z_k^{-N}}{(1 - a^N)(1 - az_k^{-1})} = \frac{1 - a^N}{(1 - a^N)(1 - az_k^{-1})} = \frac{1}{1 - az_k^{-1}} = X_k$$

Thus, even though $x(n)$ and $\widetilde{x}(n)$ are different and have different z-transforms and DTFTs, their N-point DFTs are the same. $\square$

The built-in function **datawrap** implements the wrapping process. Using this function, we may compute the N-point DFT of a length-L signal by first wrapping it modulo-N and then computing the N-point DFT or FFT of the wrapped signal:

```
xtilde = datawrap(x,N);
X = fft(xtilde, N);
```

The two signals $\mathbf{x}$ and $\widetilde{\mathbf{x}}$ are not the only ones that have a common DFT. Any other signal that has the *same* mod-N reduction as $\mathbf{x}$ will have the same DFT as $\mathbf{x}$. To see this, consider a length-L signal $\mathbf{y}$ such that $\widetilde{\mathbf{y}} = \widetilde{\mathbf{x}}$; then its N-point DFT can be obtained by applying Eq. (34):

$$\mathbf{Y} = A\,\mathbf{y} = \widetilde{A}\,\widetilde{\mathbf{y}} = \widetilde{A}\,\widetilde{\mathbf{x}} = A\,\mathbf{x} = \mathbf{X}$$

For example, the following length-8 signals all have the same 4-point DFT,

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}, \quad \begin{bmatrix} x_0 + x_4 \\ x_1 \\ x_2 \\ x_3 \\ 0 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}, \quad \begin{bmatrix} x_0 + x_4 \\ x_1 + x_5 \\ x_2 \\ x_3 \\ 0 \\ 0 \\ x_6 \\ x_7 \end{bmatrix}, \quad \begin{bmatrix} x_0 + x_4 \\ x_1 + x_5 \\ x_2 + x_6 \\ x_3 \\ 0 \\ 0 \\ 0 \\ x_7 \end{bmatrix}, \quad \begin{bmatrix} x_0 + x_4 \\ x_1 + x_5 \\ x_2 + x_6 \\ x_3 + x_7 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

because all have the same mod-4 reduction:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} x_0 + x_4 \\ x_1 + x_5 \\ x_2 + x_6 \\ x_3 + x_7 \end{bmatrix}$$

The above signals have a bottom half that becomes progressively zero, until the last vector which is recognized as the $\widetilde{\mathbf{x}}$, viewed as a length-8 vector. In fact, the mod-N wrapped signal $\widetilde{\mathbf{x}}$ is *unique* in the above class of signals in the sense that it is *shortest* signal, that is, of length N, that has the same DFT as the signal $\mathbf{x}$.

An equivalent characterization of the class of signals that have a common DFT can be given in the z-domain. Suppose the length-L signals $\mathbf{y}$ and $\mathbf{x}$ have equal mod-N reductions, $\widetilde{\mathbf{y}} = \widetilde{\mathbf{x}}$ and, therefore, equal DFTs $X_k = Y_k$. We form the difference of their z-transforms:

$$F(z) = X(z) - Y(z) = \sum_{n=0}^{L-1} x(n)z^{-n} - \sum_{n=0}^{L-1} y(n)z^{-n}$$

Evaluating $F(z)$ at the Nth roots of unity and using the equality of their N-point DFTs, we find:

$$F(z_k) = X(z_k) - Y(z_k) = X_k - Y_k = 0, \qquad k = 0, 1, \ldots, N-1$$

Thus, the N complex numbers $z_k = e^{j\omega_k} = e^{2\pi jk/N}$ are roots of the difference polynomial $F(z)$. Therefore, $F(z)$ will be divisible by the Nth order product polynomial:

$$1 - z^{-N} = \prod_{k=0}^{N-1} (1 - z_k z^{-1})$$

which represents the factorization of $1 - z^{-N}$ into its Nth root-of-unity zeros. Therefore, we can write:

$$X(z) - Y(z) = F(z) = \left(1 - z^{-N}\right) Q(z) \quad \text{or,}$$

$$X(z) = Y(z) + \left(1 - z^{-N}\right) Q(z) \tag{37}$$

Because $X(z)$ and $Y(z)$ have degree $L-1$, it follows that $Q(z)$ is an *arbitrary* polynomial of degree $L-1-N$. Denoting the coefficients of $Q(z)$ by $q(n)$, $0 \leq n \leq L-1-N$, we may write Eq. (37) in the time domain:

$$\boxed{x(n) = y(n) + q(n) - q(n-N)} \quad n = 0, 1, \ldots, L-1 \qquad (38)$$

Thus, any two sequences $x(n)$ and $y(n)$ related by Eq. (38) will have the same N-point DFT. The mod-N reduction $\widetilde{\mathbf{x}}$ and its z-transform $\widetilde{X}(z)$ are also related by Eq. (37):

$$\boxed{X(z) = \left(1 - z^{-N}\right) Q(z) + \widetilde{X}(z)} \qquad (39)$$

Because $\widetilde{X}(z)$ has degree $N-1$, Eq. (39) represents the division of the polynomial $X(z)$ by the DFT polynomial $1 - z^{-N}$, with $\widetilde{X}(z)$ being the *remainder* polynomial and $Q(z)$ the *quotient* polynomial. The remainder $\widetilde{X}(z)$ is the unique polynomial satisfying Eq. (39) that has *minimal* degree $N-1$.

Inverse DFT

The problem of inverting an N-point DFT is the problem of recovering the original length-L signal $\mathbf{x}$ from its N-point DFT $\mathbf{X}$, that is, inverting the relationship:

$$\mathbf{X} = A\,\mathbf{x} = \widetilde{A}\,\widetilde{\mathbf{x}} \tag{40}$$

When $L > N$, the matrix A is not invertible. As we saw, there are in this case several possible solutions $\mathbf{x}$, all satisfying Eq. (40) and having the same mod-N reduction $\widetilde{\mathbf{x}}$.

Among these solutions, the only one that is uniquely obtainable from the knowledge of the DFT vector $\mathbf{X}$ is $\widetilde{\mathbf{x}}$. The corresponding DFT matrix $\widetilde{A}$ is an $N \times N$ square invertible matrix. Thus, we define the *inverse DFT* by

$$\boxed{\widetilde{\mathbf{x}} = \mathrm{IDFT}(\mathbf{X}) = \widetilde{A}^{-1}\,\mathbf{X}} \quad \text{(inverse DFT)} \tag{41}$$

or, component-wise,

$$\widetilde{x}_n = \sum_{k=0}^{N-1} (\widetilde{A}^{-1})_{nk} X_k \,, \qquad n = 0, 1, \ldots, N-1 \tag{42}$$

The inverse $\widetilde{A}^{-1}$ can be obtained *without* having to perform a matrix inversion by using the following *unitarity* property of the DFT matrix $\widetilde{A}$:

$$\boxed{\frac{1}{N}\, \widetilde{A}\, \widetilde{A}^* = I_N} \tag{43}$$

where I_N is the N-dimensional identity matrix and $\widetilde{A}^*$ is the complex conjugate of $\widetilde{A}$, obtained by conjugating *every* matrix element of $\widetilde{A}$. For example, for $N = 4$, we can verify easily:

$$\frac{1}{4}\, \widetilde{A}\, \widetilde{A}^* = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Multiplying both sides of Eq. (43) by $\widetilde{A}^{-1}$, we obtain for the matrix inverse:

$$\boxed{\widetilde{A}^{-1} = \frac{1}{N}\,\widetilde{A}^{*}} \tag{44}$$

Thus, the IDFT (41) can be written in the form:

$$\boxed{\widetilde{\mathbf{x}} = \mathrm{IDFT}(\mathbf{X}) = \frac{1}{N}\,\widetilde{A}^{*}\,\mathbf{X}} \quad \text{(inverse DFT)} \tag{45}$$

We note also that the IDFT can be thought of as a DFT in the following sense. Introducing a second conjugation oparation, we have:

$$\widetilde{A}^* \, \mathbf{X} = (\widetilde{A} \, \mathbf{X}^*)^* = \left[\mathrm{DFT}(\mathbf{X}^*)\right]^*$$

where the matrix $\widetilde{A}$ acting on the conjugated vector $\mathbf{X}^*$ is the DFT of that vector. Dividing by N, we have:

$$\boxed{\mathrm{IDFT}(\mathbf{X}) = \frac{1}{N}\left[\mathrm{DFT}(\mathbf{X}^*)\right]^*} \tag{46}$$

Replacing DFT by FFT, we get a convenient inverse FFT formula, which uses an FFT to perform the IFFT. It is used in most FFT routines.

$$\boxed{\mathrm{IFFT}(\mathbf{X}) = \frac{1}{N}\left[\mathrm{FFT}(\mathbf{X}^*)\right]^*} \tag{47}$$

Example. To illustrate Eqs. (45) and (46), we calculate the IDFT of the 4-point DFT of that we found in a previous example, that is,

$$\mathbf{X} = A\,\mathbf{x} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -2 \\ 3 \\ 4 \\ -2 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix}$$

The same DFT can be computed by the DFT matrix $\widetilde{A}$ acting on the mod-4 wrapped signal $\widetilde{\mathbf{x}}$ that we determined previously:

$$\widetilde{\mathbf{X}} = \widetilde{A}\,\widetilde{\mathbf{x}} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix}$$

The inverse DFT is then,

$$\widetilde{\mathbf{x}} = \text{IDFT}(\mathbf{X}) = \frac{1}{N}\widetilde{A}^{*}\,\mathbf{X} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix} \begin{bmatrix} 6 \\ 8+4j \\ -2 \\ 8-4j \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ -3 \\ 4 \end{bmatrix}$$

Using Eq. (18) the matrix elements of $\widetilde{A}^{-1}$ are:

$$(\widetilde{A}^{-1})_{nk} = \frac{1}{N}\,\widetilde{A}^{*}_{nk} = \frac{1}{N}\,(W_N^{nk})^{*} = \frac{1}{N}\,W_N^{-nk}$$

where we used the property $W_N^{*} = e^{2\pi j/N} = W_N^{-1}$. Then, Eq. (42) can be written in the form:

$$\text{(IDFT)} \qquad \boxed{\widetilde{x}_n = \frac{1}{N}\sum_{k=0}^{N-1} W_N^{-nk} X_k} \quad n = 0, 1, \ldots, N-1 \qquad (48)$$

In terms of the DFT frequencies ω_k, we have $X_k = X(\omega_k)$ and

$$W_N^{-nk} = e^{2\pi jkn/N} = e^{j\omega_k n}$$

Therefore, the inverse DFT can be written in the alternative form:

$$\text{(IDFT)} \qquad \boxed{\widetilde{x}(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(\omega_k) e^{j\omega_k n}} \quad n = 0, 1, \ldots, N-1 \qquad (49)$$

DFS

Given any time-periodic signal, $\widetilde{x}(n)$, with period N, such as the one constructed via the periodic extension,

$$\widetilde{x}(n) = \sum_{m=-\infty}^{\infty} x(mN+n)\,, \quad -\infty < n < \infty$$

Its N-point IDFT can be thought of as its Fourier series expansion, referred to as discrete Fourier series (DFS), with the DFT coefficients being the corresponding Fourier series coefficients, albeit only N of them are required to re-build the periodic signal $\widetilde{x}(n)$,

$$\text{(DFS)} \qquad \boxed{X(\omega_k) = \sum_{n=0}^{N-1} \widetilde{x}(n)\, e^{-j\omega_k n}} \quad k = 0, 1, \ldots, N-1$$

$$\text{(IDFS)} \qquad \boxed{\widetilde{x}(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(\omega_k) e^{j\omega_k n}} \quad n = 0, 1, \ldots, N-1$$

It expresses the signal $\widetilde{x}(n)$ as a sum of N complex sinusoids of frequencies ω_k, whose *relative amplitudes and phases* are given by the DFT values $X(\omega_k)$.

The forward DFT of Eq. (7) is sometimes called an *analysis transform*, analyzing a signal $x(n)$ into N Fourier components. The inverse DFT (49) is called a *synthesis transform*, re-synthesizing the signal $\widetilde{x}(n)$ from those Fourier components. The forward and inverse N-point DFTs are akin to the more general forward and inverse DTFTs that use all frequencies, not just the N DFT frequencies:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)e^{-j\omega n}, \qquad x(n) = \int_0^{2\pi} X(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi} \tag{50}$$

The difference between the inverse DTFT and (49) is that Eq. (50) reconstructs the full original signal $x(n)$, whereas (49) reconstructs only the wrapped signal $\widetilde{x}(n)$. Eq. (49) can be thought of as a numerical approximation of the integral in (50), obtained by dividing the integration range into N equal bins:

$$\int_0^{2\pi} X(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi} \simeq \sum_{k=0}^{N-1} X(\omega_k)e^{j\omega_k n}\,\frac{\Delta\omega_{\text{bin}}}{2\pi}$$

where from the definition (8), we have $\Delta\omega_{\text{bin}}/2\pi = 1/N$.

In summary, the inverse of an N-point DFT reconstructs only the wrapped version of the original signal that was transformed.

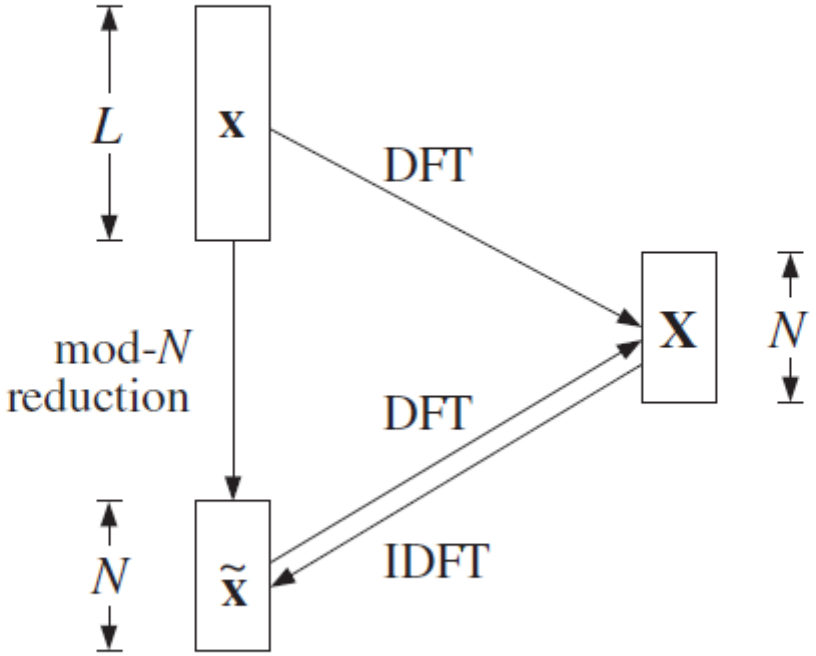


Fig. 9 Forward and inverse N-point DFTs.

In order for the IDFT to generate the original unwrapped signal $\mathbf{x}$, it is necessary to have $\widetilde{\mathbf{x}} = \mathbf{x}$. This happens only if the DFT length N is at least L, so that there will be only one length-N sub-block in $\mathbf{x}$ and there will be nothing to wrap around. Thus, we have the condition:

$$\boxed{\widetilde{\mathbf{x}} = \mathbf{x} \quad \text{only if} \quad N \geq L} \tag{51}$$

If $N = L$, then Eq. (51) is exact. If $N > L$, then we must pad $N-L$ zeros at the end of $\mathbf{x}$ so that the two sides of Eq. (51) have compatible lengths. If $N < L$, the wrapped and original signals will be different because there will be several length-N sub-blocks in $\mathbf{x}$

$$\boxed{\widetilde{\mathbf{x}} \neq \mathbf{x} \quad \text{if} \quad N < L} \tag{52}$$

FFT

The *fast Fourier transform* is a fast implementation of the DFT. It is based on a divide-and-conquer approach in which the DFT computation is divided into smaller, simpler, problems and the final DFT is rebuilt from the simpler DFTs. For a comprehensive review, history, and recent results, see the I2SP references [223-244, 303].

Another application of this divide-and-conquer approach is the computation of *very large FFTs*, in which the time data and their DFT are too large to be stored in main memory. In such cases the FFT is done in parts and the results are pieced together to form the overall FFT, and saved in secondary storage such as on hard disk.

In the simplest Cooley-Tukey version of the FFT, the dimension of the DFT is successively divided in half until it becomes unity. This requires the initial dimension N to be a power of two:

$$\boxed{N = 2^B} \qquad \Rightarrow \qquad B = \log_2(N) \tag{53}$$

The problem of computing the N-point DFT is replaced by the simpler problems of computing two $(N/2)$-point DFTs. Each of these is replaced by two $(N/4)$-point DFTs, and so on.

We will see shortly that an N-point DFT can be rebuilt from two $(N/2)$-point DFTs by an *additional* cost of $N/2$ complex multiplications. This basic merging step is shown in Fig. 10.

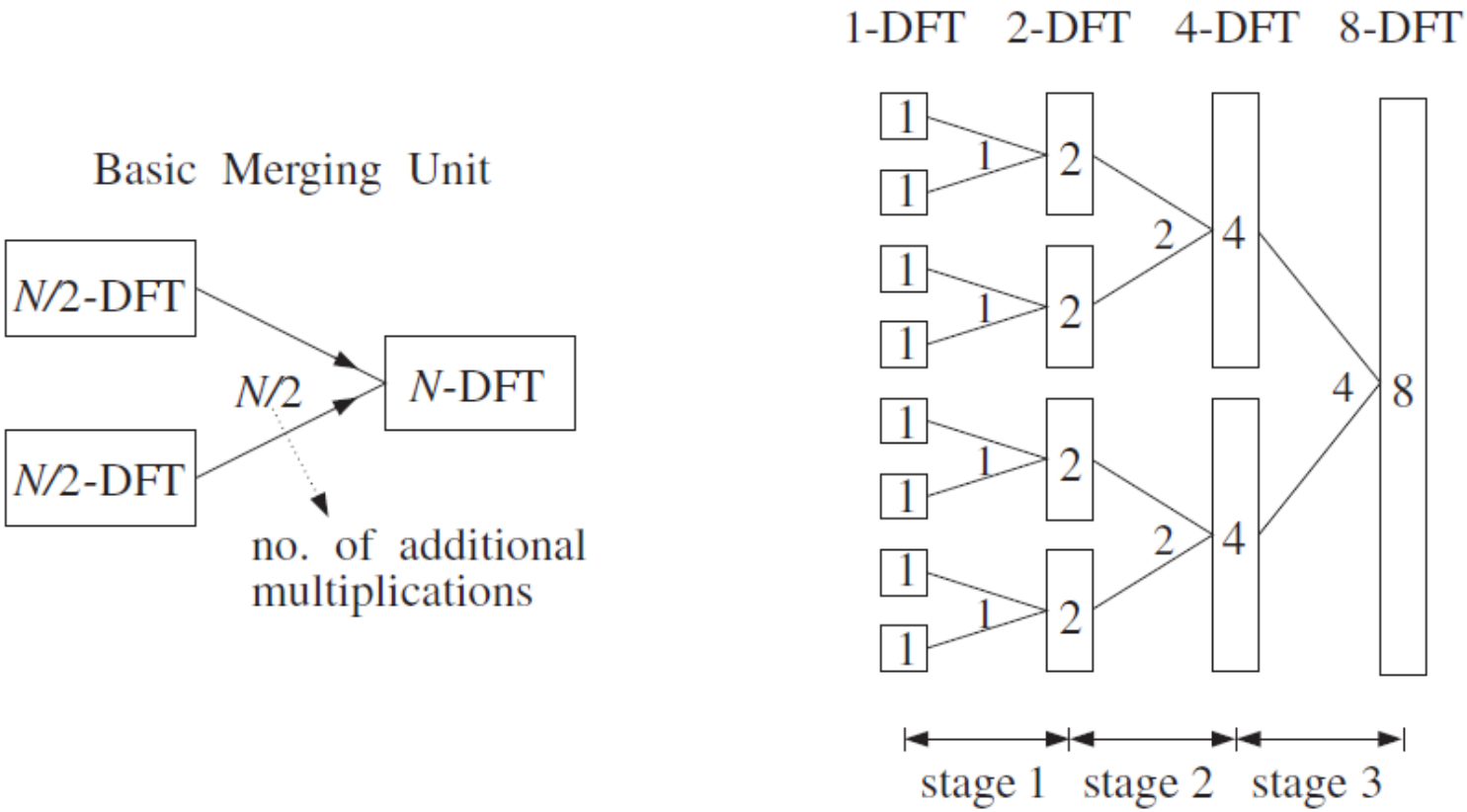


Fig. 10 Merging two $N/2$-DFTs into an N-DFT and its repeated application.

Thus, if we compute the two $(N/2)$-DFTs directly, at a cost of $(N/2)^2$ multiplications each, the total cost of rebuilding the full N-DFT will be:

$$2\left(\frac{N}{2}\right)^2 + \frac{N}{2} = \frac{N^2}{2} + \frac{N}{2} \approx \frac{N^2}{2}$$

where for large N the quadratic term dominates. This amounts to 50 percent savings over computing the N-point DFT directly at a cost of N^2.

Similarly, if the two $(N/2)$-DFTs were computed indirectly by rebuilding each of them from two $(N/4)$-DFTs, the total cost for rebuilding an N-DFT would be:

$$4\left(\frac{N}{4}\right)^2 + 2\frac{N}{4} + \frac{N}{2} = \frac{N^2}{4} + 2\,\frac{N}{2} \simeq \frac{N^2}{4}$$

Thus, we gain another factor of two, or a factor of four in efficiency over the direct N-point DFT. In the above equation, there are 4 direct $(N/4)$-DFTs at a cost of $(N/4)^2$ each, requiring an additional cost of $N/4$ each to merge them into $(N/2)$-DFTs, which require another $N/2$ for the final merge.

Proceeding in a similar fashion, we can show that if we start with $(N/2^m)$-point DFTs and perform m successive merging steps,

$$\frac{N^2}{2^m} + \frac{N}{2}m \tag{54}$$

The first term, $N^2/2^m$, corresponds to performing the initial $(N/2^m)$-point DFTs directly. Because there are 2^m of them, they will require a total cost of $2^m(N/2^m)^2 = N^2/2^m$.

However, if the subdivision process is continued for $m = B$ stages, as shown in Fig. 10, the final dimension will be $N/2^m = N/2^B = 1$, which requires no computation at all because the 1-point DFT of a 1-point signal is itself.

In this case, the first term in Eq. (54) will be absent, and the total cost will arise from the second term. Thus, carrying out the subdivision/merging process to its logical extreme of $m = B = \log_2(N)$ stages, allows the computation to be done in:

$$\boxed{\frac{1}{2}NB = \frac{1}{2}N\log_2(N)} \quad \text{(FFT computational cost)} \tag{55}$$

It can be seen Fig. 10 that the total number of multiplications needed to perform all the mergings in each stage is $N/2$, and B is the number of stages. Thus, we may interpret Eq. (55) as,

$$\text{(total multiplications)} = \text{(multiplications per stage)} \times \text{(no. stages)} = \frac{N}{2}B$$

For the $N = 8$ example shown in Fig. 10, we have $B = \log_2(8) = 3$ stages and $N/2 = 8/2 = 4$ multiplications per stage. Therefore, the total cost is $BN/2 = 3 \cdot 4 = 12$ multiplications.

Next, we discuss the so-called *decimation-in-time radix-2 FFT algorithm*. There is also a decimation-in-frequency version, which is very similar. The term radix-2 refers to the choice of N as a power of 2, in Eq. (53).

Given a length-N sequence $x(n)$, $n = 0, 1, \ldots, N-1$, its N-point DFT $X(k) = X(\omega_k)$ can be written in the component-form of Eq. (17):

$$X(k) = \sum_{n=0}^{N-1} W_N^{kn} x(n), \qquad k = 0, 1, \ldots, N-1 \tag{56}$$

The summation index n ranges over both even and odd values in the range $0 \le n \le N-1$. By grouping the even-indexed and odd-indexed terms, we may rewrite Eq. (56) as

$$X(k) = \sum_n W_N^{k(2n)} x(2n) + \sum_n W_N^{k(2n+1)} x(2n+1)$$

To determine the proper range of summations over n, we consider the two terms separately. For the even-indexed terms, the index $2n$ must be within the range $0 \leq 2n \leq N-1$. But, because N is even (a power of two), the upper limit $N-1$ will be odd. Therefore, the highest even index will be $N-2$. This gives the range:

$$0 \leq 2n \leq N-2 \quad \Rightarrow \quad 0 \leq n \leq \frac{N}{2} - 1$$

Similarly, for the odd-indexed terms, we must have $0 \leq 2n+1 \leq N-1$. Now the upper limit can be realized, but the lower one cannot; the smallest odd index is unity. Thus, we have:

$$1 \leq 2n+1 \leq N-1 \quad \Rightarrow \quad 0 \leq 2n \leq N-2 \quad \Rightarrow \quad 0 \leq n \leq \frac{N}{2} - 1$$

Therefore, the summation limits are the same for both terms:

$$X(k) = \sum_{n=0}^{N/2-1} W_N^{k(2n)} x(2n) + \sum_{n=0}^{N/2-1} W_N^{k(2n+1)} x(2n+1) \qquad (57)$$

This expression leads us to define the two length-$(N/2)$ subsequences:

$$\boxed{\begin{aligned} g(n) &= x(2n) \\ h(n) &= x(2n+1) \end{aligned}} \qquad n = 0, 1, \ldots, \frac{N}{2} - 1 \tag{58}$$

and their $(N/2)$-point DFTs:

$$\boxed{\begin{aligned} G(k) &= \sum_{n=0}^{N/2-1} W_{N/2}^{kn}\, g(n) \\ H(k) &= \sum_{n=0}^{N/2-1} W_{N/2}^{kn}\, h(n) \end{aligned}} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \tag{59}$$

Then, the two terms of Eq. (57) can be expressed in terms of $G(k)$ and $H(k)$. We note that the twiddle factors W_N and $W_{N/2}$ of orders N and $N/2$ are related as follows:

$$W_{N/2} = e^{-2\pi j/(N/2)} = e^{-4\pi j/N} = W_N^2$$

Therefore, we may write:

$$W_N^{k(2n)} = (W_N^2)^{kn} = W_{N/2}^{kn}, \quad W_N^{k(2n+1)} = W_N^k\, W_N^{2kn} = W_N^k\, W_{N/2}^{kn}$$

Using the definitions (58), Eq. (57) can be written as:

$$X(k) = \sum_{n=0}^{N/2-1} W_{N/2}^{kn}\, g(n) + W_N^k \sum_{n=0}^{N/2-1} W_{N/2}^{kn}\, h(n)$$

and using Eq. (59),

$$\boxed{X(k) = G(k) + W_N^k\, H(k)} \quad k = 0, 1, \ldots, N-1 \tag{60}$$

This is the basic merging result. It states that $X(k)$ can be rebuilt out of the two $(N/2)$-point DFTs $G(k)$ and $H(k)$. There are N additional multiplications, $W_N^k\, H(k)$.

Using the periodicity of $G(k)$ and $H(k)$, the additional multiplications may be reduced by half to $N/2$. To see this, we split the full index range $0 \le k \le N-1$ into two half-ranges parametrized by the two indices k and $k+N/2$:

$$0 \le k \le \frac{N}{2} - 1 \qquad \Rightarrow \qquad \frac{N}{2} \le k + \frac{N}{2} \le N - 1$$

Therefore, we may write the N equations (60) as two groups of $N/2$ equations:

$$\begin{aligned} X(k) &= G(k) + W_N^k H(k) \\ X(k+N/2) &= G(k+N/2) + W_N^{(k+N/2)} H(k+N/2) \end{aligned} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1$$

Using the periodicity property that any DFT is periodic in k with period its length, we have $G(k+N/2) = G(k)$ and $H(k+N/2) = H(k)$. We also have the twiddle factor property:

$$W_N^{N/2} = (e^{-2\pi j/N})^{N/2} = e^{-j\pi} = -1$$

Then, the DFT merging equations become:

$$\boxed{\begin{aligned} X(k) &= G(k) + W_N^k H(k) \\ X(k+N/2) &= G(k) - W_N^k H(k) \end{aligned}} \qquad k = 0, 1, \ldots, \frac{N}{2} - 1 \qquad (61)$$

They are known as the *butterfly* merging equations. The upper group generates the upper half of the N-dimensional DFT vector $\mathbf{X}$, and the lower group generates the lower half. The $N/2$ multiplications $W_N^k H(k)$ may be used both in the upper and the lower equations, thus reducing the total extra merging cost to $N/2$. Vectorially, we may write them in the form:

$$\begin{bmatrix} X_0 \\ X_1 \\ \vdots \\ X_{N/2-1} \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_{N/2-1} \end{bmatrix} + \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N/2-1} \end{bmatrix} \times \begin{bmatrix} W_N^0 \\ W_N^1 \\ \vdots \\ W_N^{N/2-1} \end{bmatrix}$$

$$\begin{bmatrix} X_{N/2} \\ X_{N/2+1} \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} G_0 \\ G_1 \\ \vdots \\ G_{N/2-1} \end{bmatrix} - \begin{bmatrix} H_0 \\ H_1 \\ \vdots \\ H_{N/2-1} \end{bmatrix} \times \begin{bmatrix} W_N^0 \\ W_N^1 \\ \vdots \\ W_N^{N/2-1} \end{bmatrix} \tag{62}$$

where the indicated multiplication is meant to be component-wise. Together, the two equations generate the full DFT vector $\mathbf{X}$. The operations are shown below.

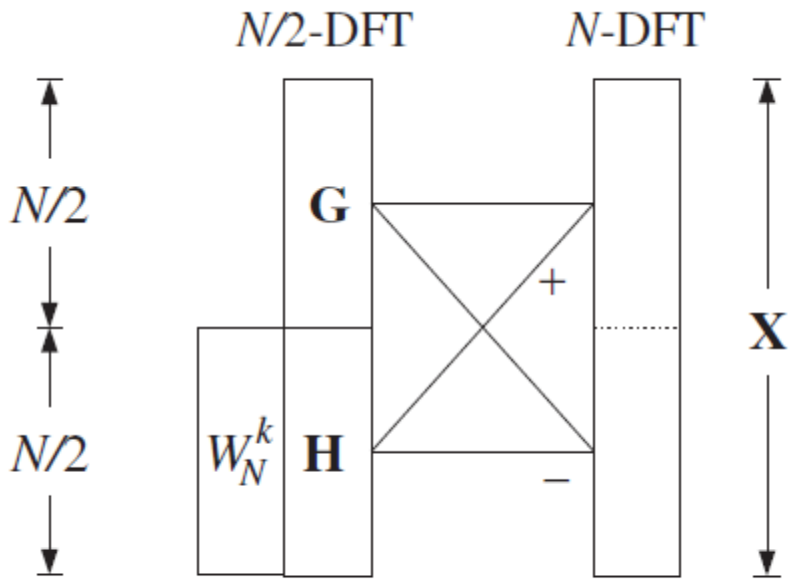


Fig. 11 Butterfly merging builds upper and lower halves of length-N DFT.

As an example, consider the case $N = 2$. The twiddle factor is now $W_2 = -1$, but only its zeroth power appears $W_2^0 = 1$. Thus, we get two 1-dimensional vectors, making up the final 2-dimensional DFT:

$$\begin{aligned}\left[X_0\right] &= \left[G_0\right] + \left[H_0\, W_2^0\right] \\ \left[X_1\right] &= \left[G_0\right] - \left[H_0\, W_2^0\right]\end{aligned}$$

For $N = 4$, we have $W_4 = -j$, and only the powers W_4^0, W_4^1 appear:

$$\begin{aligned}\begin{bmatrix}X_0 \\ X_1\end{bmatrix} &= \begin{bmatrix}G_0 \\ G_1\end{bmatrix} + \begin{bmatrix}H_0\, W_4^0 \\ H_1\, W_4^1\end{bmatrix} \\ \begin{bmatrix}X_2 \\ X_3\end{bmatrix} &= \begin{bmatrix}G_0 \\ G_1\end{bmatrix} - \begin{bmatrix}H_0\, W_4^0 \\ H_1\, W_4^1\end{bmatrix}\end{aligned}$$

And, for $N = 8$, we have:

$$\begin{aligned}\begin{bmatrix}X_0 \\ X_1 \\ X_2 \\ X_3\end{bmatrix} &= \begin{bmatrix}G_0 \\ G_1 \\ G_2 \\ G_3\end{bmatrix} + \begin{bmatrix}H_0\, W_8^0 \\ H_1\, W_8^1 \\ H_2\, W_8^2 \\ H_3\, W_8^3\end{bmatrix} \\ \begin{bmatrix}X_4 \\ X_5 \\ X_6 \\ X_7\end{bmatrix} &= \begin{bmatrix}G_0 \\ G_1 \\ G_2 \\ G_3\end{bmatrix} - \begin{bmatrix}H_0 W_8^0 \\ H_1 W_8^1 \\ H_2 W_8^2 \\ H_3 W_8^3\end{bmatrix}\end{aligned}$$

To begin the merging process shown in Fig. 10, we need to know the starting one-dimensional DFTs. Once these are known, they may be merged into DFTs of dimension 2,4,8, and so on. The starting 1-point DFTs are obtained by the so-called *shuffling* or *bit reversal* of the input time sequence. Thus, the typical FFT algorithm consists of three conceptual parts:

1. Shuffling the N-dimensional input into N length-1 signals.
2. Performing N length-1 DFTs.
3. Merging the N length-1 DFTs into one N-point DFT.

Performing the one-dimensional DFTs is only a conceptual part that lets us pass from the time to the frequency domain. Computationally, it is trivial because the one-point DFT $\mathbf{X} = [X_0]$ of a 1-point signal $\mathbf{x} = [x_0]$ is itself, that is, $X_0 = x_0$, as follows by setting $N = 1$ in Eq. (56).

The shuffling process is shown in Fig. 12 for $N = 8$. It has $B = \log_2(N)$ stages. During the first stage, the given length-N signal block $\mathbf{x}$ is divided into two length-$(N/2)$ blocks $\mathbf{g}$ and $\mathbf{h}$ by putting every other sample into $\mathbf{g}$ and the remaining samples into $\mathbf{h}$.

During the second stage, the same subdivision is applied to $\mathbf{g}$, resulting into the length-$(N/4)$ blocks $\{\mathbf{a}, \mathbf{b}\}$ and to $\mathbf{h}$ resulting into the blocks $\{\mathbf{c}, \mathbf{d}\}$, and so on. Eventually, the signal $\mathbf{x}$ is time-decimated down to N length-1 subsequences.

These subsequences form the starting point of the DFT merging process, which is depicted in Fig. 13 for $N = 8$. The butterfly merging operations are applied to each pair of DFTs to generate the next DFT of doubled dimension.

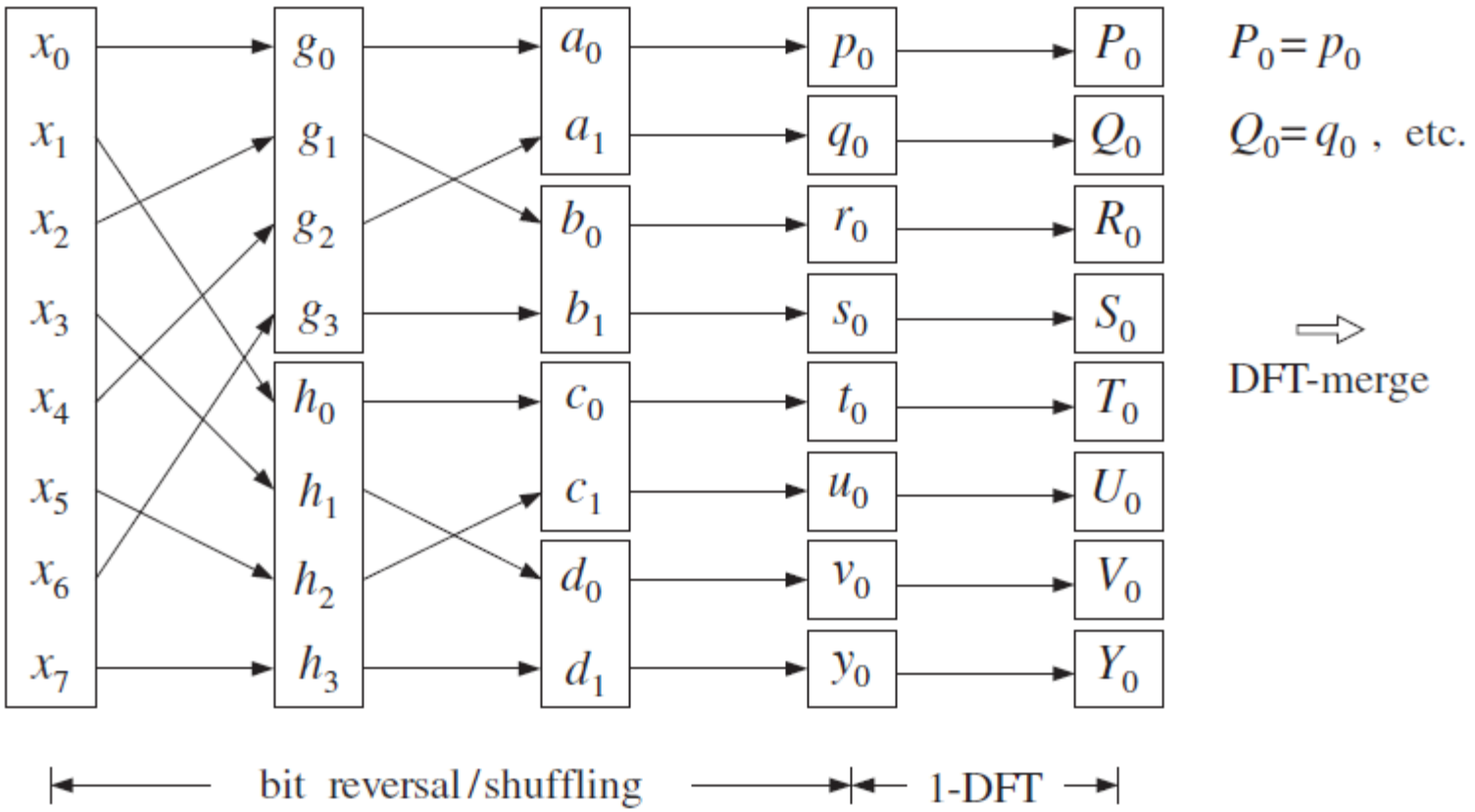


Fig. 12 Shuffling process generates N 1-dimensional signals.

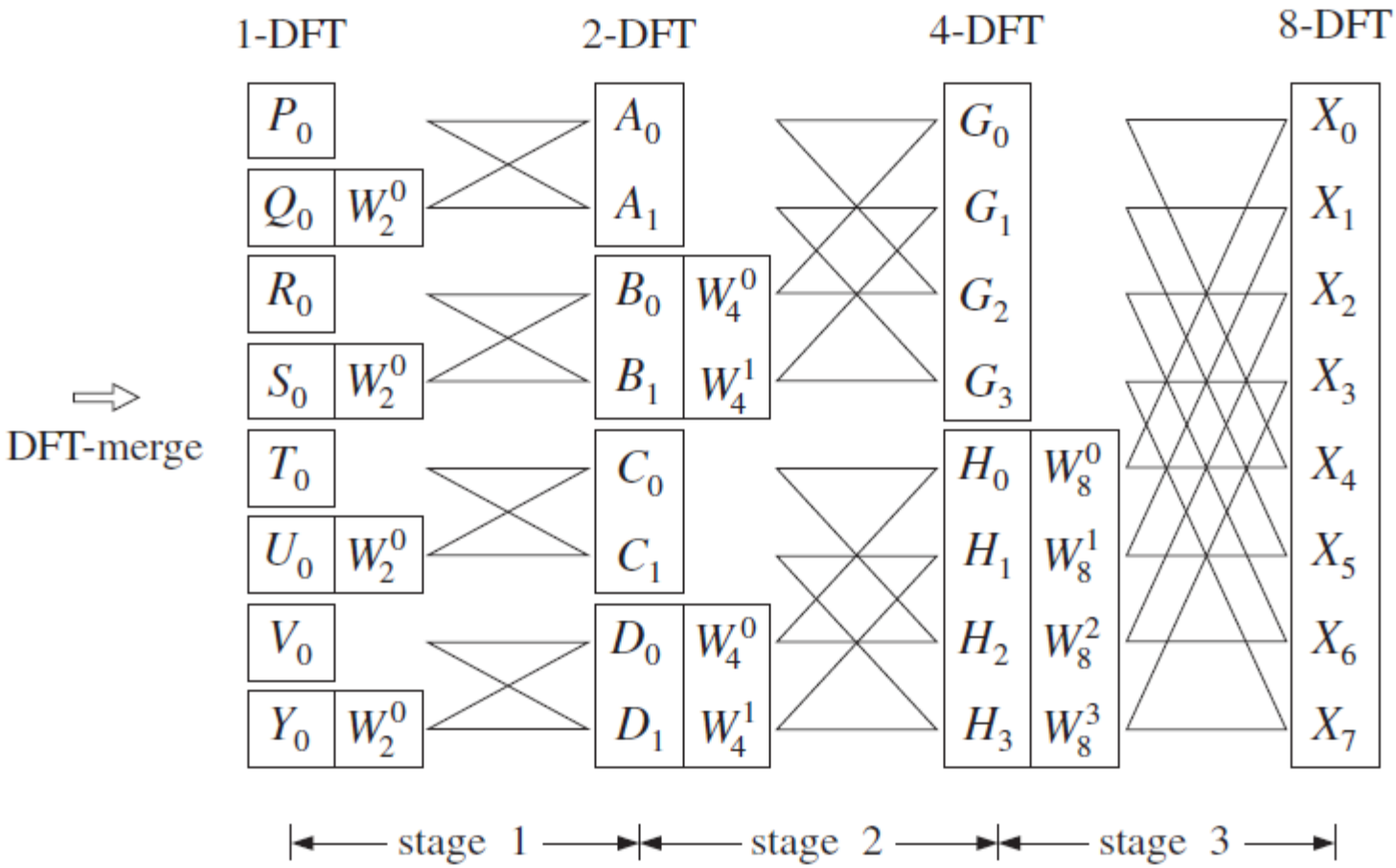


Fig. 13 DFT merging.

To summarize the operations, the shuffling process generates the smaller and smaller signals:

$$\mathbf{x} \rightarrow \{\mathbf{g}, \mathbf{h}\} \rightarrow \big\{\{\mathbf{a}, \mathbf{b}\}, \{\mathbf{c}, \mathbf{d}\}\big\} \rightarrow \cdots \rightarrow \{1\text{-point signals}\}$$

and the merging process rebuilds the corresponding DFTs:

$$\{1\text{-point DFTs}\} \rightarrow \cdots \rightarrow \big\{\{\mathbf{A}, \mathbf{B}\}, \{\mathbf{C}, \mathbf{D}\}\big\} \rightarrow \{\mathbf{G}, \mathbf{H}\} \rightarrow \mathbf{X}$$

The shuffling process may also be understood as a bit-reversal process, shown in Fig. 14. Given a time index n in the range $0 \leq n \leq N-1$, it may be represented in binary by $B = \log_2(N)$ bits. For example, if $N = 8 = 2^3$, we may represent n by three bits $\{b_0, b_1, b_2\}$, which are zero or one:

$$n = (b_2\, b_1\, b_0) \equiv b_2 2^2 + b_1 2^1 + b_0 2^0$$

The binary representations of the time index n for x_n are indicated in Fig. 14, for both the input and the final shuffled output arrays. The bit-reversed version of n is obtained by reversing the order of the bits:

$$r = \text{bitrev}(n) = (b_0\, b_1\, b_2) \equiv b_0 2^2 + b_1 2^1 + b_2 2^0$$

We observe in Fig. 14 that the overall effect of the successive shuffling stages is to put the nth sample of the input array into the rth slot of the output array, that is, swap the locations of x_n with x_r, where r is the bit-reverse of n. Some slots are reverse-invariant so that $r = n$; those samples remain unmoved. All the others get swapped with the samples at the corresponding bit-reversed positions.

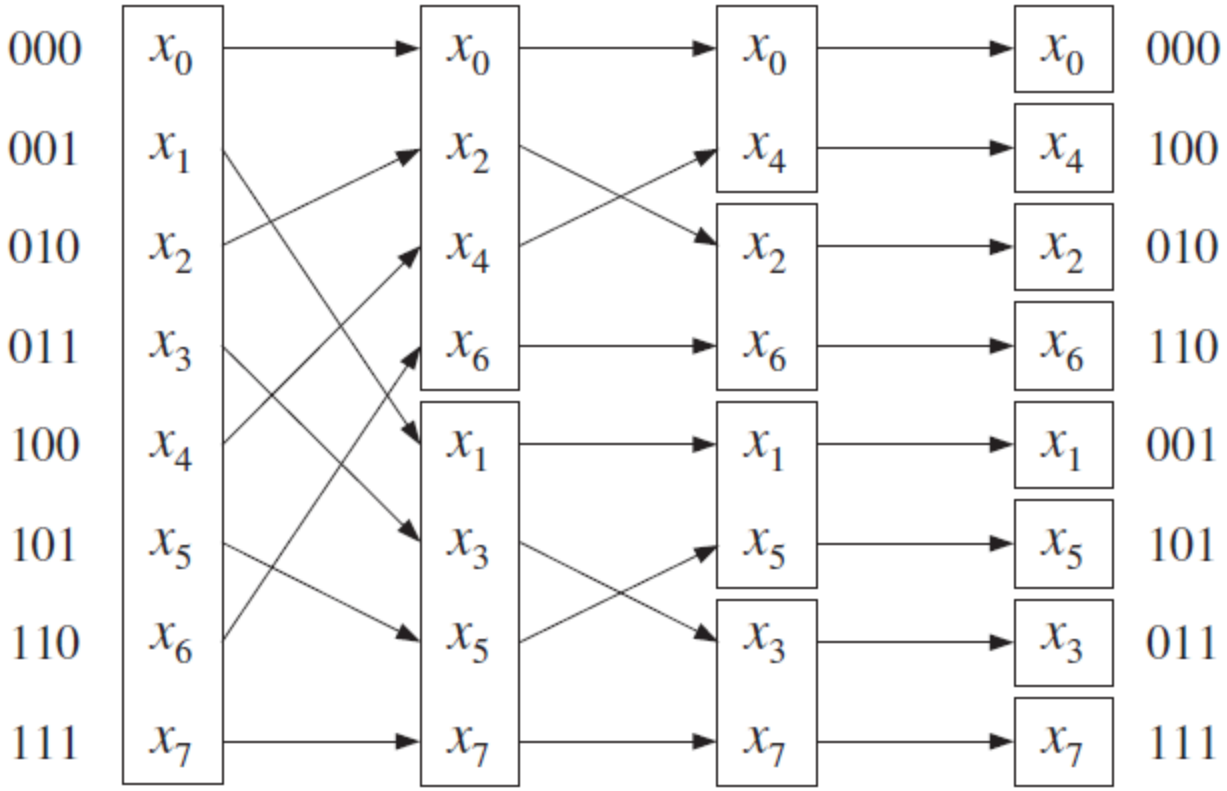


Fig. 14 Shuffling is equivalent to bit reversal.

FFT Computation

The built-in MATLAB function **fft** is very fast and efficient,

```
X = fft(x,N);        % N-point FFT

% if N is omitted, it uses N = L = length(x)
% if N > L, it pads N-L zeros at the end of x before processing
% if N < L, it incorrectly truncates the signal to length N
%           without wrapping it mod-N
%           this can be fixed by using datawrap,
% X = fft(datawrap(x,N),N);
%
% x can be an LxK matrix of K columns of length-L
% the FFTs of all columns are returned into the NxK output X
% again, correct calculation requires N >= L
```

Next, we present some FFT examples. In the merging operations from 2-point to 4-point DFTs and from to 4-DFTs to 8-DFTs, the following twiddle factors are used:

$$\begin{bmatrix} W_4^0 \\ W_4^1 \end{bmatrix} = \begin{bmatrix} 1 \\ -j \end{bmatrix}, \qquad \begin{bmatrix} W_8^0 \\ W_8^1 \\ W_8^2 \\ W_8^3 \end{bmatrix} = \begin{bmatrix} 1 \\ (1-j)/\sqrt{2} \\ -j \\ -(1+j)/\sqrt{2} \end{bmatrix}$$

Example. Using the FFT algorithm, compute the 4-point DFT of the 4-point wrapped signal of a previous example, $\widetilde{\mathbf{x}} = [5,\ 0,\ -3,\ 4]$.

Solution: The sequence of FFT operations are shown in Fig. 15. The shuffling operation was stopped at dimension 2, and the corresponding 2-point DFTs were computed by taking the sum and difference of the time sequences, as in Eq. (22). The DFT merging stage merges the two 2-DFTs into the final 4-DFT.

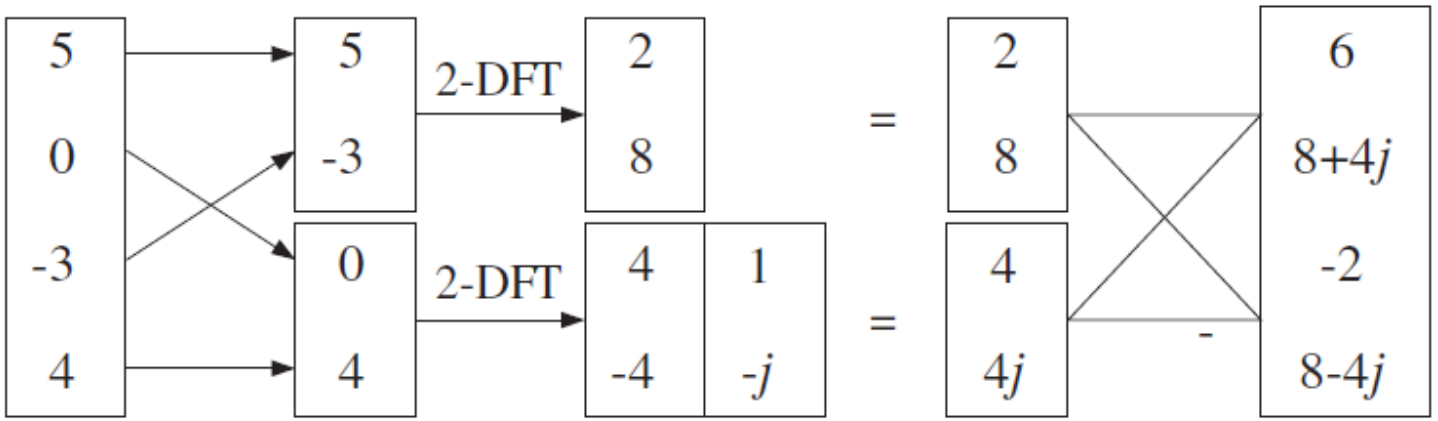


Fig. 15 4-point FFT example.

Example. Using the FFT algorithm, compute the 8-point DFT of the following 8-point signal:

$$\mathbf{x} = [4,\ -3,\ 2,\ 0,\ -1,\ -2,\ 3,\ 1]^T$$

Then, compute the inverse FFT of the result to recover the original time sequence.

Solution: The required FFT operations are shown in Fig. 16. Again, the shuffling stages stop with 2-dimensional signals which are transformed into their 2-point DFTs by forming sums and differences.

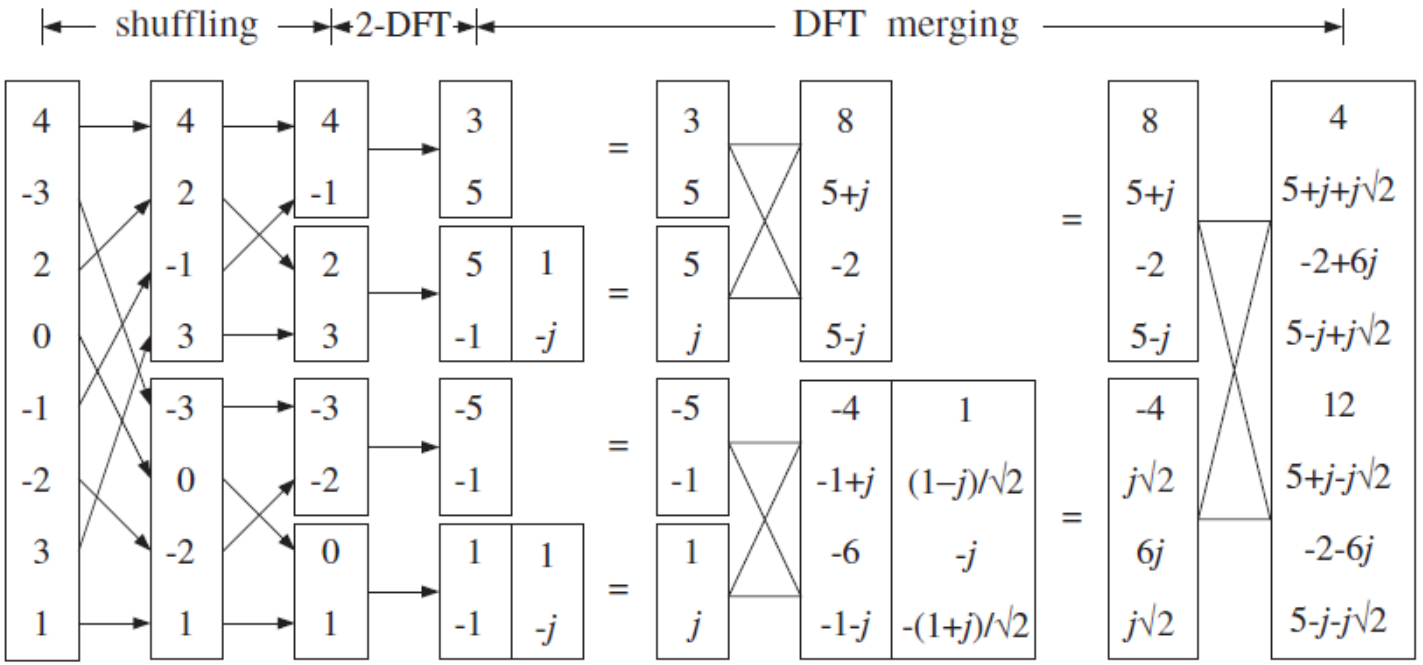


Fig. 16 8-point FFT example.

The inverse FFT is carried out by the expression (47). The calculations are shown in Fig. 17. First, the just computed DFT is complex conjugated. Then, its FFT is computed by carrying out the required shuffling and merging processes. The result must be conjugated (it is real already) and divided by $N = 8$ to recover the original sequence $\mathbf{x}$.

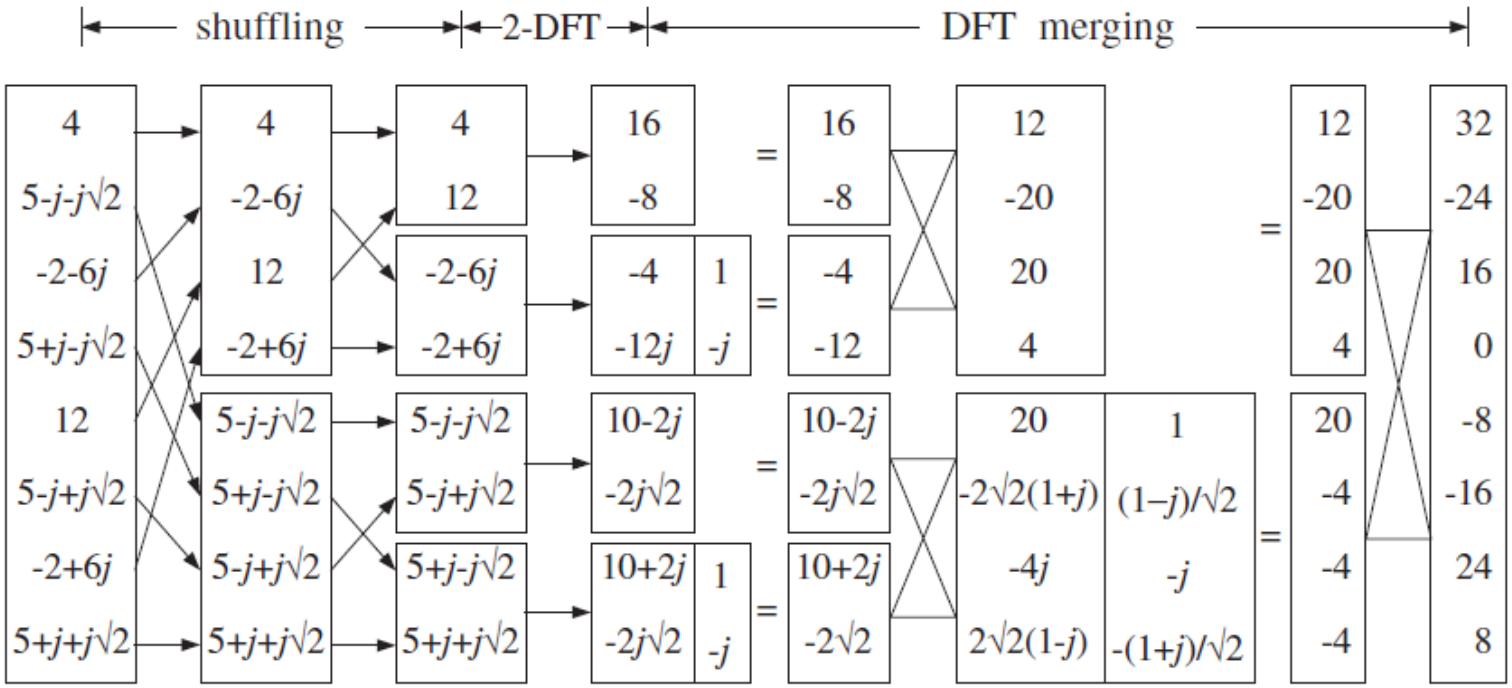


Fig. 17 8-point inverse FFT of the FFT in Fig. 16.

Circular Convolution

In the frequency domain, convolution of two sequences **h** and **x** is equivalent to multiplication of the respective DTFTs:

$$\mathbf{y} = \mathbf{h} * \mathbf{x} \qquad \Leftrightarrow \qquad Y(\omega) = H(\omega)X(\omega) \tag{63}$$

Therefore, $y(n)$ can be recovered by the inverse DTFT of the product of the two DTFTs:

$$y(n) = \int_{-\pi}^{\pi} Y(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} H(\omega)X(\omega)e^{j\omega n}\,\frac{d\omega}{2\pi} \tag{64}$$

Symbolically, we write Eq. (64) as:

$$\mathbf{y} = \text{IDTFT}\big(\text{DTFT}(\mathbf{h}) \cdot \text{DTFT}(\mathbf{x})\big) \tag{65}$$

Equation (64) is not a practical method of computing $y(n)$ even in the case of finite-duration signals, because the ω-integration requires knowledge of $Y(\omega)$ at a continuous range of ω's.

A practical approach would be to replace all the DTFTs by N-point DFTs. But if Eq. (64) is replaced by an inverse DFT, we saw in Eq. (49) that it will reconstruct the wrapped signal $\widetilde{y}(n)$ instead of the desired one:

$$\widetilde{y}(n) = \frac{1}{N}\sum_{k=0}^{N-1} Y(\omega_k)e^{j\omega_k n} = \frac{1}{N}\sum_{k=0}^{N-1} H(\omega_k)X(\omega_k)e^{j\omega_k n} \tag{66}$$

$$\widetilde{\mathbf{y}} = \text{IDFT}\big(\text{DFT}(\mathbf{h}) \cdot \text{DFT}(\mathbf{x})\big) \tag{67}$$

Because the unwrapped $\mathbf{y}$ is the ordinary convolution $\mathbf{y} = \mathbf{h} * \mathbf{x}$, we can write the above as the wrapped convolution:

$$\boxed{\widetilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} = \text{IDFT}\big(\text{DFT}(\mathbf{h}) \cdot \text{DFT}(\mathbf{x})\big)} \quad \text{(circular convolution)} \tag{68}$$

This expression is the definition of the length-N or modulo-N *circular convolution* of the two signals $\mathbf{h}$ and $\mathbf{x}$. A fast version is obtained by replacing DFTs by FFTs resulting in:

$$\boxed{\widetilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} = \mathrm{IFFT}\big(\mathrm{FFT}(\mathbf{h}) \cdot \mathrm{FFT}(\mathbf{x})\big)} \tag{69}$$

If $\mathbf{h}$ and $\mathbf{x}$ are length-N signals, the computational cost of Eq. (69) is the cost for three FFTs (i.e., of $\mathbf{x}$, $\mathbf{h}$, and the inverse FFT) plus the cost of the N complex multiplications $Y(\omega_k) = H(\omega_k)X(\omega_k)$, $k = 0, 1, \ldots, N-1$. Thus, the total number of multiplications to implement Eq. (69) is:

$$3\frac{1}{2}N\log_2(N) + N \tag{70}$$

Some alternative ways of expressing $\widetilde{\mathbf{y}}$ can be obtained by replacing $\mathbf{h}$ and/or $\mathbf{x}$ by their wrapped versions. This would not change the result because the wrapped signals have the same DFTs as the unwrapped ones, that is, $\text{DFT}(\mathbf{h}) = \text{DFT}(\widetilde{\mathbf{h}})$ and $\text{DFT}(\mathbf{x}) = \text{DFT}(\widetilde{\mathbf{x}})$. Thus, we can write:

$$\begin{aligned}
\widetilde{\mathbf{y}} = \widetilde{\mathbf{h} * \mathbf{x}} &= \text{IDFT}\big(\text{DFT}(\mathbf{h}) \cdot \text{DFT}(\mathbf{x})\big) \\
= \widetilde{\widetilde{\mathbf{h}} * \widetilde{\mathbf{x}}} &= \text{IDFT}\big(\text{DFT}(\widetilde{\mathbf{h}}) \cdot \text{DFT}(\widetilde{\mathbf{x}})\big) \\
= \widetilde{\widetilde{\mathbf{h}} * \mathbf{x}} &= \text{IDFT}\big(\text{DFT}(\widetilde{\mathbf{h}}) \cdot \text{DFT}(\mathbf{x})\big) \\
= \widetilde{\mathbf{h} * \widetilde{\mathbf{x}}} &= \text{IDFT}\big(\text{DFT}(\mathbf{h}) \cdot \text{DFT}(\widetilde{\mathbf{x}})\big)
\end{aligned} \tag{71}$$

According to Eq. (51), in order for the circular convolution $\widetilde{\mathbf{y}}$ to agree with the ordinary "linear" convolution $\mathbf{y}$, the DFT length N must be chosen to be at least the length L_y of the sequence $\mathbf{y}$. We recall that if a length-L signal $\mathbf{x}$ is convolved with an order-M filter $\mathbf{h}$, the length of the resulting convolutional output will be $L_y = L + M$. Thus, we obtain the constraint on the choice of N:

$$\boxed{\widetilde{\mathbf{y}} = \mathbf{y} \quad \text{only if} \quad N \geq L_y = L + M} \tag{72}$$

With this choice of N, Eq. (69) represents a fast way of computing linear convolution. Because both the filter and input vectors $\mathbf{h}$, $\mathbf{x}$ have lengths less than N (because $L + M = L_y \leq N$), we must increase them to length N by *padding zeros* at their ends, before we actually compute their N-point FFTs.

If $N < L_y$, part of the tail of $\mathbf{y}$ gets wrapped around to ruin the beginning part of $\mathbf{y}$. The following example illustrates the successive improvement of the circular convolution as the length N increases to the value required by (72).

Example. For the values $N = 3, 5, 7, 9, 11$, compute the mod-N circular convolution of the two signals:

$$\mathbf{h} = [1, 2, -1, 1], \qquad \mathbf{x} = [1, 1, 2, 1, 2, 2, 1, 1]$$

Solution: For this example, we work exclusively in the time domain and perform ordinary convolution and wrap it modulo-N. The convolution table method, or the function **conv**, gives the output signal:

$$\mathbf{y} = \mathbf{x} * \mathbf{h} = [1, 3, 3, 5, 3, 7, 4, 3, 3, 0, 1]$$

The mod-3 circular convolution is obtained by dividing $\mathbf{y}$ into length-3 contiguous blocks, wrapping them around, and summing them to get:

$$\mathbf{y} = [1, 3, 3][5, 3, 7][4, 3, 3][0, 1, 0] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [10, 10, 13]$$

where we padded a 0 at the end to make the last block of length-3. In a similar fashion, we determine the other cases:

(mod-5):

$$\mathbf{y} = [1,3,3,5,3][7,4,3,3,0][1] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [9,7,6,8,3]$$

(mod-7):

$$\mathbf{y} = [1,3,3,5,3,7,4][3,3,0,1] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [4,6,3,6,3,7,4]$$

(mod-9):

$$\mathbf{y} = [1,3,3,5,3,7,4,3,3][0,1] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [1,4,3,5,3,7,4,3,3]$$

(mod-11):

$$\mathbf{y} = [1,3,3,5,3,7,4,3,3,0,1] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [1,3,3,5,3,7,4,3,3,0,1]$$

As N increases to $L_y = L + M = 8 + 3 = 11$, the lengths of the parts that get wrapped around become less and less, making $\widetilde{\mathbf{y}}$ resemble $\mathbf{y}$ more and more. $\square$

Example. Recompute the length-3 circular convolution of the previous example by first wrapping mod-3 the signals $\mathbf{h}$ and $\mathbf{x}$, performing their linear convolution, and wrapping it mod-3.

Solution: We find for the mod-3 reductions:

$$\begin{aligned} \mathbf{h} &= [1,2,-1][1] \quad \Rightarrow \quad \widetilde{\mathbf{h}} = [2,2,-1] \\ \mathbf{x} &= [1,1,2][1,2,2][1,1] \quad \Rightarrow \quad \widetilde{\mathbf{x}} = [3,4,4] \end{aligned}$$

The convolution of the wrapped signals is:

$$\widetilde{\mathbf{h}} * \widetilde{\mathbf{x}} = [2,2,-1] * [3,4,4] = [6,14,13,4,-4]$$

and, its mod-3 reduction:

$$\widetilde{\mathbf{h}} * \widetilde{\mathbf{x}} = [6,14,13][4,-4] \quad \Rightarrow \quad \widetilde{\widetilde{\mathbf{h}} * \widetilde{\mathbf{x}}} = [10,10,13]$$

which agrees with $\widetilde{\mathbf{y}}$, in accordance with Eq. (71). □

Example. Compute the mod-4 circular convolution of the following signals in two ways: (a) working in the time domain, and (b) using DFTs.

$$\mathbf{h} = [1, 2, 2, 1], \qquad \mathbf{x} = [1, 3, 3, 1]$$

Solution: The linear convolution is:

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [1, 2, 2, 1] * [1, 3, 3, 1] = [1, 5, 11, 14, 11, 5, 1]$$

wrapping it mod-4, we get:

$$\mathbf{y} = [1, 5, 11, 14][11, 5, 1] \quad \Rightarrow \quad \widetilde{\mathbf{y}} = [12, 10, 12, 14]$$

Alternatively, we compute the 4-point DFTs of **h** and **x**:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 6 \\ -1-j \\ 0 \\ -1+j \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ -2-2j \\ 0 \\ -2+2j \end{bmatrix}$$

Multiplying them pointwise, we get:

$$\mathbf{Y} = \begin{bmatrix} Y_0 \\ Y_1 \\ Y_2 \\ Y_3 \end{bmatrix} = \begin{bmatrix} H_0 X_0 \\ H_1 X_1 \\ H_2 X_2 \\ H_3 X_3 \end{bmatrix} = \begin{bmatrix} 48 \\ 4j \\ 0 \\ -4j \end{bmatrix}$$

To take the inverse DFT, we conjugate, take the 4-point DFT, divide by 4, and conjugate the answer:

$$\widetilde{\mathbf{y}} = \text{IDFT}(\mathbf{Y}) = \frac{1}{N}\big[\text{DFT}(\mathbf{Y}^*)\big]^*$$

$$\widetilde{\mathbf{y}} = \frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix}\begin{bmatrix} 48 \\ -4j \\ 0 \\ 4j \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 12 \\ 14 \end{bmatrix}$$

The final conjugation is not necessary because $\widetilde{\mathbf{y}}$ is real. □

Deconvolution

Besides the efficient computation of convolution, the FFT can also be used to determine the impulse response of an unknown system, such as the reverberation impulse response of a room. Given a length-N input and a corresponding length-N measured output, we may compute their N-point DFTs and solve for the DFT of the impulse response of the system:

$$Y(\omega_k) = H(\omega_k)X(\omega_k) \qquad \Rightarrow \qquad H(\omega_k) = \frac{Y(\omega_k)}{X(\omega_k)}, \quad k = 0, 1, \ldots, N-$$

Then, taking the inverse DFT, we have:

$$\widetilde{h}(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(\omega_k)e^{j\omega_k n} = \frac{1}{N}\sum_{k=0}^{N-1} \frac{Y(\omega_k)}{X(\omega_k)}\, e^{j\omega_k n} \tag{73}$$

or, symbolically,

$$\widetilde{\mathbf{h}} = \text{IDFT}\left[\frac{\text{DFT}(\mathbf{y})}{\text{DFT}(\mathbf{x})}\right] = \text{IFFT}\left[\frac{\text{FFT}(\mathbf{y})}{\text{FFT}(\mathbf{x})}\right] \tag{74}$$

The result is again the wrapped version $\widetilde{h}(n)$ of the desired impulse response. For this type of application, the true impulse response $h(n)$ is typically infinite, and therefore, its wrapped version will be different from $h(n)$. However, if the wrapping length N is sufficiently large, such that the exponentially decaying tails of $h(n)$ can be ignored, then $\widetilde{h}(n)$ may be an adequate approximation.

Overlap-Add and Overlap-Save Methods

When the length L of the input signal $\mathbf{x}$ is infinite or very long, the length $L_y = L + M$ of the output will be infinite and the condition (72) cannot be satisfied.

A practical approach is to divide the long input into *contiguous* non-overlapping blocks of manageable length, say L samples, then filter each block and piece the output blocks together to obtain the overall output, as shown in Fig. 18, as discussed in I2SP Ch.4. Thus, processing is carried out on a block by block basis.

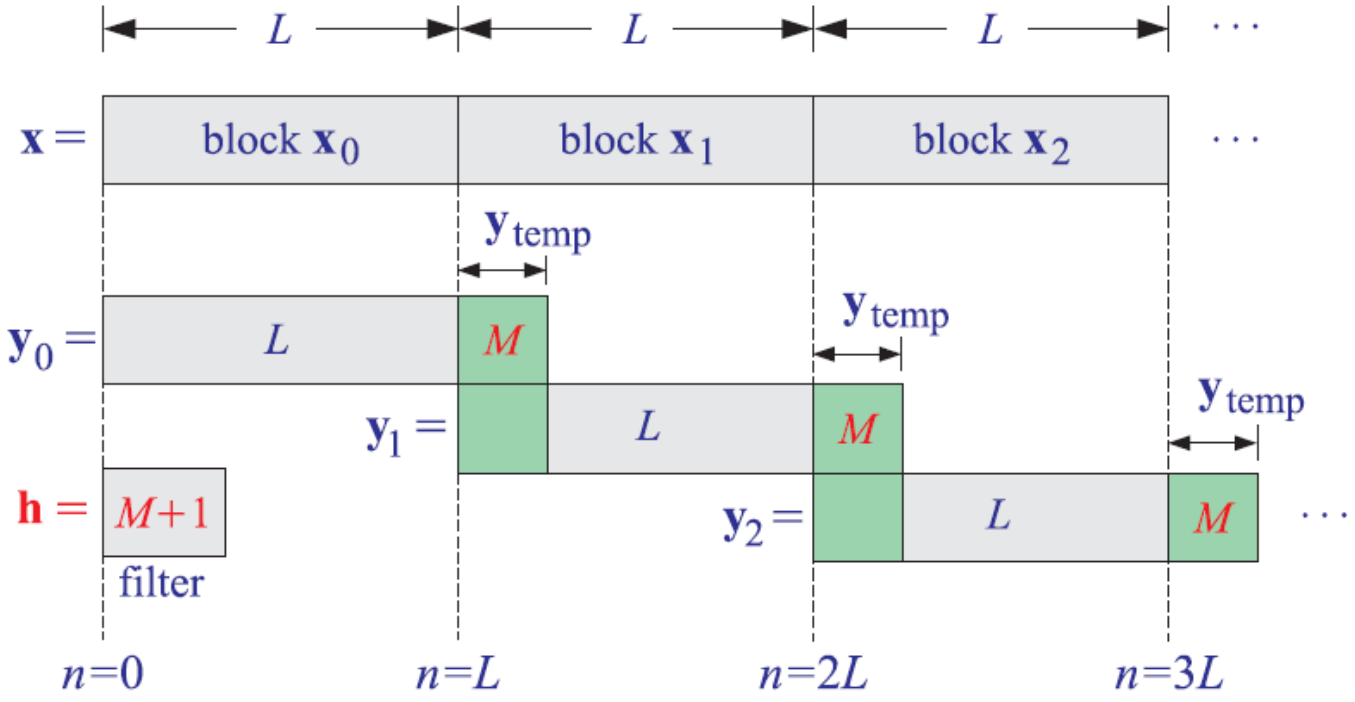


Fig. 18 Overlap-add block convolution method.

This is the *overlap-add* method of block convolution. Each of the input sub-blocks $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, ..., is convolved with the order-M filter $\mathbf{h}$ producing the outputs blocks:

$$\begin{aligned} \mathbf{y}_0 &= \mathbf{h} * \mathbf{x}_0 \\ \mathbf{y}_1 &= \mathbf{h} * \mathbf{x}_1 \\ \mathbf{y}_2 &= \mathbf{h} * \mathbf{x}_2 \end{aligned} \tag{75}$$

and so on. The resulting blocks are pieced together according to their absolute timing. Block $\mathbf{y}_0$ starts at absolute time $n = 0$; block $\mathbf{y}_1$ starts at $n = L$ because the corresponding input block $\mathbf{x}_1$ starts then; block $\mathbf{y}_2$ starts at $n = 2L$, and so forth.

Because each output block is longer than the corresponding input block by M samples, the *last* M samples of each output block will *overlap* with the first M outputs of the *next* block.

Note that only the next sub-block will be involved if we assume that $2L > L + M$, or, $L > M$. To get the correct output points, the overlapped portions must be added together (hence the name, overlap-add).

A fast version of the method can be obtained by performing the convolutions of the input blocks using circular convolution and the FFT by Eq. (69). The FFT length N must satisfy Eq. (72) in order for the output blocks to be correct. Given a desired power of two for the FFT length N, we determine the length of the input segments via:

$$N = L + M \qquad \Rightarrow \qquad \boxed{L = N - M} \tag{76}$$

With this choice of N, there would be no wrap-around errors, and the outputs of the successive input blocks $\{\mathbf{x}_0, \mathbf{x}_1, \ldots\}$, can be computed by:

$$\begin{aligned} \mathbf{y}_0 &= \widetilde{\mathbf{y}}_0 = \text{IFFT}\big(\text{FFT}(\mathbf{h}) \cdot \text{FFT}(\mathbf{x}_0)\big) \\ \mathbf{y}_1 &= \widetilde{\mathbf{y}}_1 = \text{IFFT}\big(\text{FFT}(\mathbf{h}) \cdot \text{FFT}(\mathbf{x}_1)\big) \\ \mathbf{y}_2 &= \widetilde{\mathbf{y}}_2 = \text{IFFT}\big(\text{FFT}(\mathbf{h}) \cdot \text{FFT}(\mathbf{x}_2)\big) \end{aligned} \tag{77}$$

and so on.

In counting the computational cost of this method, the FFT of $\mathbf{h}$ need not be counted. It can be computed once, $\mathbf{H} = \mathrm{FFT}(\mathbf{h})$, and used in all convolutions of Eq. (77). We must only count the cost of *two* FFTs plus the N pointwise multiplications. Thus, the number of multiplications per input block is:

$$2\frac{1}{2}N\log_2 N + N = N(\log_2 N + 1)$$

This must be compared with the cost of $(M+1)L = (M+1)(N-M)$ for performing the ordinary time-domain convolution of each block with the filter. The relative cost of the fast versus the conventional slow method is:

$$\frac{\text{fast}}{\text{slow}} = \frac{N(\log_2 N + 1)}{(M+1)(N-M)} \simeq \frac{\log_2 N}{M} \tag{78}$$

where the last equation follows in the limit $N \gg M \gg 1$.

The *overlap-save* fast convolution method is an alternative method that also involves partitioning the input into blocks and filtering each block by Eq. (69). The method is shown in Fig. 19.

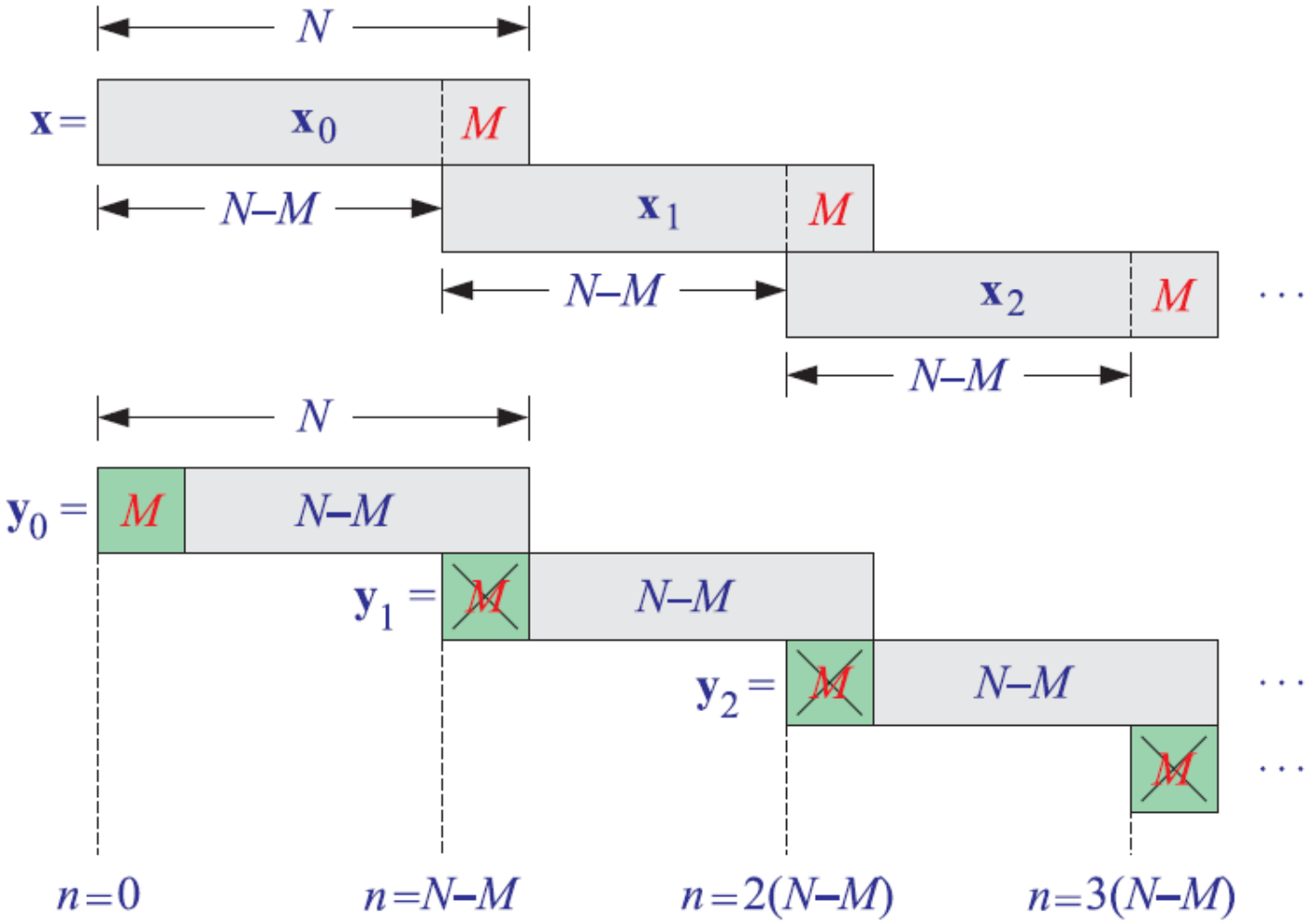


Fig. 19 Overlap-save method of fast convolution.

In this method, the input blocks have length equal to the FFT length, $L = N$, but they are made to overlap each other by M points, where M is the filter order. The output blocks will have length $L_y = L + M = N + M$ and therefore, do not satisfy the condition Eq. (72).

If the output blocks are computed via Eq. (69), then the last M points of each output block will get wrapped around and be added to the first M output points, ruining them. This is shown in Fig. 20. Assuming $N > M$, the remaining output points will be correct.

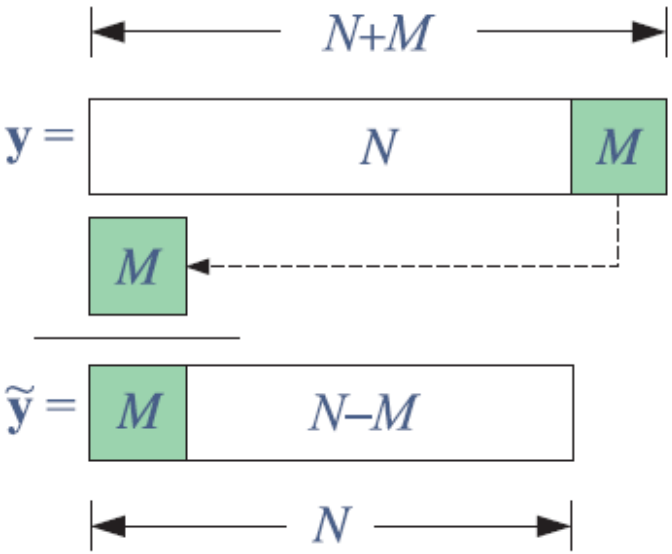


Fig. 20 Mod-N reduction of output block ruins first M output samples.

As shown in Fig. 19, because the input blocks overlap by M points, when the wrapped output blocks are aligned according to their absolute timings, the first M points of each block can be ignored because the correct outputs have already been computed from the previous block.

There is only one exception, that is, the very first M points of the output sequence are not computed correctly. This can be corrected by delaying the input by M time units before commencing the filtering operation.

The computational cost of the method is essentially the same as that of the overlap-add method, with the relative performance over conventional convolution given by Eq. (78).

Example. Using the overlap-save method of fast convolution, implemented in the time domain by mod-8 circular convolutions, compute the linear convolution of the "long" input:

$$\mathbf{x} = [1, 1, 1, 1, 3, 3, 3, 3, 1, 1, 1, 2, 2, 2, 2, 1, 1, 1, 1]$$

with the "short" filter:

$$\mathbf{h} = [1, -1, -1, 1]$$

Solution: For comparison, we compute the linear convolution using the convolution table:

$$\mathbf{y} = [1, 0, -1, 0, 2, 0, -2, 0, -2, 0, 2, 1, 0, -1, 0, -1, 0, 1, 0, -1, 0, 1]$$

For the overlap-save method, we divide the input into length-8 blocks which overlap by $M = 3$ points. These blocks are:

$$\mathbf{x} = [1, 1, 1, 1, 3, \big(3, 3, 3], 1, 1, [1, 2, 2\big), 2, 2, \big(1, 1, 1], 1, 0, 0, 0, 0\big)$$

Convolving these blocks with $\mathbf{h}$ gives:

$$\begin{aligned}
\mathbf{y}_0 &= \mathbf{h} * [1,1,1,1,3,3,3,3] = [1,0,-1,0,2,0,-2,0,-3,0,3] \\
\mathbf{y}_1 &= \mathbf{h} * [3,3,3,1,1,1,2,2] = [3,0,-3,-2,0,2,1,0,-3,0,2] \\
\mathbf{y}_2 &= \mathbf{h} * [1,2,2,2,2,1,1,1] = [1,1,-1,-1,0,-1,0,1,-1,0,1] \\
\mathbf{y}_3 &= \mathbf{h} * [1,1,1,1,0,0,0,0] = [1,0,-1,0,-1,0,1,0,0,0,0]
\end{aligned}$$

Reducing them modulo-8 and ignoring the first M points (indicated by $*$),

$$\begin{aligned}
\widetilde{\mathbf{y}}_0 &= [*,*,*,0,2,0,-2,0] \\
\widetilde{\mathbf{y}}_1 &= [*,*,*,-2,0,2,1,0] \\
\widetilde{\mathbf{y}}_2 &= [*,*,*,-1,0,-1,0,1] \\
\widetilde{\mathbf{y}}_3 &= [*,*,*,0,-1,0,1,0]
\end{aligned}$$

These would be the outputs computed via the FFT method. Putting them together, we obtain the overall output signal:

$$\mathbf{y} = [*,*,*,0,2,0,-2,0][-2,0,2,1,0][-1,0,-1,0,1][0,-1,0,1,0]$$

With the exception of the first 3 points, the answer is correct. □

further notes on fast convolution & project-5

The following MATLAB example clarifies the operations that are required for implementing the overlap-add and overlap-save method both in the time-domain and in the frequency-domain via the FFT. It uses the built-in function **buffer**. This code can serve as the basis of the functions **ovadd** and **ovsave** required in project-5.

The script for this example, **ov1.m**, as well as the overlap-add function, **ola.m**, are included as part of project-5.

The function **ola** will be needed also in future projects.

overlap-add example

```
h = [1 2 -1 1];
x = [1 1 2 1 2 2 1 1 3 3 1 1];
y = conv(h,x);

% y =
%     1   3   3   5   3   7   4   3   6   9   5   3   4   0   1    % expected result

N = 8;                  % fft length
M = length(h)-1;        % filter order
L = N-M;                % block length, L=5

Xb = buffer(x,L);       % divide x into length-L blocks

% Xb =
%       1    2    1
%       1    1    1
%       2    1    0
%       1    3    0
%       2    3    0

Y = [];
for s = Xb
   ys = conv(h,s);
   Y = [Y,ys];                  % collect output columns
end
```

for-loop acts on each column of Xb

overlap-add example

```
% Y' =
%       1    3    3    5    3    5   -1    2
%       2    5    1    6    9    4    0    3
%       1    3    1    0    1    0    0    0

% overlapp-add:
%       1   3   3   5   3   5 -1   2
%                       2   5   1   6   9   4   0   3
%                                       1   3   1   0   1   0   0   0
%       ---------------------------------------------------------------
%       1   3   3   5   3   7   4   3   6   9   5   4   4   0   1   0   0   0
%
% compare with y:
%       1   3   3   5   3   7   4   3   6   9   5   4   4   0   1

% alternatively, y = ola(Y,N-M);  % N-M = hop size here

% FFT method for computing Y

H = fft(h(:),N);

Y = [];
```

function **ola** is included in project-5

overlap-add example

```
% FFT method for computing Y

H = fft(h(:),N);

Y = [];

for s = Xb
    Ys = H.*fft(s,N);
    ys = real(ifft(Ys,N));
    Y = [Y,ys];
end

% Y' =
%       1   3   3   5   3   5  -1   2
%       2   5   1   6   9   4   0   3
%       1   3   1   0   1   0   0   0

% alternatively, without loops,
% Nb = size(Xb,2);                      % no. of frames
% Hb = repmat(H,1,Nb);                  % replicate FFT
% Y = real(ifft(Hb.*fft(Xb,N), N);      % do all FFTs at once
```

for-loop acts on each column of Xb

real( ) is required to remove numerically generated very small imaginary parts

same block outputs as before

overlap-save example

```
% use same h,x, and FFT length N=8

% y =
%     1   3   3   5   3   7   4   3   6   9   5   3   4   0   1    % expected result

Xb = buffer([x(:); zeros(M,1)], N, M);

% Xb =
%       0    2    1
%       0    1    3
%       0    2    3
%       1    2    1
%       1    1    1
%       2    1    0
%       1    3    0
%       2    3    0

Y = [];
for s = Xb
   ys = conv(h,s);
   Y = [Y,ys];
end

% Y' =
%    0    0    0    1    3    3    5    3    5   -1    2
%    2    5    2    7    4    3    6    9    4    0    3
%    1    5    8    5    3    4    0    1    0    0    0
```

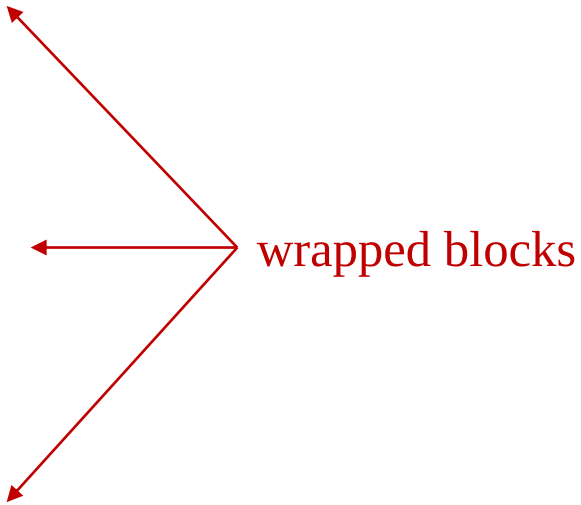
length-*N* blocks overlapped by *M* samples

M zeros are inserted by **buffer** at the beginning, and *M* zeros are padded at the end

time-domain version, runs over the columns of Xb

overlap-save example

```
% wrap each block mod-N:
%
%    0    0    0    1    3    3    5    3
%    5   -1    2
%    ------------------------------
%    5   -1    2    1    3    3    5    3
%
%    2    5    2    7    4    3    6    9
%    4    0    3
%    ------------------------------
%    6    5    5    7    4    3    6    9
%
%    1    5    8    5    3    4    0    1
%    0    0    0
%    ------------------------------
%    1    5    8    5    3    4    0    1
```



overlap-save example

output the wrapped blocks by their start-times,
ignoring the first *M* samples of each block

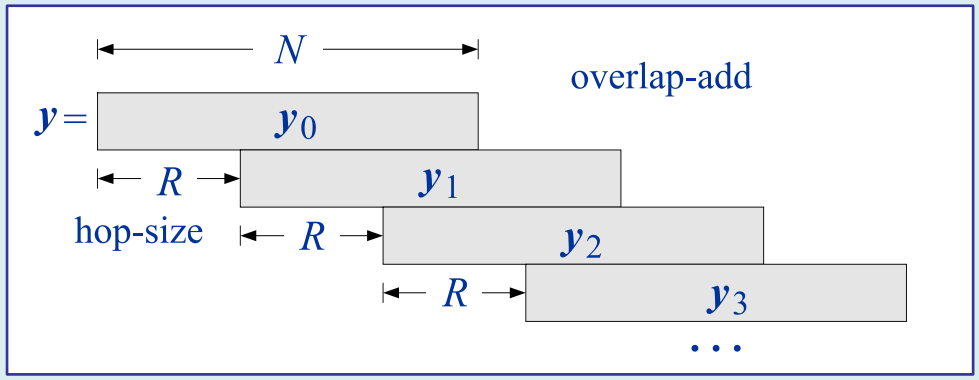
```
%     (5 -1   2) 1   3   3   5   3
%                      (6   5   5) 7   4   3   6   9
%                                          (1   5 8) 5   3   4   0   1
% compare with y:
% y =
%                1   3   3   5   3   7   4   3   6   9   5   3   4   0   1
```

overlap-save example

```
% FFT method

H = fft(h(:),N);

Y = [];
for s = Xb
    ys = real(ifft(H.*fft(s,N), N));
    Y = [Y, ys];
end

% or, w/o loops,
%   Y = real(ifft(repmat(H,1,size(Xb,2)).*fft(Xb,N),N));

% round(Y')
%    5   -1    2    1    3    3    5    3
%    6    5    5    7    4    3    6    9
%    1    5    8    5    3    4    0    1

Y(1:M,:) = [];      % discard first M outputs from each frame
y = Y(:)';          % concatenate frames
% y =
%              1   3   3   5   3   7   4   3   6   9   5   3   4   0   1
```

ola function

```
% ola.m - overlap-add procedure
%
% Usage: y = ola(Y,R)
%
% Y = NxM matrix of columns to be overlap-added by hop-size R
% R = hop-size, must be 0 < R <= N, R=N (no-overlap)
%
% y = column vector of overlap-added columns

function y = ola(Y,R)

[N,M] = size(Y);

L = R*(M-1)+N;
y = zeros(L,1);

n = (1:N)';

for m = 0:M-1
    y(m*R + n) = y(m*R + n) + Y(:,m+1);
end
```



Matched Filter

assume pulse and filter, $s(n), h(n)$, have duration, $0 \leq n \leq n_0$

$$x(n) = s(n) + v(n) = \text{signal + noise}$$

$$y_s(n) = \sum_m h(m)s(n-m) = \text{filtered signal}$$

$$y_v(n) = \sum_m h(m)v(n-m) = \text{filtered noise}$$

and, because of the finite durations,

$$y_s(n_0) = \sum_{m=0}^{n_0} h(m)s(n_0 - m)$$

$$\text{SNR} = \frac{|y_s(n_0)|^2}{E[y_v^2(n)]} = \frac{\left|\sum_{m=0}^{n_0} h(m)s(n_0-m)\right|^2}{\sigma_v^2 \sum_{m=0}^{n_0} h^2(m)} = \max$$

Cauchy-Scwarz inequality:

$$\frac{\left|\sum_{m=0}^{n_0} h(m)s(n_0-m)\right|^2}{\sum_{m=0}^{n_0} h^2(m)} \le \frac{\sum_{m=0}^{n_0} h^2(m) \cdot \sum_{m=0}^{n_0} s^2(n_0-m)}{\sum_{m=0}^{n_0} h^2(m)} \le \sum_{m=0}^{n_0} s^2(n_0-m)$$

optimum solution:

$$\boxed{h(m) = s(n_0-m) = \text{matched filter}} \qquad m = 0, 1, \ldots, n_0$$

$$y_s(n) = \sum_m h(m)s(n-m) = \sum_m s(n_0-m)s(n-m) = R_{ss}(n-n_0)$$

$$\boxed{\begin{array}{c} y_s(n) = R_{ss}(n-n_0) \\ |y(n_0)|_{\max} = R_{ss}(0) \end{array}} \quad \text{(autocorrelation)}$$

for a delayed received signal, $r(n) = s(n-D)$, we have the cross-correlation:

$$y_r(n) = \sum_m h(m)r(n-m) = \sum_m r(n-m)s(n_0-m) = R_{rs}(n-n_0)$$

$$\boxed{\begin{gathered} y_r(n) = R_{rs}(n-n_0) = y_s(n-D) = R_{ss}(n-n_0-D) \\ y_r(n_0+D) = R_{rs}(D) = R_{ss}(0) = y_s(n_0)\big|_{\text{max}} \end{gathered}}$$

i.e., matched filter output peaks n_0 samples later than the cross-correlator.