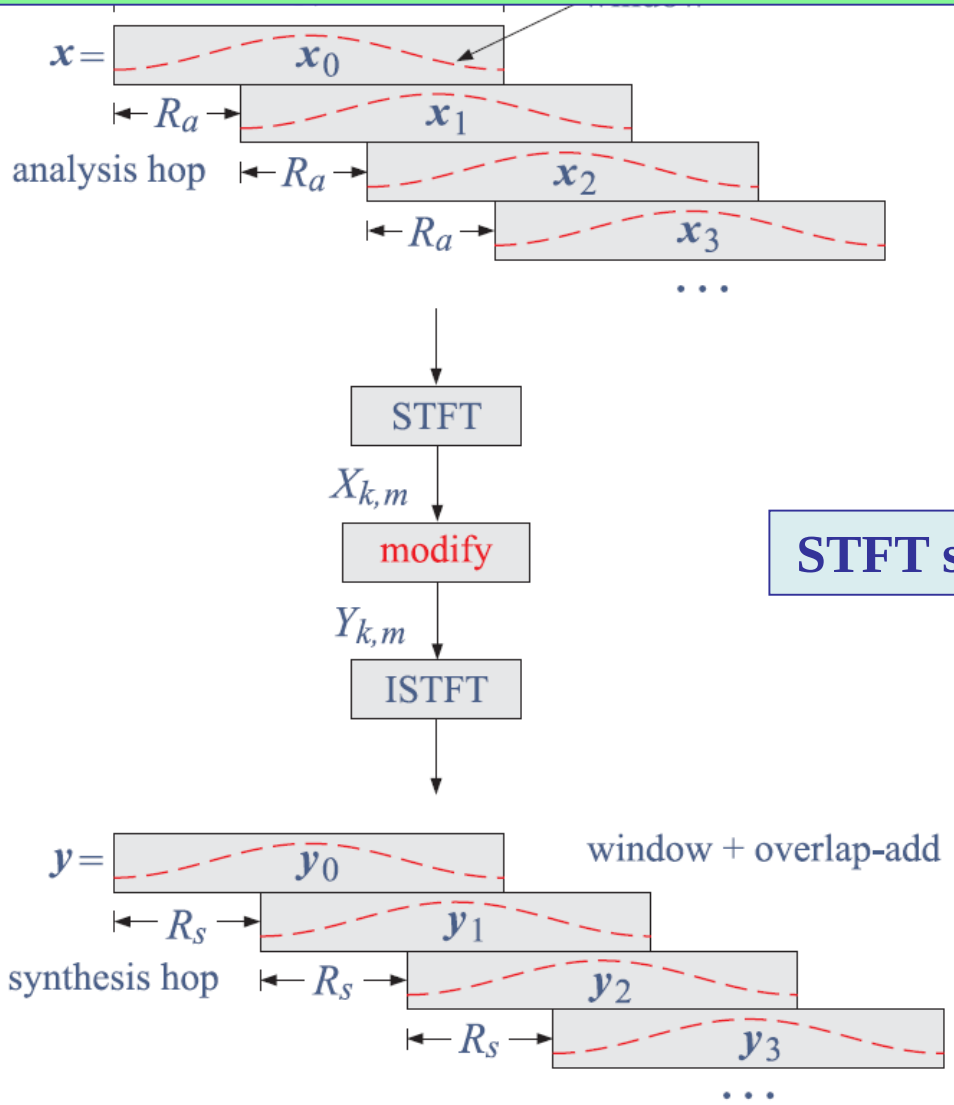
DSA – March 8, 2021

Topics: FFT algorithm, STFT, ISTFT, OLA reconstruction, spectrograms, phase vocoder, time-scale & pitch-scale modification, resampling, COLA window property.



STFT signal processing system

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

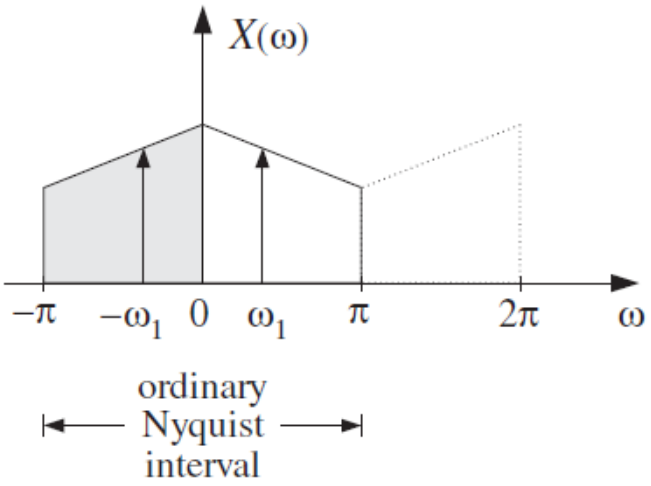
(c) the *coding* of waveforms, such as speech or pictures, for efficient transmission and storage

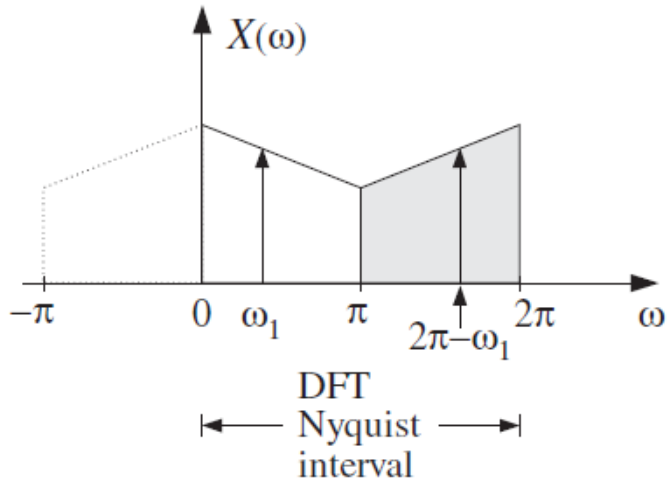
DTFT Computation

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*.





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,

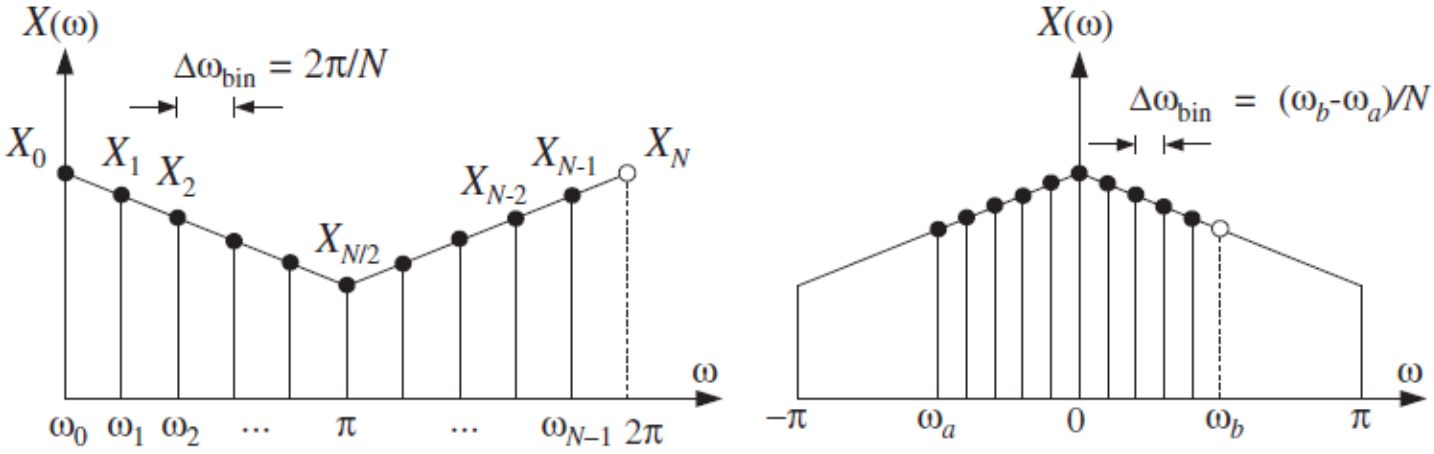


Fig. 1 N-point DTFTs over $[0, 2\pi)$ and over subinterval $[\omega_a, \omega_b)$, for $N = 10$.

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.

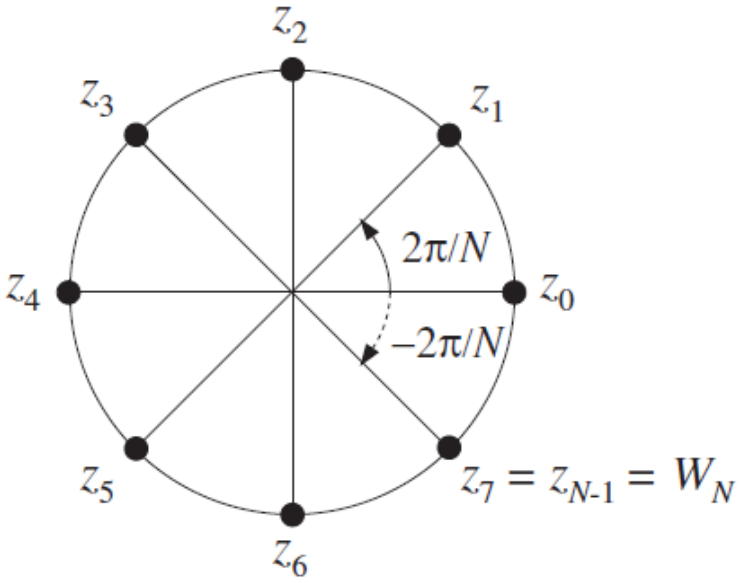


Fig. 2 Nth roots of unity, for $N = 8$.

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);
```

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 $W_N^{(nk) \bmod (N)}$.

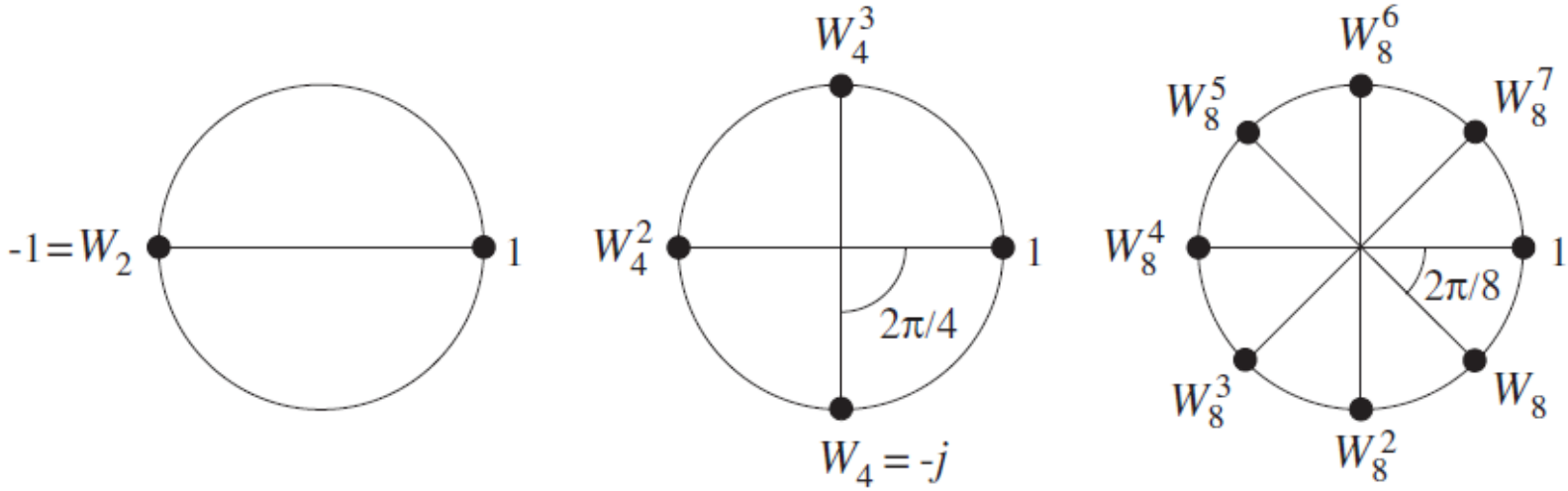


Fig. 5 Twiddle factor lookup tables for $N = 2, 4, 8$.

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 $N = 8$,

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

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

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.

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

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:

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

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

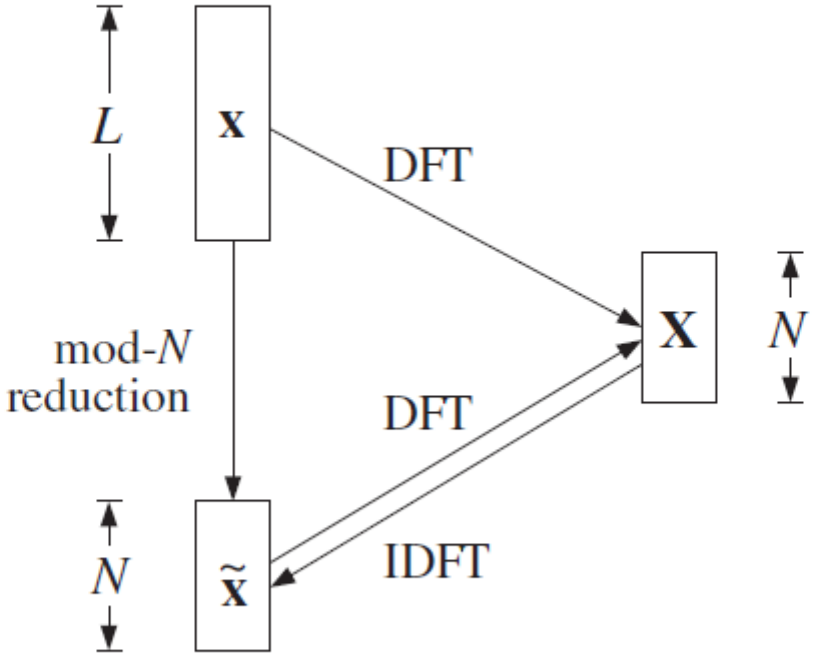


Fig. 9 Forward and inverse N-point DFTs.

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.

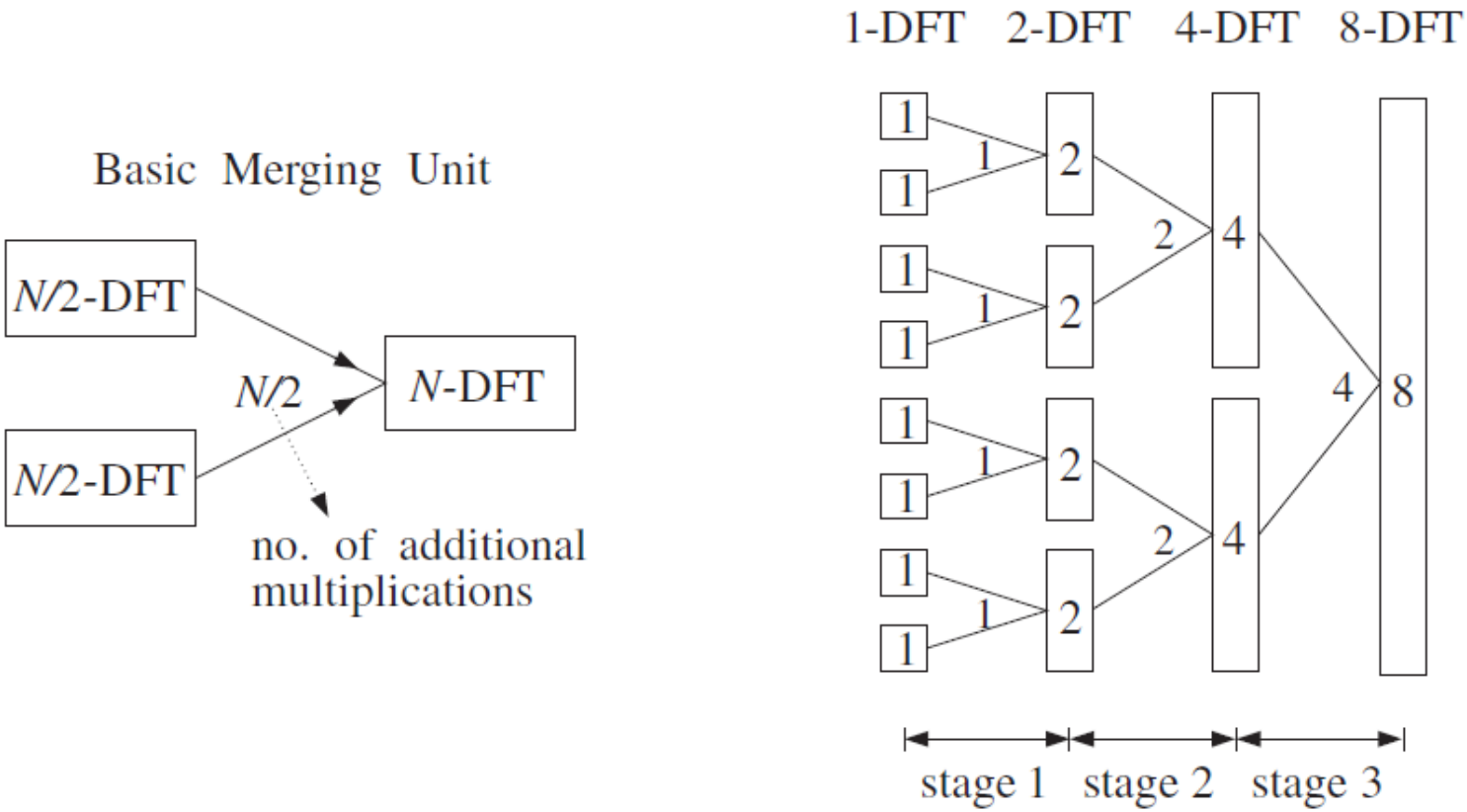


Fig. 10 Merging two $N/2$-DFTs into an N-DFT and its repeated application.

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:

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

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

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.

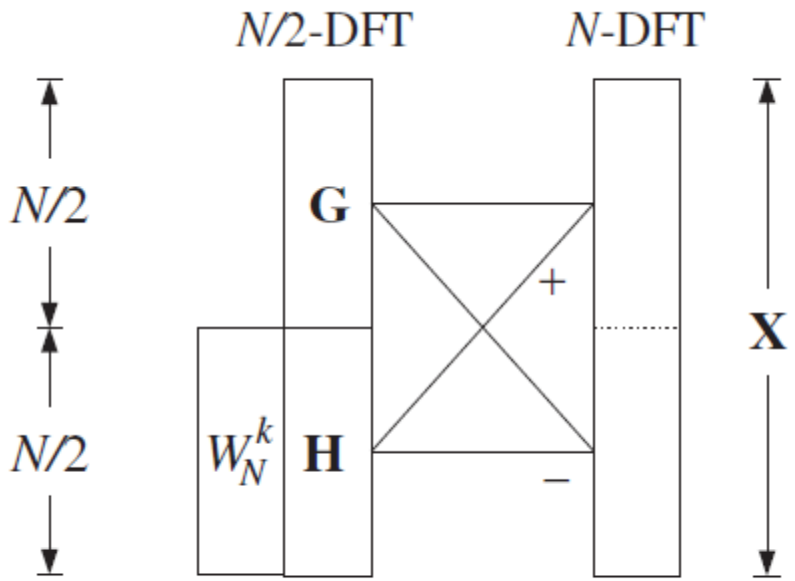


Fig. 11 Butterfly merging builds upper and lower halves of length-N DFT.

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.

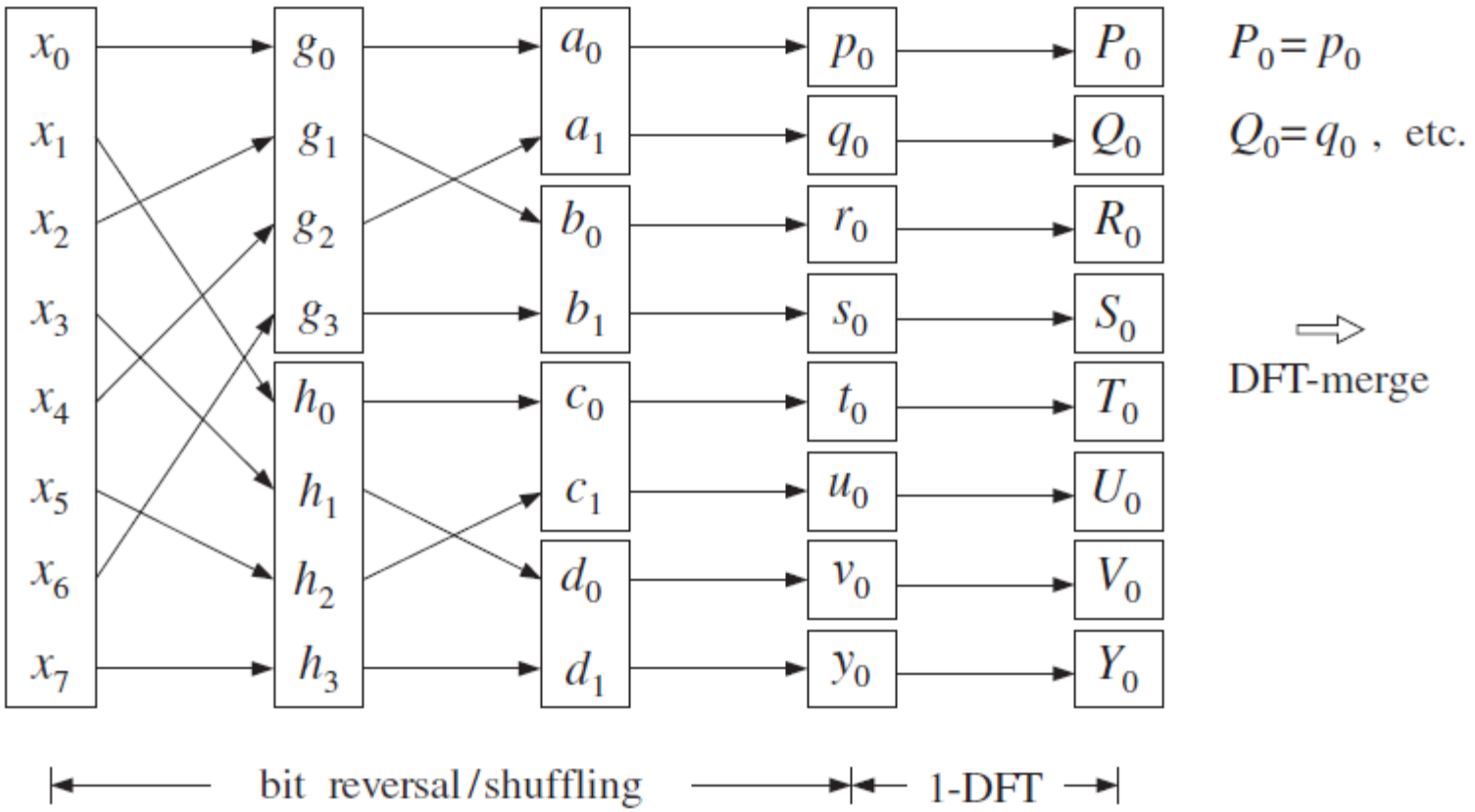


Fig. 12 Shuffling process generates N 1-dimensional signals.

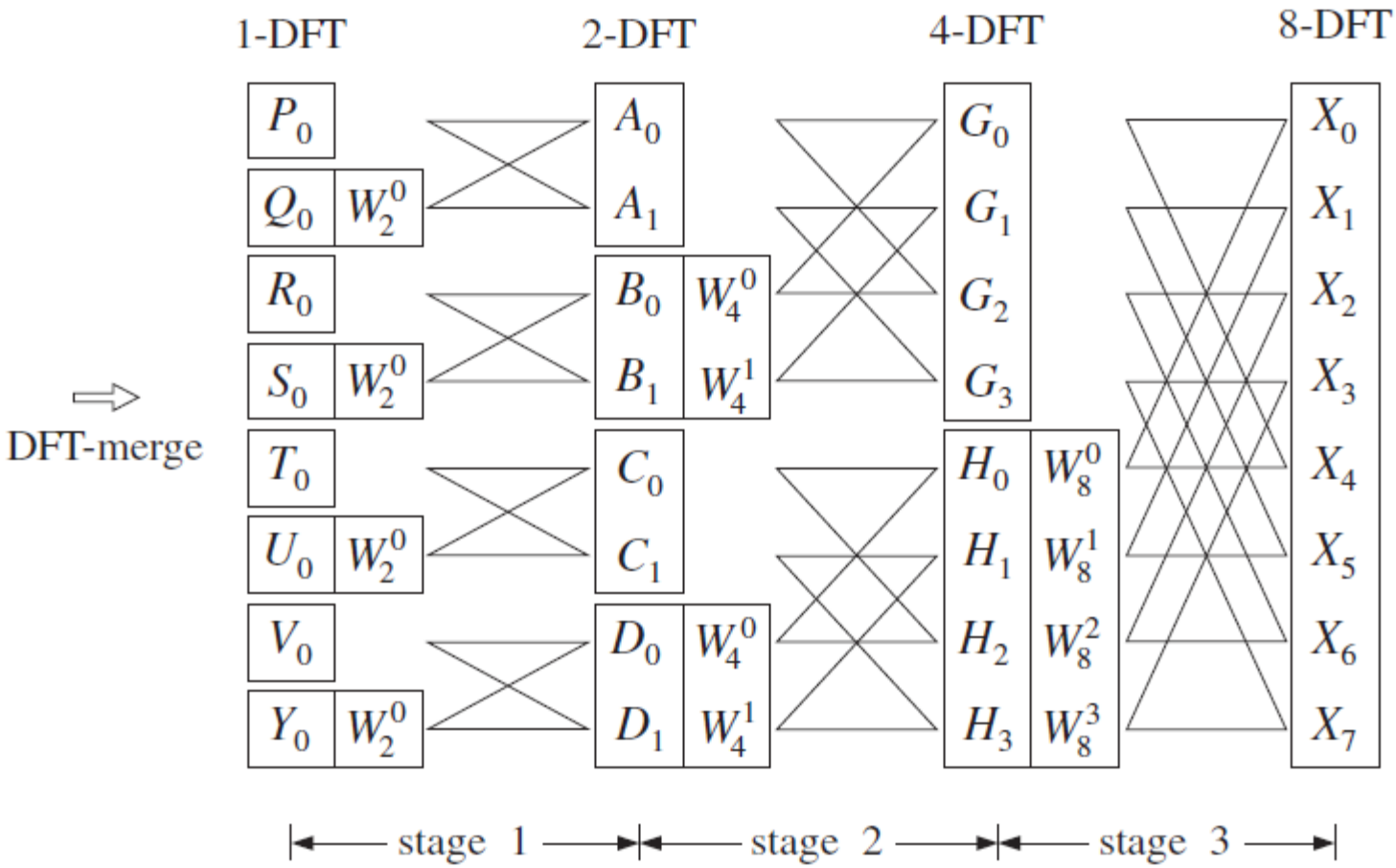


Fig. 13 DFT merging.

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.

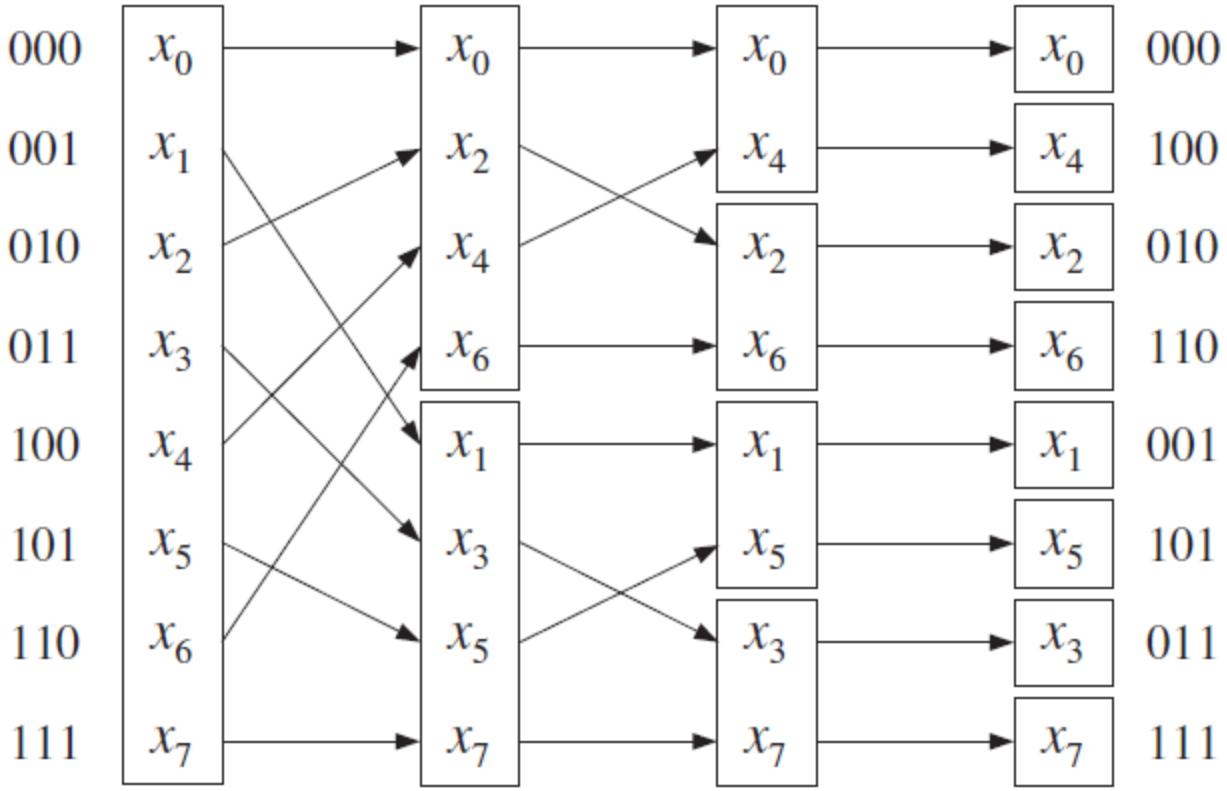


Fig. 14 Shuffling is equivalent to bit reversal.

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.

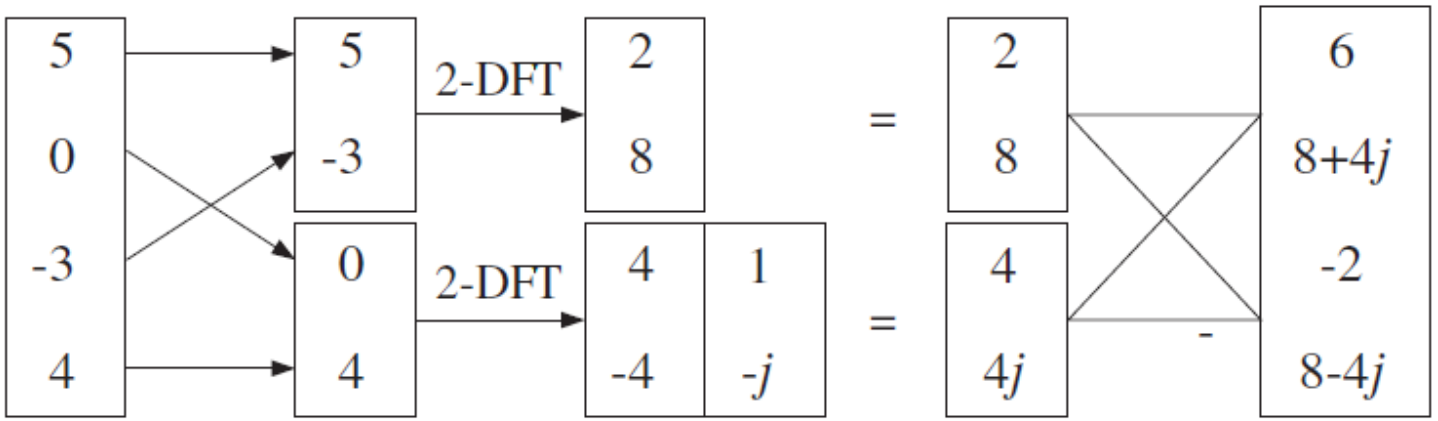


Fig. 15 4-point FFT example.

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.

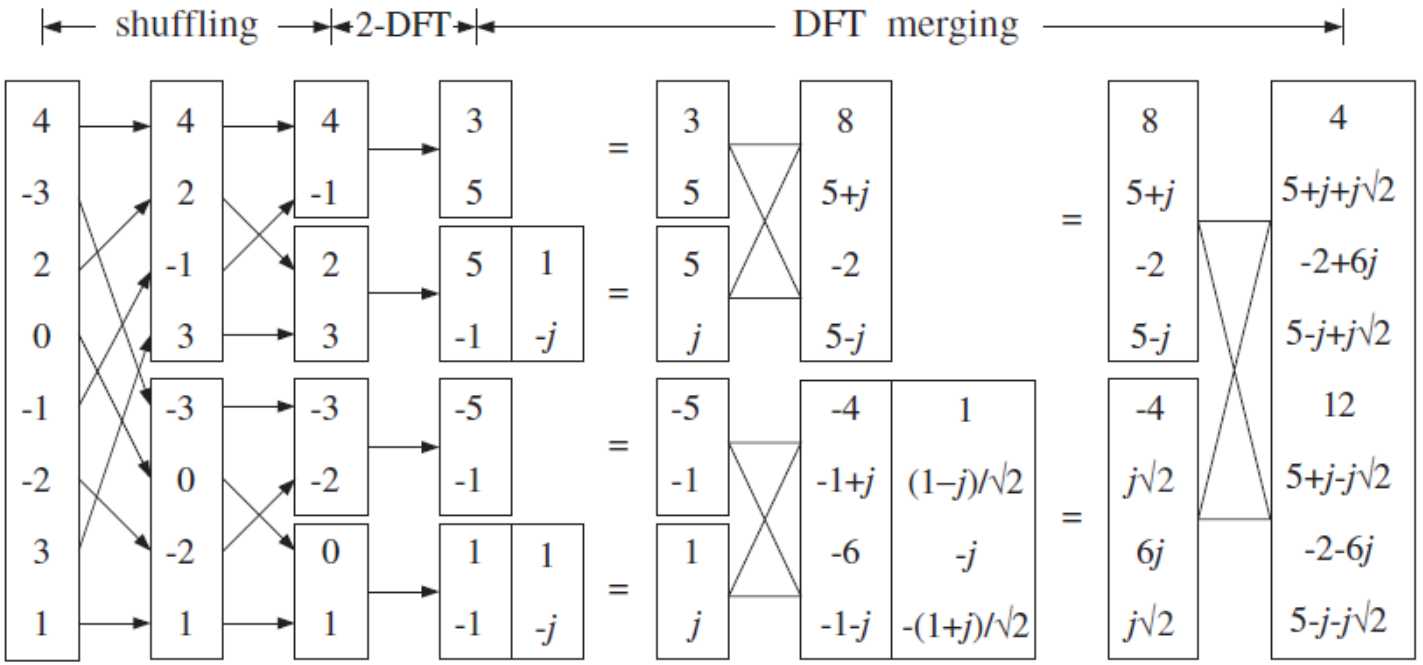


Fig. 16 8-point FFT example.

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 $\mathbf{x}$.

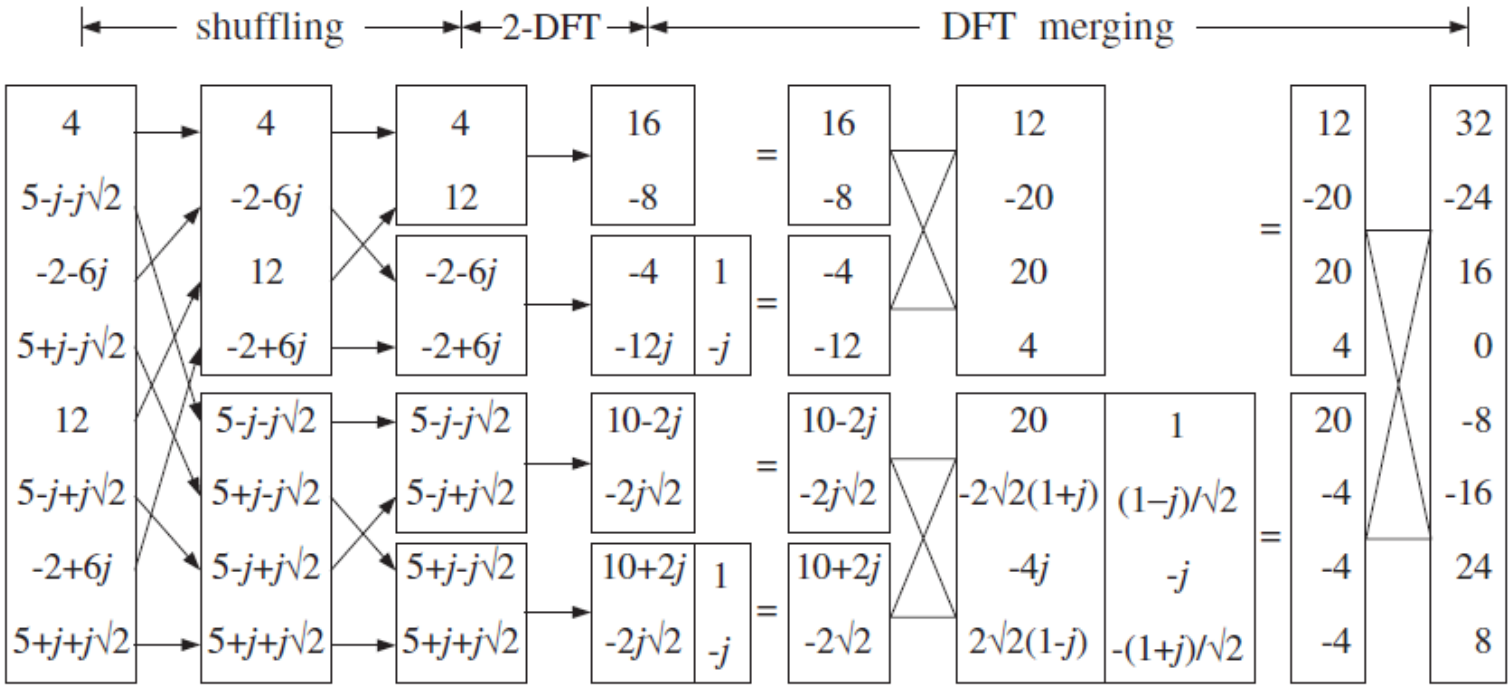


Fig. 17 8-point inverse FFT of the FFT in Fig. 16.

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:

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

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.

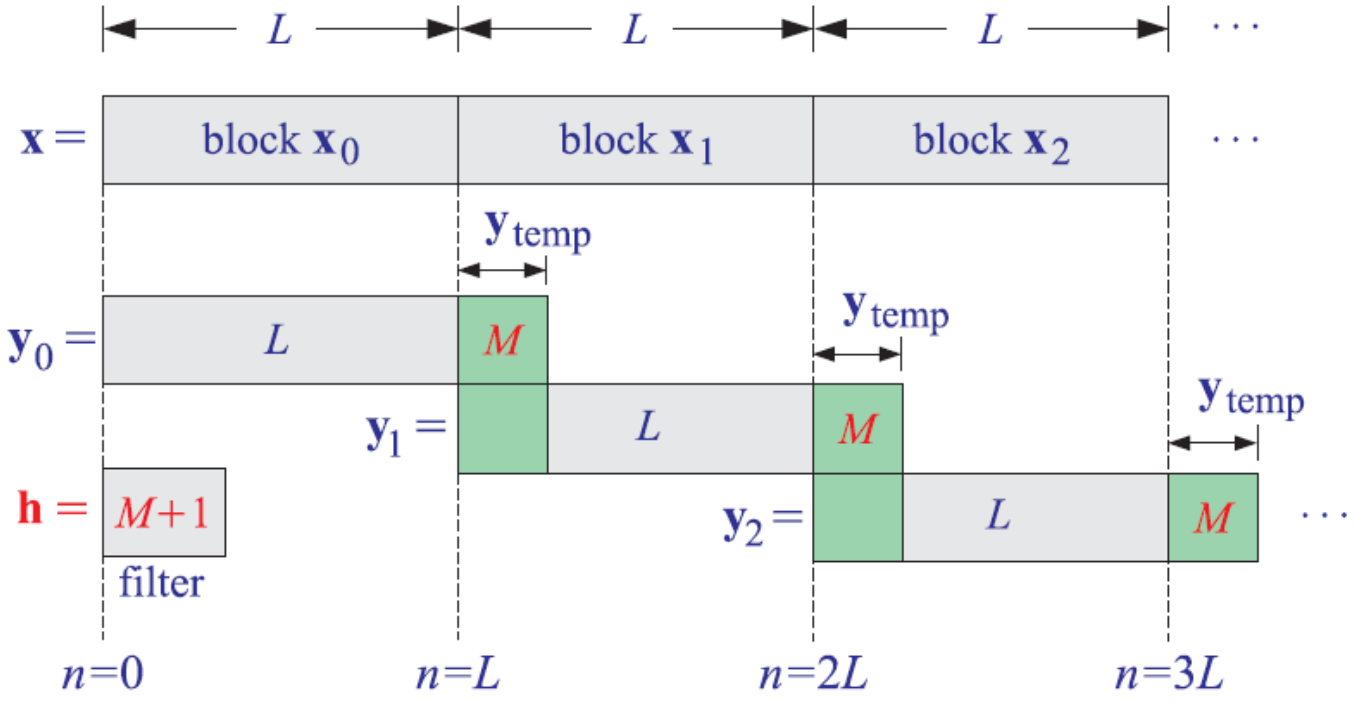


Fig. 18 Overlap-add block convolution method.

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.

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.

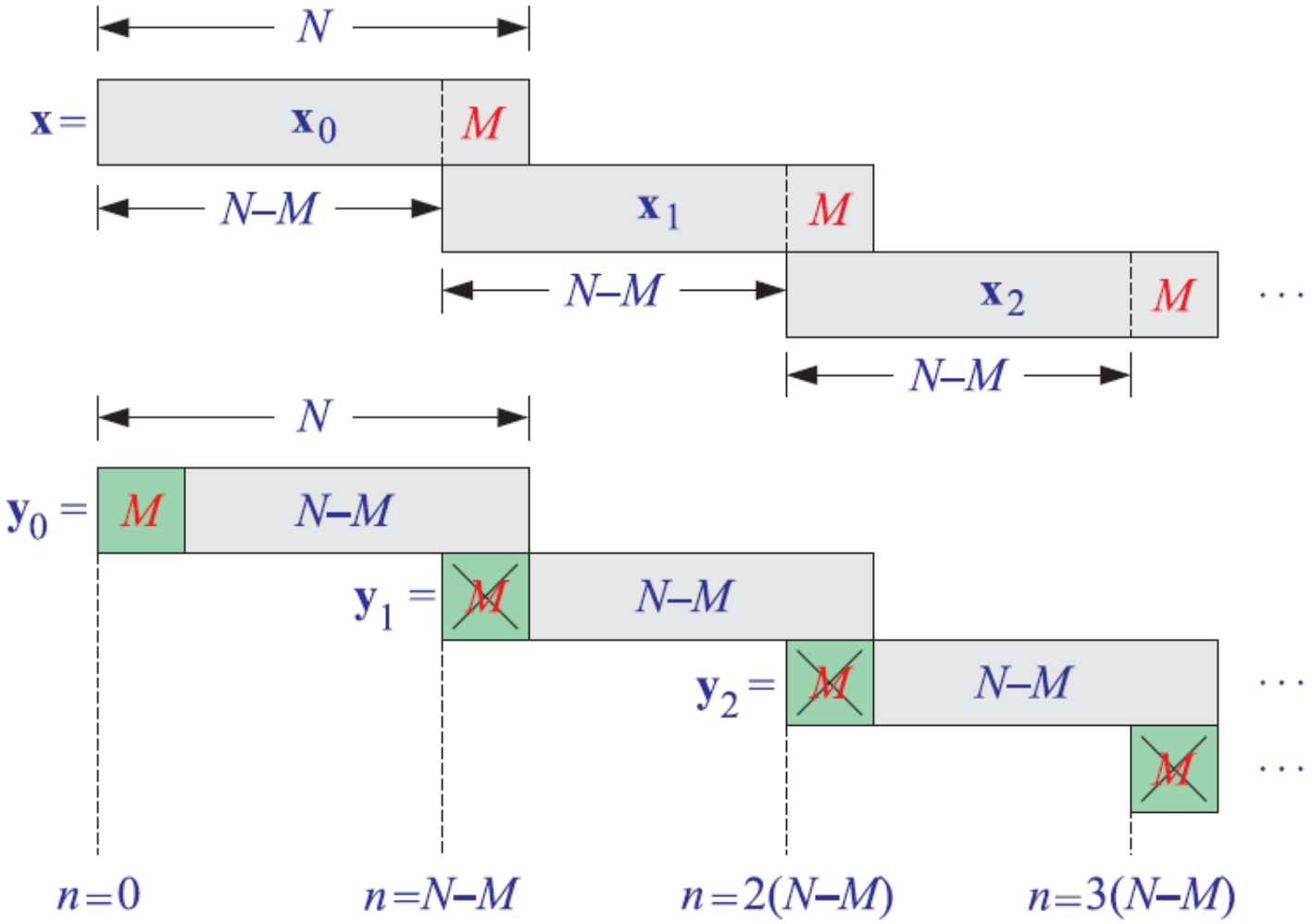


Fig. 19 Overlap-save method of fast convolution.

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.

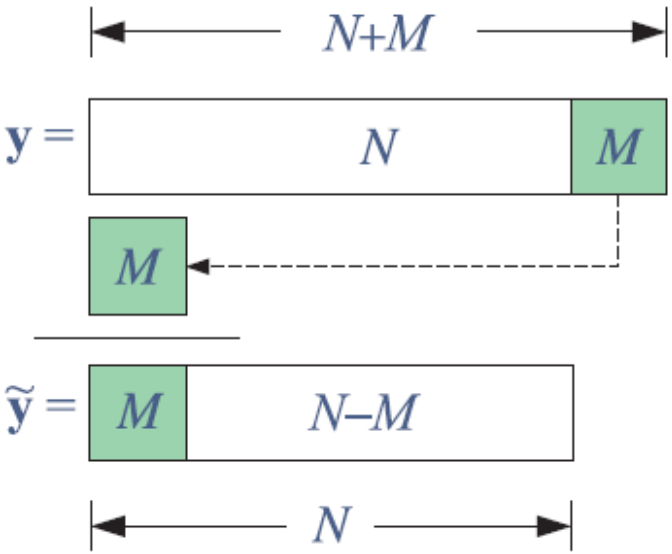


Fig. 20 Mod-N reduction of output block ruins first M output samples.

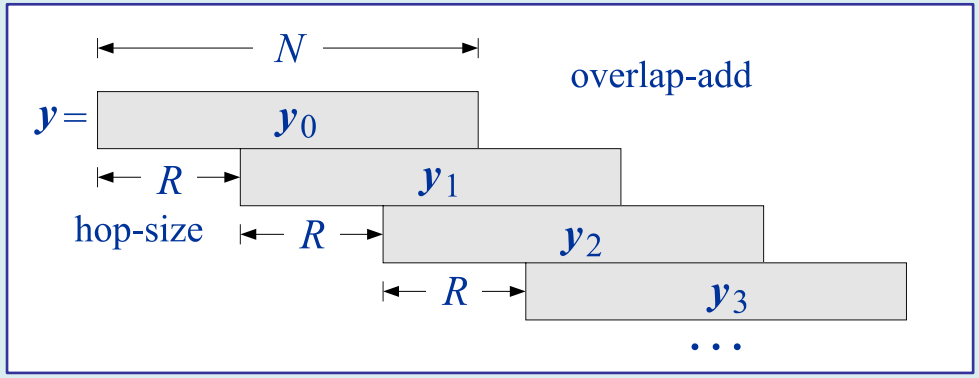
ola function

Last week we discussed the use of the **buffer** function in conjunction with the overlap-add and overlap-save methods. The **ola** function acts on the output of the buffer function to implement the overlap-add operation.

The function **ola** will be needed also in implementing short-time Fourier transform **(STFT)** operations, as in project-6.

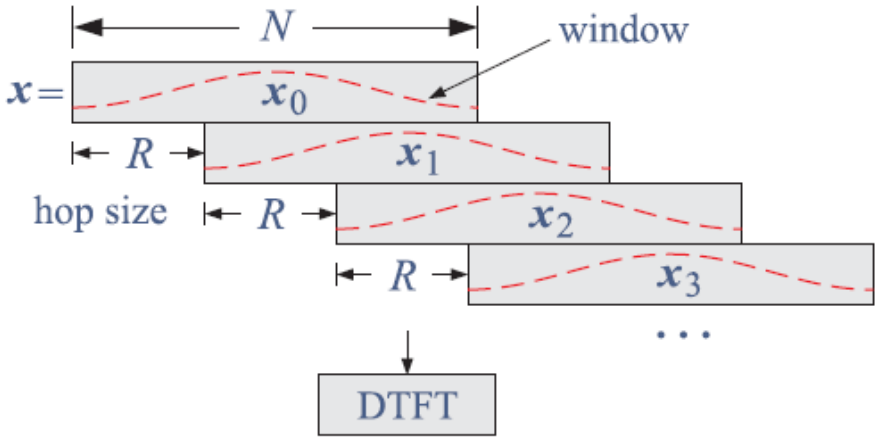
ola function

```
% 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
```

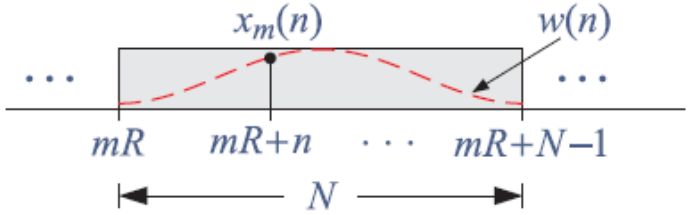


STFT

The short-time Fourier transform (STFT) is defined by dividing the input signal $x(n)$ into successive overlapping length-N blocks, shifted relative to each other by R samples (the hop size), then windowing each block by an appropriate length-N window, $w(n)$, and taking the DTFT of each block,



$$X(\omega, mR) = \sum_{n=0}^{N-1} x(mR+n)w(n)e^{-j\omega n}$$



where $R \leq N$ and the N time samples within the mth segment being transformed are,

$$x_m(n) = x(mR+n)w(n)\,, \quad n = 0, 1, \ldots, N-1$$

$$x_m(n) = x(mR+n)w(n)\,, \quad n = 0, 1, \ldots, N-1$$

The discrete-frequency STFT is obtained by replacing the above DTFTs by N-point DFTs, that is, evaluating them at the N DFT frequencies,

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

Thus, we set, $X_{k,m} \equiv X(\omega_k, mR)$, for, $k = 0, 1, \ldots, N-1$, and, $m = 0, 1, \ldots, M$, where the total number of segments is $M+1$,

(STFT)
$$\boxed{X_{k,m} = \sum_{n=0}^{N-1} x(mR+n)w(n)e^{-j\omega_k n} = \sum_{n=0}^{N-1} x_m(n)e^{-j\omega_k n}} \qquad (1)$$

$$k = 0, 1, \ldots, N-1\,, \quad m = 0, 1, \ldots, M$$

Given an input signal of length L_x, that is, $x(n)$, $n = 0, 1, \ldots, L_x - 1$, the number of segments can be calculated as follows, and then, prior to calculating the STFT, the signal $x(n)$ can be extended by padding enough zeros at its end until all frames have length N,

$$\begin{aligned} M &= \text{floor}\left(\frac{L_x}{R}\right) \\ L_{\text{ext}} &= MR + N \end{aligned} \quad \Rightarrow \quad x_{\text{ext}}(n) = \begin{cases} x(n), & 0 \le n \le L_x - 1 \\ 0, & L_x \le n \le L_{\text{ext}} - 1 \end{cases} \tag{2}$$

We will assume that this extension has been done and denote the extended signal by $x(n)$.

STFT can be displayed with a spectrogram plot

```
% stftgram.m - STFT spectrogram
%
% [t,f,S] = stftgram(x,R,N,fs)
%
% x  = input signal
% R  = hop size
% N  = FFT frame length
% fs = sampling rate              Ts = 1/fs
%
% t = m*R*Ts, m=0:M,              hop times in sec
% f = k*fs/N, k=0:N/2-1,          DFT frequencies in Hz
%
% S = STFT magnitude in dB,       make a surf plot
%
% size(S) = size(f) x size(t)
```

see also the built-in function: **spectrogram**

Spectrogam Example

$$x(t) = \begin{cases} \cos\left(2\pi f_1 t + \frac{1}{2}at^2\right), & 0 \le t \le T_0 \\ \cos(2\pi f_2 t), & T_0 < t \le 2T_0 \\ \cos(2\pi f_3 t) + \cos(2\pi f_4 t), & 2T_0 < t \le 3T_0 \end{cases}$$

$T_0 = 1000$ sec

$a = 3\pi 10^{-3}$ rad/sec^2

$f_1 = 1$ Hz, $f_2 = 2$ Hz, $f_3 = 3$ Hz, $f_4 = 4$ Hz

$f_s = 10$ Hz, $T_s = 1/f_s = 0.1$ sec

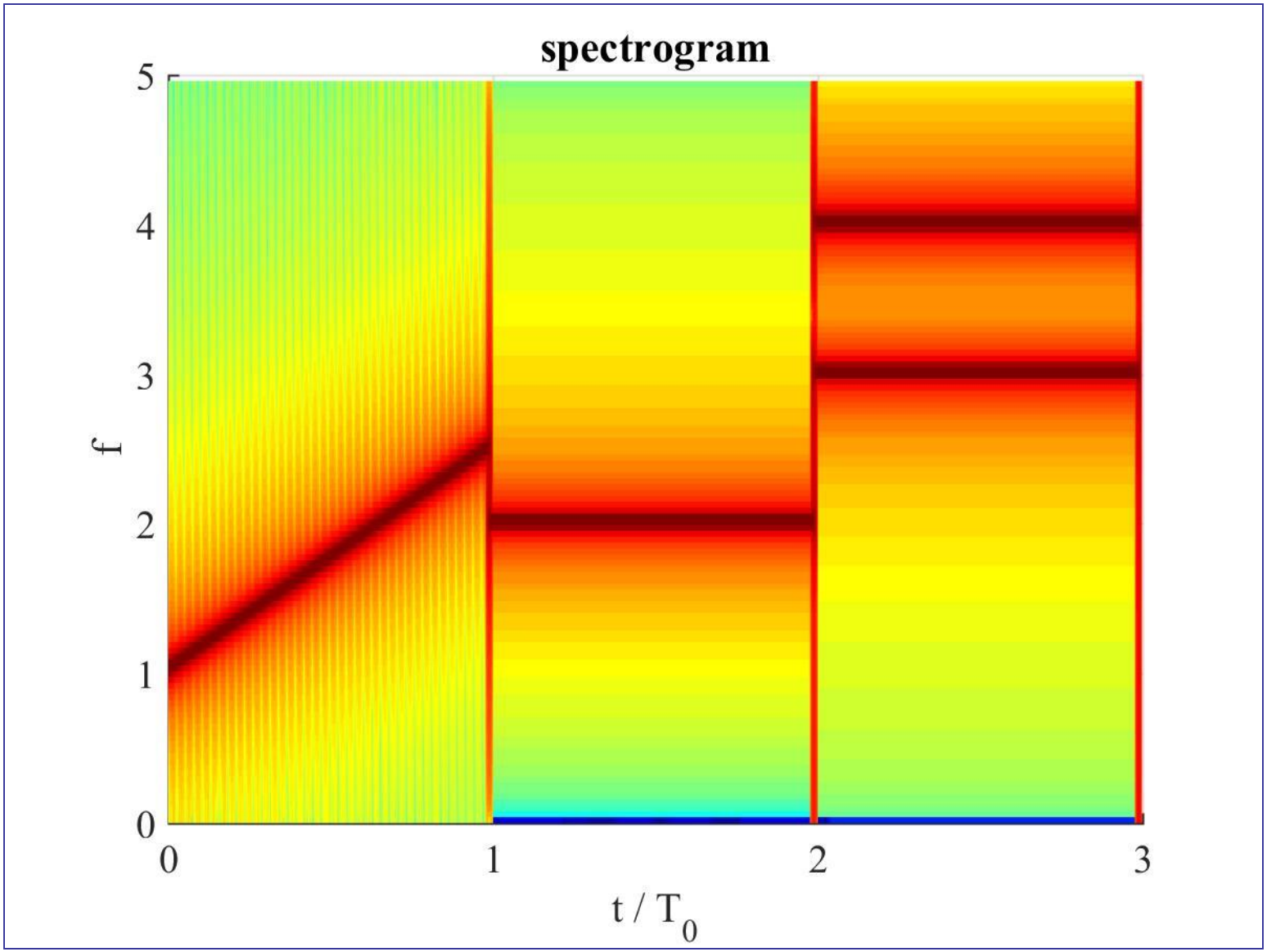
```
fs = 10; T = 1/fs;

T0 = 1e3;                                   % time scale, sec

a = 3*pi*1e-3;                              % rads/sec^2
f1 = 1;  f2 = 2;  f3 = 3;  f4 = 4;

T1 = T0; T2 = 2*T0; T3 = 3*T0;

x = @(t) cos(2*pi*f1*t + a*t.^2/2) .* (t>=0 & t<=T1) + ...
         cos(2*pi*f2*t) .* (t>T1 & t<=T2) + ...
         (cos(2*pi*f3*t) + cos(2*pi*f4*t)) .* (t>T2);

R = 20; N = 256;

tn = 0:T:T3; xt = x(tn);                    % sampled signal

[t,f,S] = stftgram(xt,R,N,fs);              % spectrogram
```

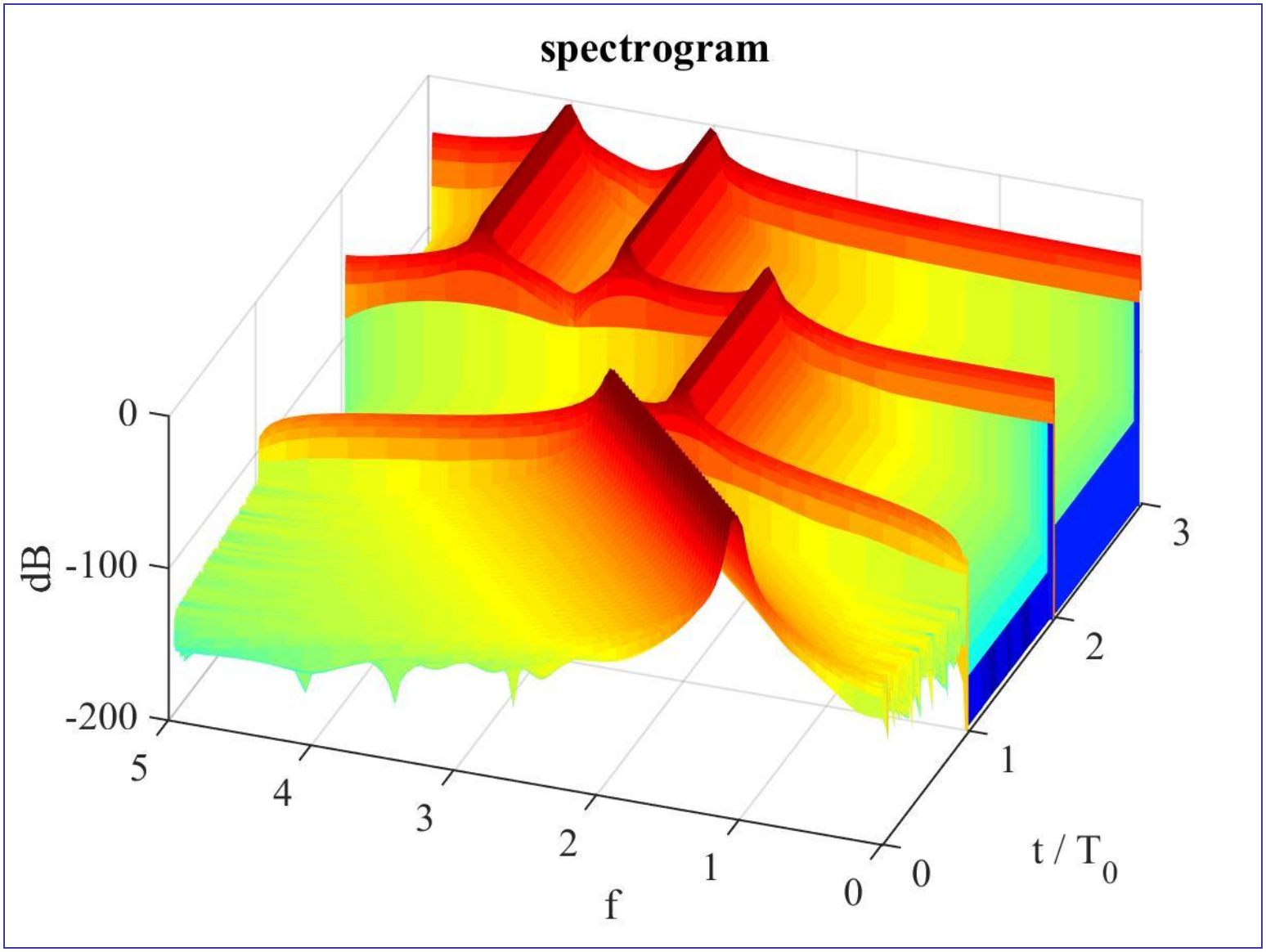
standard display as 2-D intensity plot

```
figure;
surf(t/T0,f,S,'edgecolor','none');      % plot vs. t/T0
axis tight; view(0,90); colormap(jet);
xlabel('t / T_0'); ylabel('f');
xaxis(0,3, 0:3); yaxis(0,5, 0:5);
title('spectrogram');
```

can also be displayed as 3-D surface plot

```
figure;
surf(t/T0,f,S,'edgecolor','none');      % plot vs. t/T0
view(-70, 50); colormap(jet);
xlabel('t / T_0'); ylabel('f');
xaxis(0,3, 0:3); yaxis(0,5, 0:5);
zaxis(-200,0, -200:100:0);
title('spectrogram');
```


spectrogram
5
4
3
2
1
0
f
0
1
2
3
t / T0


spectrogram
0
-100
-200
dB
5
4
3
2
1
0
f
0
1
2
3
t / T0

The STFT can be visualized as an $N \times (M+1)$–dimensional matrix whose columns are the N-point DFTs of the time segments $x_m(n)$,

$$\mathbf{X}_{\text{frames}} = \left[\mathbf{x}_0,\ \mathbf{x}_1,\ \ldots,\ \mathbf{x}_M\right], \quad \mathbf{x}_m = \begin{bmatrix} x_m(0) \\ x_m(1) \\ \vdots \\ x_m(N-1) \end{bmatrix}$$

$$\mathbf{X} = \left[\text{DFT}(\mathbf{x}_0),\ \text{DFT}(\mathbf{x}_1),\ \ldots,\ \text{DFT}(\mathbf{x}_M)\right]$$

In MATLAB, all the DFTs can be computed with a single FFT call,

$$\mathbf{X} = \text{FFT}(\mathbf{X}_{\text{frames}}) = \left[\text{FFT}(\mathbf{x}_0),\ \text{FFT}(\mathbf{x}_1),\ \ldots,\ \text{FFT}(\mathbf{x}_M)\right]$$

Assembling the overlapping frames into the frame matrix, $\mathbf{X}_{\text{frames}}$, can be done conveniently with the help of the **buffer** function.

But prior to calling the **fft** function, each column of $\mathbf{X}_{\text{frames}}$ must be windowed by the chosen window function $w(n)$ — this operation can also be done efficiently in MATLAB, as we discuss below.

ISTFT, OLA Reconstruction

The inverse STFT can be obtained by performing the inverse DFT, reconstructing the mth segment,

$$x(mR+n)w(n) = x_m(n) = \frac{1}{N}\sum_{k=0}^{N-1} X_{k,m}\, e^{j\omega_k n} \tag{3}$$

$$n = 0, 1, \ldots, N-1\,, \quad m = 0, 1, \ldots, M$$

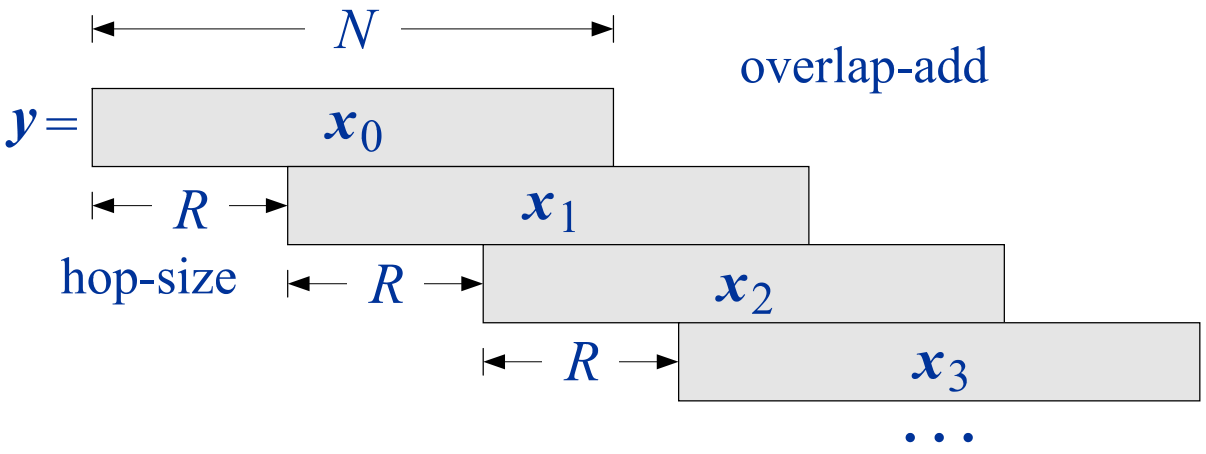
However, solving for $x(mR+n)$ requires division by $w(n)$, which is typically very small near its end points, and this would cause the amplification of even small amounts of noise that might be present.

For this reason, a better reconstruction procedure is by the overlap-add (OLA) method, that is, aligning the inverse DFTs $x_m(n)$ according to their absolute timing, starting at $n = mR$ for the m segment, and then adding them up,

$$\boxed{y(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR)} \quad \text{(ISTFT, OLA reconstruction)} \tag{4}$$

$$y(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR)$$

(ISTFT, OLA reconstruction)



The **ola** function can be used to carry out the overlap-add reconstruction operation

It can be shown (see O&S), that for most windows and many practical choices for R, the signal $y(n)$ is equal to $x(n)$ up to a constant factor that depends on the window and R.

But even if such window property, known as the *constant-overlap-add* (COLA) property, is not completely valid, one can still reconstruct $x(n)$ exactly by noting that $y(n)$ is related to $x(n)$, by $y(n) = x(n)\tilde{w}(n)$, where $\tilde{w}(n)$ is the overlapped-added version of the window,

$$\tilde{w}(n) = \sum_{m=-\infty}^{\infty} w(n - mR) \tag{5}$$

Thus, even if $\tilde{w}(n)$ is not constant in n, we can still solve for $x(n)$ by,

$$y(n) = x(n)\tilde{w}(n) = \sum_{m=-\infty}^{\infty} x_m(n - mR) \quad \Rightarrow$$

$$x(n) = \frac{\sum_{m=-\infty}^{\infty} x_m(n - mR)}{\sum_{m=-\infty}^{\infty} w(n - mR)} \tag{6}$$

O&S
Sect. 10.3 & 10.4

Since $\tilde{w}(n)$ is periodic in n with period R, it can be expanded in its R-point discrete Fourier series,

$$\boxed{\begin{aligned} \tilde{w}(n) &= \sum_{m=-\infty}^{\infty} w(n-mR) = \frac{1}{R}\sum_{r=0}^{R-1} W(\omega_r)\, e^{j\omega_r n}\,, \quad \omega_r = \frac{2\pi r}{R} \\ W(\omega_r) &= \sum_{n=0}^{N-1} w(n)\, e^{-j\omega_r n} = \text{DTFT of } w(n) \text{ evaluated at } \omega = \omega_r \end{aligned}} \tag{7}$$

Thus, the condition for the COLA property is that

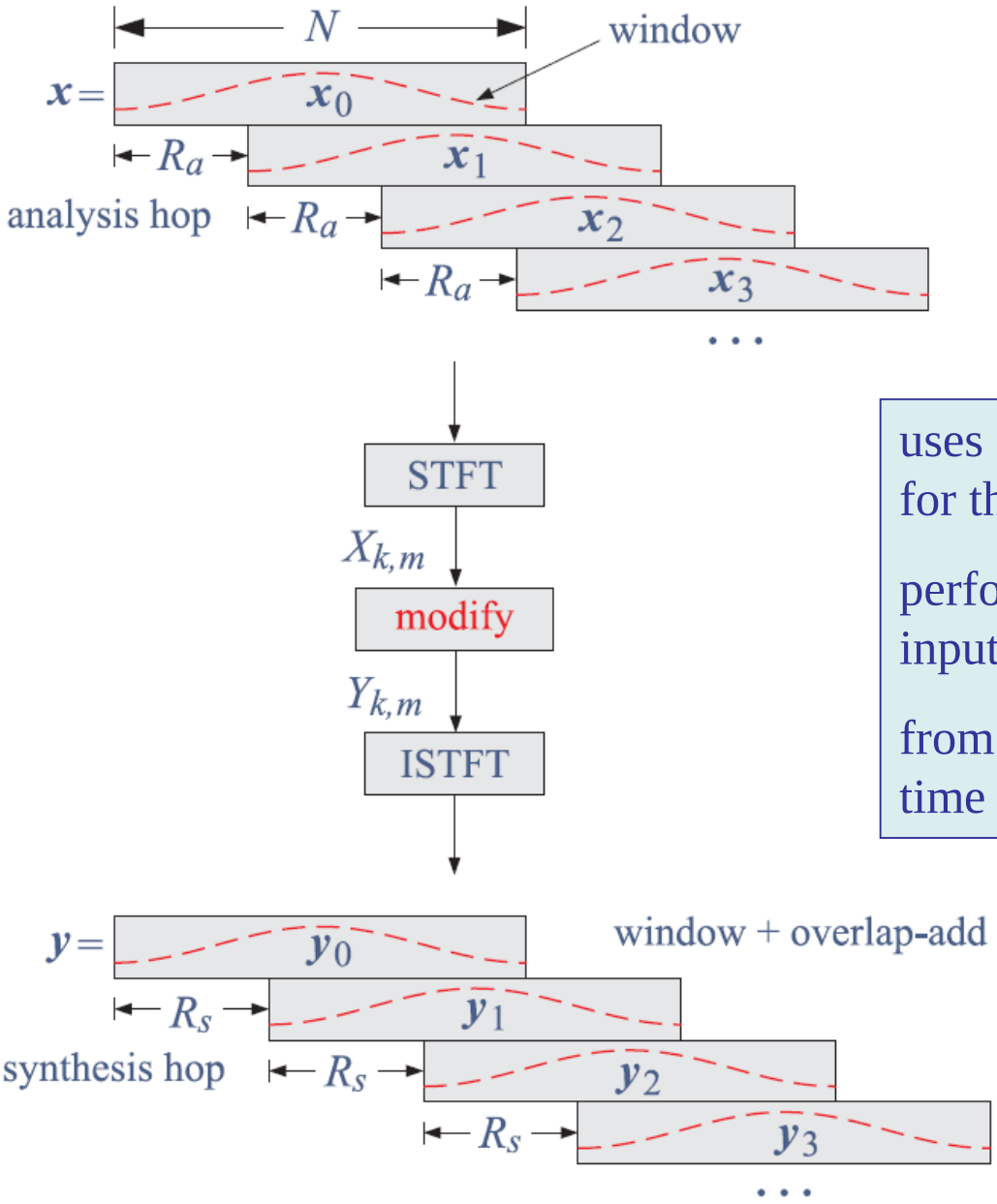
$$W(\omega_r) = 0\,, \quad r = 1, 2, \ldots, R-1$$

so that only the $r = 0$, or $\omega_r = 0$, term is present in Eq. (7), resulting into the constant value,

$$\tilde{w}(n) = \frac{W(0)}{R}$$

O&S
Sect. 10.3 & 10.4

STFT signal processing system



uses different hop sizes R_a , R_s
for the analysis and synthesis parts,

performs a modification operation on the input STFT, generating an output STFT,

from the ISTFT, it reconstructs the output time signal by overlap-add procedure

The following steps are carried out:

a. The input signal $x(n)$ is extended to length, $L_a = MR_a + N$, as in Eq. (2), and the output signal $y(n)$ is initialized to zero over its extended length, $L_s = MR_s + N$.

b. The STFT $X_{k,m}$ of $x(n)$ is computed with analysis hop size R_a,

$$X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n)w(n)e^{-j\omega_k n} \qquad (8)$$
$$0 \le k \le N-1\,, \quad 0 \le m \le M$$

c. Next, $X_{k,m}$ is modified according to some transformation, such as filtering, gain control, or phase modification as in the phase vocoder, resulting in an output STFT, say, $Y_{k,m}$.

d. Then, the inverse STFT of $Y_{k,m}$ is computed, and each segment is windowed by another length-N window, which is usually the same as the analysis window $w(n)$,

$$y_m(n) = w(n) \cdot \frac{1}{N} \sum_{k=0}^{N-1} Y_{k,m} e^{j\omega_k n}, \quad 0 \le n \le N-1 \tag{9}$$

e. The resulting windowed segments are overlapped-added with the synthesis hop R_s to obtain the synthesized transformed output $y(n)$,

$$y(n) = \sum_{m=-\infty}^{\infty} y_m(n - mR_s) \tag{10}$$

Computation

The STFT can be computed efficiently in MATLAB with a single FFT as follows. Assuming that $x(n)$ has been extended to length, $L_a = MR_a + N$, then with the help of the built-in MATLAB function **buffer**, the signal $x(n)$ can be rearranged into an $N \times (M+1)$ matrix whose columns are the time frames, $X_{\text{frames}} = \left[\mathbf{x}_0\,,\ \mathbf{x}_1\,,\ \ldots\,,\ \mathbf{x}_M\right]$, i.e., $X_{\text{frames}}(n,m) = x_m(n)$.

Then, the N-point FFT of that matrix will generate, after windowing, the FFTs of all the columns, resulting in the STFT matrix X,

$$\boxed{\begin{aligned} X_{\text{frames}} &= \textbf{buffer}\big(\mathbf{x},\ N,\ N - R_a,\ \texttt{'nodelay'}\big) \\ W &= \textbf{repmat}\big(\mathbf{w},\, 1,\, M+1\big) = \text{window} \\ X &= \textbf{fft}\big(W.{*}X_{\text{frames}},\, N\big) \end{aligned}} \qquad \text{(STFT)} \qquad (11)$$

where $\mathbf{w}$ is the N-dimensional *column vector* of the chosen window,

$$\mathbf{w} = \begin{bmatrix} w(0) \\ w(1) \\ w(2) \\ \vdots \\ w(N-1) \end{bmatrix} \quad \Rightarrow \quad W = \big[\underbrace{\mathbf{w},\ \mathbf{w},\ \ldots,\ \mathbf{w}}_{M+1 \text{ columns}}\big]$$

and W is its replication into an $N \times (M+1)$ matrix so that it can be multiplied point-wise by X_{frames}.

Once X is computed, it may be subjected to a transformation resulting in the output STFT matrix Y, which also has dimension $N \times (M+1)$. Its inverse can be carried out by a single IFFT call, resulting in the output matrix of time-frames,

$$\boxed{\begin{aligned} Y_{\text{frames}} &= \textbf{ifft}(Y, N) \\ y_m(n) &= Y_{\text{frames}}(n, m) = n\text{th element of } m\text{th column} \end{aligned}} \quad \text{(ISTFT)} \quad (12)$$

The ISTFT overlap-add operation of Eq. (4) may be implemented efficiently by the following iteration that reconstructs $y(n)$ segment-by-segment while windowing,

$$\boxed{\begin{aligned} &\text{for } m = 0, 1, 2, \ldots, M \\ &\quad \text{for } n = 0, 1, \ldots, N-1 \\ &\qquad y(mR_s + n) = y(mR_s + n) + y_m(n)w(n) \end{aligned}} \quad \text{(OLA reconstruction)}$$

(13)

where the n-loop can be vectorized and we must initialize $y(n)$ to zero, that is, $y(n) = 0$, for $n = 0, 1, \ldots, L_s - 1$, where, $L_s = MR_s + N$.

For example, see the following MATLAB code segment into which the windowing operation has been added, with the column vector **w** representing the N-dimensional window,

```
% define R, and, extract N,M from Y,
% define w = length-N column vector of window samples

N = size(Y,1);
M = size(Y,2) - 1;    % Y is Nx(M+1)

L = R*M + N;          % length of output y(n)

y = zeros(L,1);       % pre-allocate

n = (1:N)';           % column-vector

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

see included references in project-6

Phase Vocoder

The phase vocoder is an example of an STFT signal processing system, realized by Eqs. (11)–(13).

Its main use is to "replay" a signal, such as an audio recording, faster or slower without changing its frequency content. As for example, in playing the same piano piece more slowly, or singing a song faster without altering its frequencies.

We state the algorithm below, and later provide a justification. Several review papers are included as part of project-6.

The transformation step, $X_{k,m} \to Y_{k,m}$, to be carried out between Eqs. (11) and (12) is, in its simplest form, a modification of the phase of $X_{k,m}$, while preserving its magnitude. We have in polar form,

$$\begin{aligned} X_{k,m} &= |X_{k,m}|\, e^{j\Phi_{k,m}} \\ Y_{k,m} &= |Y_{k,m}|\, e^{j\Psi_{k,m}} \end{aligned} \tag{14}$$

where the magnitudes are preserved,

$$|Y_{k,m}| = |X_{k,m}| \tag{15}$$

and the output phases $\Psi_{k,m}$ are computed recursively from the input phases $\Phi_{k,m}$, as follows, where the k-loop may be vectorized,

$$
\begin{aligned}
&\text{for } k = 0, 1, \ldots, N-1, \\
&\qquad \omega_k = \frac{2\pi k}{N} \\
&\qquad \Psi_{k,0} = \Phi_{k,0} \\
&\qquad \text{for } m = 1, 2, \ldots, M, \\
&\qquad\qquad \Delta\omega_{k,m} = \frac{1}{R_a} \cdot \Big[\Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k\Big]_{\bmod 2\pi} \\
&\qquad\qquad \omega_{k,m} = \omega_k + \Delta\omega_{k,m} \\
&\qquad\qquad \Psi_{k,m} = \Psi_{k,m-1} + R_s \omega_{k,m}
\end{aligned}
\tag{16}
$$

(phase modification)

The notation, $\left[x\right]_{\mathrm{mod}\,2\pi}$

stands for the phase unwrapping of the angle x modulo 2π, that is, adding to, or subtracting from, x enough multiples of 2π until it lies in the symmetric Nyquist interval, $-\pi \leq x \leq \pi$.

It can be implemented easily by the following vectorized anonymous MATLAB function, **mod2pi**(x),

```
mod2pi = @(x) mod(x+pi, 2*pi) - pi;
```

In summary, the complete phase vocoder algorithm is described by Eqs. (11), (15), (16), (12), and (13), in that order.

Time-Scale Modification

By choosing different hop sizes R_a, R_s, the duration of the output signal $y(n)$ can be made longer or shorter than that of the input $x(n)$, depending on whether $R_s > R_a$ or $R_s < R_a$, respectively.

The purpose of the phase modification equations (15) and (16) is to preserve the frequency content of the signal under such change in duration. We may define the speed-up factor by,

$$\boxed{r = \frac{R_a}{R_s}} \quad \text{(speed-up factor)}$$
$$\boxed{\frac{1}{r} = \frac{R_s}{R_a}} \quad \text{(time-stretching factor)} \tag{17}$$

so that $R_s = R_a/r$, and $r > 1$ corresponds to a faster rendition (i.e. shorter duration) of the signal, and $r < 1$, slower rendition (i.e., longer duration).

If we specify R_a, r, then we may calculate R_s by rounding,

$$R_s = \text{round}\left(\frac{R_a}{r}\right)$$

or conversely, as is preferred in practice, if we specify, R_s, r, then,

$$R_a = \text{round}\big(r\, R_s\big)$$

Phase Vocoder Model

Here, we provide a simplified justification of the phase vocoder algorithm. Given a sinusoidal signal of varying amplitude and varying phase,

$$x(t) = A(t)\, e^{j\Phi(t)}$$

the instantaneous analog frequency is defined as the derivative of the phase:

$$\Omega(t) = \dot{\Phi}(t)$$

Considering the signal values at two nearby time instants, t and $t + \Delta t$, we may expand the phase to first-order in Δt and approximate the phase and the instantaneous frequency as,

$$\begin{aligned} \Phi(t + \Delta t) &= \Phi(t) + \dot{\Phi}(t)\, \Delta t = \Phi(t) + \Omega(t)\, \Delta t \\ \Omega(t + \Delta t) &= \Omega(t) \end{aligned} \tag{18}$$

The phase vocoder is based on the implicit assumption that the signal is a sum of such sinusoidal terms with varying amplitudes and phases,

$$x(t) = \sum_i A_i(t)\, e^{j\Phi_i(t)}\,, \quad \Omega_i(t) = \dot{\Phi}_i(t) \tag{19}$$

The main objective is to ensure that the instantaneous frequencies contained in the signal are preserved in going from the overlapped analysis frames at hop size R_a to the overlapped synthesis frames at hop size R_s. Ideally, the STFTs of the input and output frames would be:

$$X_{k,m} = \sum_{n=0}^{N-1} x(mR_a + n)w(n)e^{-j\omega_k n}$$

$$Y_{k,m} = \sum_{n=0}^{N-1} x(mR_s + n)w(n)e^{-j\omega_k n}$$

where ω_k are the DFT frequencies,

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

and, we assumed that the signal $x(t)$ was sampled at a rate $f_s = 1/T$, so that the time intervals that correspond to the length-N window of the mth frame are as follows, for the analysis and synthesis cases,

$$t^a_{mn} = mR_aT + nT = t^a_m + nT \qquad \text{(analysis)}$$

$$t^s_{mn} = mR_sT + nT = t^s_m + nT \qquad \text{(synthesis)}$$

where, $t^a_m = mR_aT$, and, $t^s_m = mR_sT$, are the beginning times of the mth segments. It follows that the sampled signal for the analysis frames, $x(mR_s + n)$, would be according to Eq. (19),

$$x(t^a_m + nT) = \sum_i A_i(t^a_m + nT)\, e^{j\Phi_i(t^a_m + nT)}$$

Assuming a small enough sampling interval T, we may expand the signal phases using the approximation of Eq. (18). Assuming also that the signal amplitudes vary slowly across each windowed frame, we obtain the following approximations, over the length-N window of the mth frame,

$$A_i(t_m^a + nT) \approx A_i(t_m^a), \quad 0 \le n \le N-1$$

$$\Phi_i(t_m^a + nT) = \Phi_i(t_m^a) + (nT)\Omega_i(t_m^a)$$

Defining the digital instantaneous frequencies in [rads/sample],

$$\omega_i(t_m^a) = \Omega_i(t_m^a)T$$

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

then, the phase approximations can be written as,

$$\Phi_i(t_m^a + nT) = \Phi_i(t_m^a) + n\omega_i(t_m^a)$$

so that the signal can be approximated as follows within the mth frame,

$$x(t_m^a + nT) \approx \sum_i A_i(t_m^a) e^{j\Phi_i(t_m^a) + jn\omega_i(t_m^a)} \qquad (20)$$

Inserting Eq. (20) into the analysis STFT, we have,

$$\begin{aligned} X_{k,m} &= \sum_{n=0}^{N-1} x(mR_a + n)\, w(n)\, e^{-j\omega_k n} \\ &= \sum_{n=0}^{N-1} \sum_i A_i(t_m^a)\, e^{j\Phi_i(t_m^a) + jn\omega_i(t_m^a)}\, w(n)\, e^{-j\omega_k n} \\ &= \sum_i A_i(t_m^a)\, e^{j\Phi_i(t_m^a)} \sum_{n=0}^{N-1} e^{jn\omega_i(t_m^a)} e^{-jn\omega_k}\, w(n) \end{aligned}$$

where the summation over n is recognized to be the frequency-shifted DTFT of the window $w(n)$,

$$\hat{W}(\omega) = \sum_{n=0}^{N-1} e^{-j\omega n} w(n) = \text{DTFT of } w(n)$$

$$\hat{W}\left(\omega_k - \omega_i(t_m^a)\right) = \sum_{n=0}^{N-1} e^{jn\omega_i(t_m^a)} e^{-jn\omega_k}\, w(n)$$

Because we always assume that the window $w(n)$ is real-valued and symmetric abouts its middle, it follows that its DTFT can be factored into a real and even function of ω, and a phase part corresponding to a delay by $(N-1)/2$ samples,

$$\hat{W}(\omega) = W(\omega)e^{-j\omega(N-1)/2}, \quad W(\omega) = W(-\omega) = \text{real-valued}$$

for example, we recall the rectangular window case,

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

it follows that the STFT of the analysis frames will be,

$$X_{k,m} = \sum_i A_i(t_m^a)\, e^{j\Phi_i(t_m^a)}\, W\big(\omega_i(t_m^a) - \omega_k\big)\, e^{j(\omega_i(t_m^a)-\omega_k)(N-1)/2} \qquad (21)$$

For large N, the function $W(\omega)$ is highly concentrated about $\omega = 0$, and therefore, only that sinusoidal term i whose instantaneous frequency $\omega_i(t_m^a)$ falls within the kth DFT bin will effectively contribute to the above sum, that is, we can keep approximately only the $i = k$ term,

$$X_{k,m} \approx A_k(t_m^a) e^{j\Phi_k(t_m^a)} W\left(\omega_k(t_m^a) - \omega_k\right) e^{j(\omega_k(t_m^a)-\omega_k)(N-1)/2} \tag{22}$$

In this expression, the frequency $\omega_k(t_m^a)$ is not equal to ω_k but it is nearby, i.e., we can introduce a deviation from ω_k to be determined,

$$\omega_k(t_m^a) = \omega_k + \Delta\omega_{k,m} \tag{23}$$

In order to obtain the phase of the STFT, that is,

$$X_{k,m} = |X_{k,m}| e^{j\Phi_{k,m}}$$

we observe that Eq. (22) is separated into real factors and phases, so that,

$$\Phi_{k,m} = \Phi_k(t_m^a) + \left(\omega_k(t_m^a) - \omega_k\right)(N-1)/2 \tag{24}$$

Our objective, eventually, is to determine the instantaneous frequecies, $\omega_k(t^a_m)$, from the computed STFT phases $\Phi_{k,m}$. To do so, we consider how these phases change from frame to frame, i.e., from time, $t^a_{m-1} = (m-1)R_aT$, to time, $t^a_m = mR_aT$. But since, we have,

$$t^a_m = t^a_{m-1} + R_aT$$

we may use the approximate expansion of Eq. (18) to write,

$$\Phi_k(t^a_m) = \Phi_k(t^a_{m-1} + R_aT) \approx \Phi_k(t^a_{m-1}) + R_aT\,\Omega_k(t^a_m)\,, \quad \text{or,}$$

$$\Phi_k(t^a_m) \approx \Phi_k(t^a_{m-1}) + R_a\omega_k(t^a_m)$$

and also from Eq. (18), we have approximately, $\omega_k(t^a_{m-1}) = \omega_k(t^a_m)$, so that,

$$\begin{aligned}
\Phi_{k,m} &= \Phi_k(t^a_m) + \big(\omega_k(t^a_m) - \omega_k\big)\,(N-1)/2 \\
&= \Phi_k(t^a_{m-1}) + R_a\,\omega_k(t^a_m) + \big(\omega_k(t^a_m) - \omega_k\big)\,(N-1)/2 \\
&= \Phi_k(t^a_{m-1}) + \big(\omega_k(t^a_{m-1}) - \omega_k\big)\,(N-1)/2 + R_a\,\omega_k(t^a_m) \\
&= \Phi_{k,m-1} + R_a\,\omega_k(t^a_m)\,, \quad \text{or,} \\
\Phi_{k,m} &= \Phi_{k,m-1} + R_a\,\omega_k(t^a_m) \\
\Phi_{k,m} &= \Phi_{k,m-1} + R_a\,(\omega_k + \Delta\omega_{k,m})
\end{aligned}$$

thus,

$$\boxed{\begin{aligned} \Phi_{k,m} &= \Phi_{k,m-1} + R_a\,(\omega_k + \Delta\omega_{k,m}) \\ R_a\,\Delta\omega_{k,m} &= \Phi_{k,m} - \Phi_{k,m-1} + R_a\,\omega_k \end{aligned}} \tag{25}$$

In a similar fashion, we can show that the synthesis STFTs,

$$Y_{k,m} = |Y_{k,m}|\,e^{j\Psi_{k,m}}$$

will satisfy similar recursions which will ensure that the instantaneous frequencies are preserved:

$$\Psi_{k,m} = \Psi_{k,m-1} + R_s\,(\omega_k + \Delta\omega_{k,m}) \tag{26}$$

Therefore, if we can solve Eq. (25) for $\Delta\omega_{k,m}$, we can reconstruct the output STFTs recursively. To solve for, $\Delta\omega_{k,m}$, we note that in the equation,

$$R_a \, \Delta\omega_{k,m} = \Phi_{k,m} - \Phi_{k,m-1} + R_a \omega_k$$

the right-hand side, being a phase, is defined up to a multiple of 2π. Thus, phase-unwrapping it mod-2π to fall within the standard Nyquist interval $[-\pi, \pi]$, we obtain,

$$R_a \, \Delta\omega_{km} = \left[\Phi_{k,m} - \Phi_{k,m-1} + R_a \omega_k\right]_{\text{mod}2\pi}$$

thus,

$$\boxed{\Delta\omega_{k,m} = \frac{1}{R_a} \cdot \left[\Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k\right]_{\text{mod}\,2\pi}} \tag{27}$$

To summarize, we start by carrying out an analysis STFT with hop-size R_a,

$$X_{k,m} = |X_{k,m}| e^{j\Phi_{k,m}}$$

then, we construct the synthesis STFTs, $Y_{k,m}$, with hop-size R_s by preserving their magnitudes,

$$|Y_{k,m}| = |X_{k,m}|$$

and constructing their phases, $\Psi_{k,m}$ recursively by the phase-modification algorithm of Eq. (16),

$$\begin{aligned}
&\text{for } k = 0, 1, \ldots, N-1, \\
&\qquad \omega_k = \frac{2\pi k}{N} \\
&\qquad \Psi_{k,0} = \Phi_{k,0} \\
&\qquad \text{for } m = 1, 2, \ldots, M, \\
&\qquad\qquad \Delta\omega_{k,m} = \frac{1}{R_a} \cdot \Big[\Phi_{k,m} - \Phi_{k,m-1} - R_a \omega_k\Big]_{\text{mod}\,2\pi} \\
&\qquad\qquad \Psi_{k,m} = \Psi_{k,m-1} + R_s\,(\omega_k + \Delta\omega_{k,m})
\end{aligned} \tag{28}$$

Pitch-Scale Modification

Pitch shifting refers to scaling all frequencies by a factor r, that is, replacing, $f \to r\,f$, without altering the duration of the signal, as for example in playing a piano piece an octave higher. This is not the same as frequency translation in which all frequencies are shifted by a fixed amount, $f \to f + f_0$.

If you have an audio signal $x(n)$ recorded at a sampling rate f_s and you replay it a rate that is r times faster, $f_s' = r\,f_s$, then both the duration and the pitch will be altered, with the duration becoming r times shorter, and the pitch becoming r times higher—this is known as the "chipmunk" effect.

In order to perform pitch shifting without changing the duration of the signal, we may combine a phase vocoder time-scale modification with a resampling operation. In other words, we may "chipmunk" the signal and then correct its duration with a phase vocoder.

For example, suppose $r > 1$, and we resample $x(n)$ at the rate f_s/r, and play it back at f_s/r, then it would sound like the original, but if we play it back at $f_s = r \cdot f_s/r$, then it would have a pitch r times higher but it will also have r times shorter duration.

Thus, if we follow this operation with a phase vocoder with a time-stretching factor r, or equivalently, speed-up factor $1/r$, we would restore the original duration, while not affecting the already pitch-shifted frequencies.

Alternatively, we may reverse these operations. First we apply a phase vocoder with speed-up factor of $1/r$. Now the signal will have r times longer duration, but its pitch will not have shifted if played at rate f_s.

If we now resample this at the rate f_s/r and played it a rate f_s/r, it would sound the same as that longer version, but if we play it back at $f_s = r \cdot f_s/r$, it will be pitch-shifted by a factor of r, and its duration will be scaled down by a factor of r to the original duration.

The two alternative approaches are depicted below, where it is advantageous to apply the top version when $r > 1$, and the bottom, when $r < 1$.

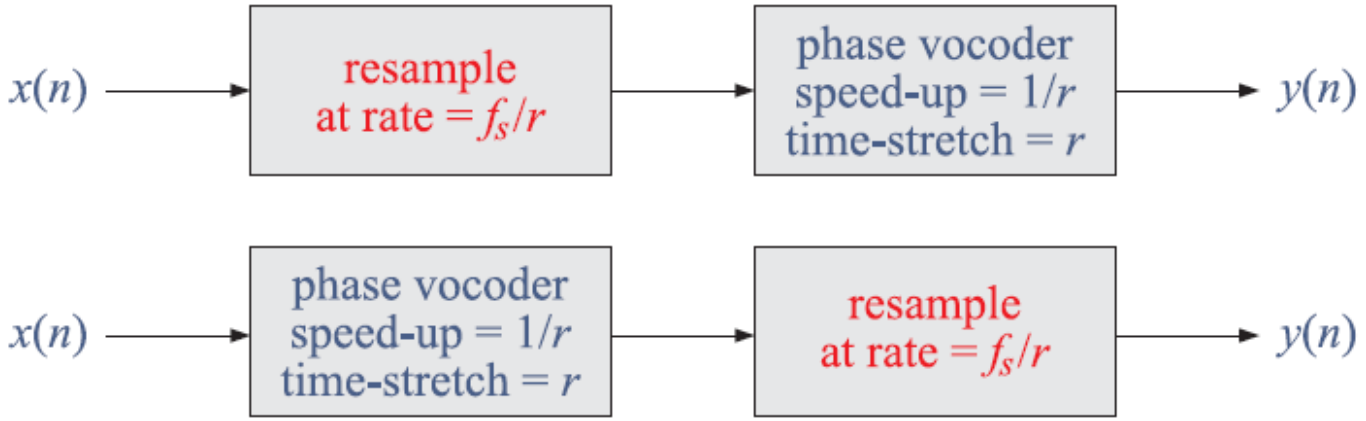
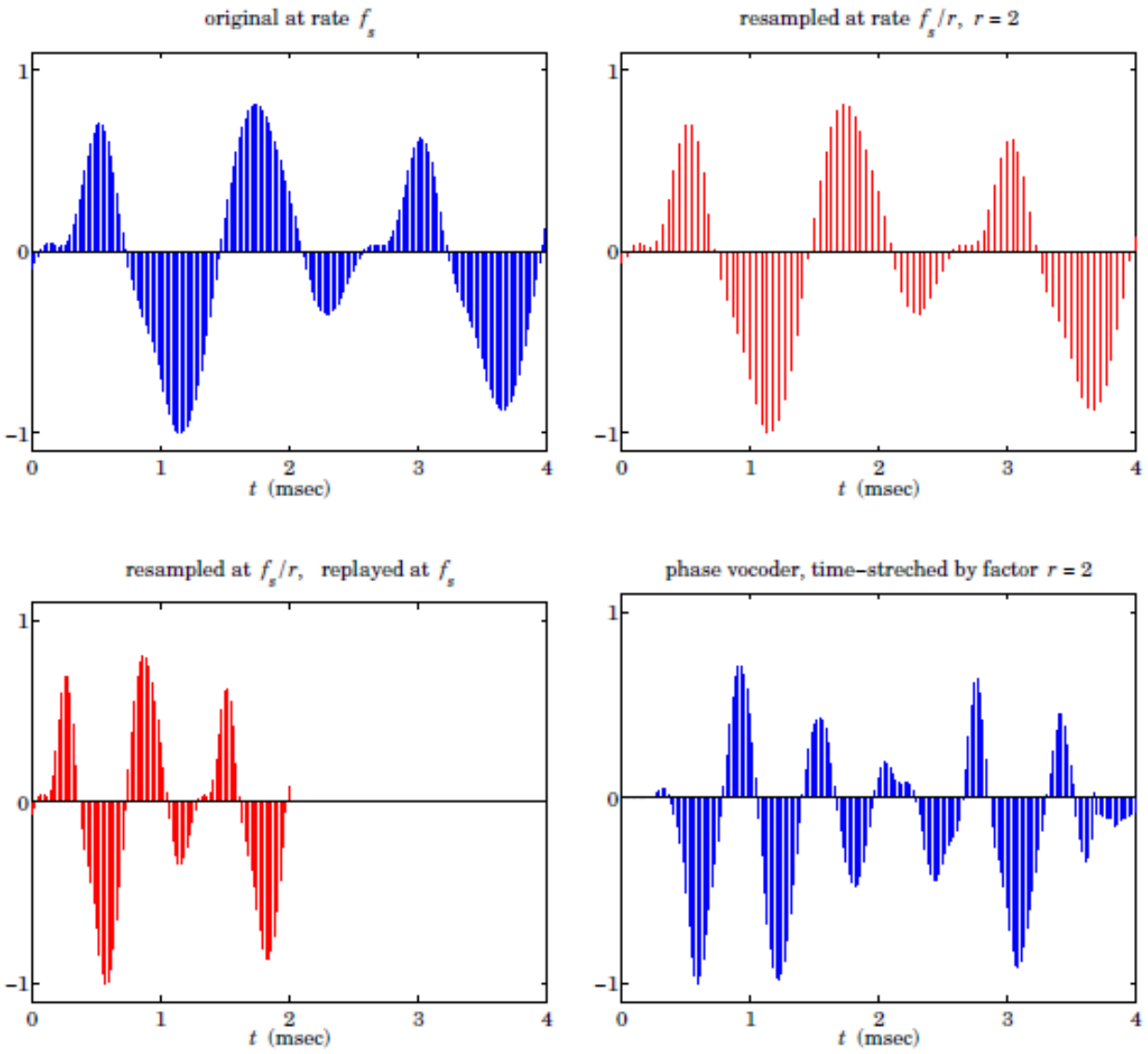
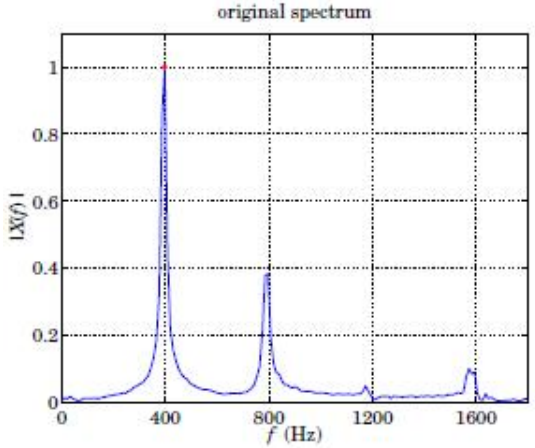
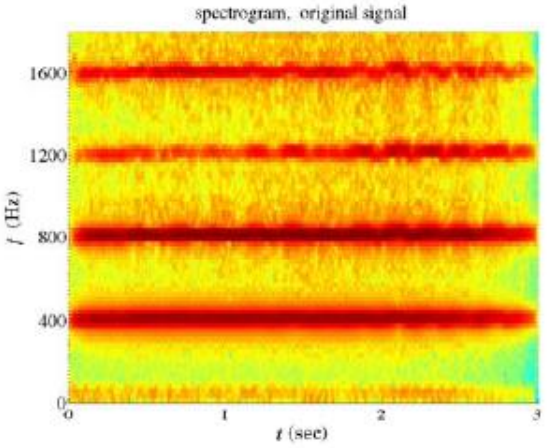


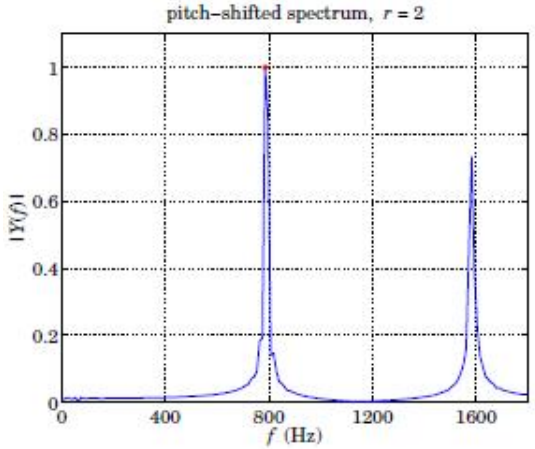
Fig. 2 Pitch shifting by a factor of r.

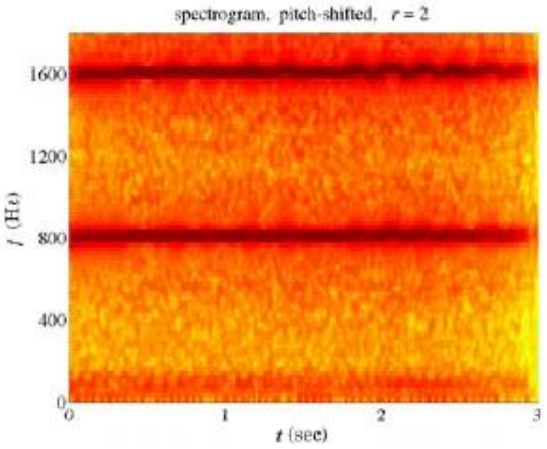
pitch-shifting example

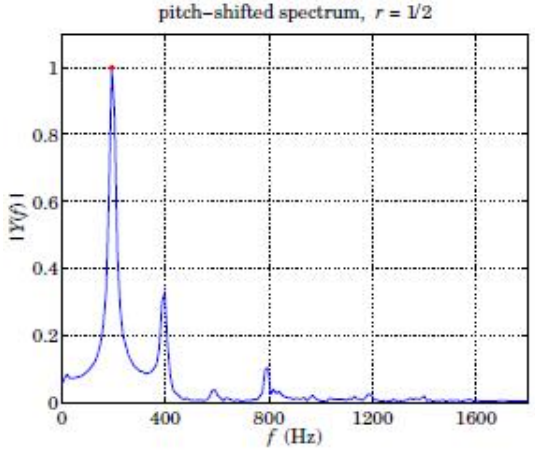


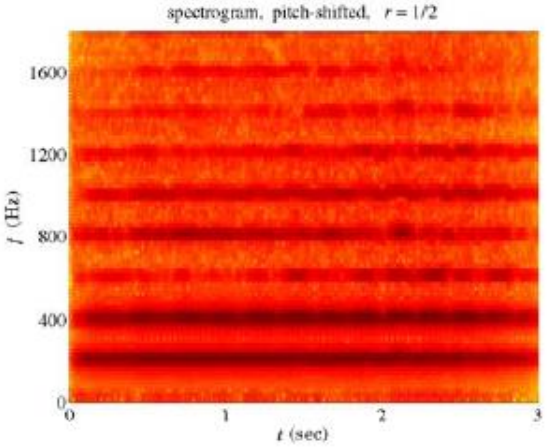

original spectrum
|X(f)|
f (Hz)
0
0.2
0.4
0.6
0.8
1
0
400
800
1200
1600

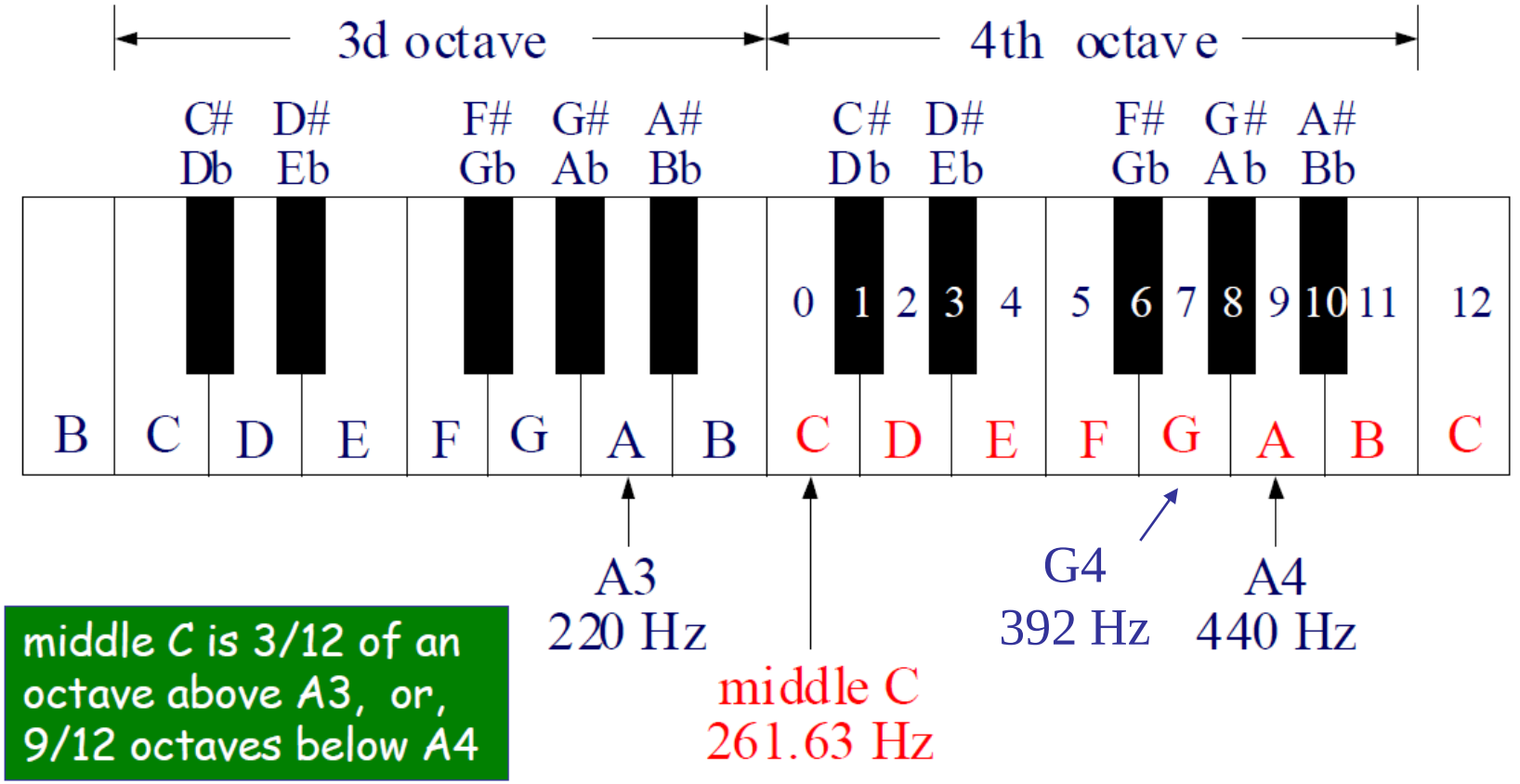

spectrogram, original signal
f (Hz)
t (sec)
0
400
800
1200
1600
0
1
2
3


pitch–shifted spectrum, r = 2
|Y(f)|
f (Hz)
0
0.2
0.4
0.6
0.8
1
0
400
800
1200
1600


spectrogram, pitch-shifted, r = 2
f (Hz)
t (sec)
0
400
800
1200
1600
0
1
2
3


pitch–shifted spectrum, r = 1/2
|Y(f)|
f (Hz)
0
0.2
0.4
0.6
0.8
1
0
400
800
1200
1600


spectrogram, pitch-shifted, r = 1/2
f (Hz)
t (sec)
0
400
800
1200
1600
0
1
2
3



middle C is 3/12 of an octave above A3, or, 9/12 octaves below A4

$$261.63 = 220 \cdot 2^{3/12} = 440 \cdot 2^{-9/12}$$

4th octave keys, k = 0:12, MATLAB index = k+1 = 1:13

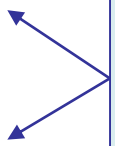
major keys are a subset of k, m = [0, 2, 4, 5, 7, 9, 11, 12]

```
% stftgram.m - STFT spectrogram
%
% [t,f,S] = stftgram(x,R,N,fs)
%
% x  = input signal
% R = hop size
% N  = FFT frame length
% fs = sampling rate
%
% t = m*R*Ts, m=0:M,          hop times in sec
% f = k*fs/N, k=0:N/2-1,      non-negative DFT frequencies in Hz
% S = STFT magnitude in dB,   if nargout==0, make a surf plot

function [t,f,S] = stftgram(x,R,N,fs)

X = stft(x,R,N).';
M = size(X,2)-1;

k = 0:N/2-1; f = k*fs/N; t = (0:M)*R/fs;

Xmag = abs(X(k+1,:)); S = 20*log10(Xmag/max(max(Xmag)));

if nargout==0
   surf(t,f,S,'edgecolor','none');
   axis tight; view(0,90); colormap(jet);
   xlabel('{\itt} (sec)'); ylabel('{\itf}  (Hz)');
end
```

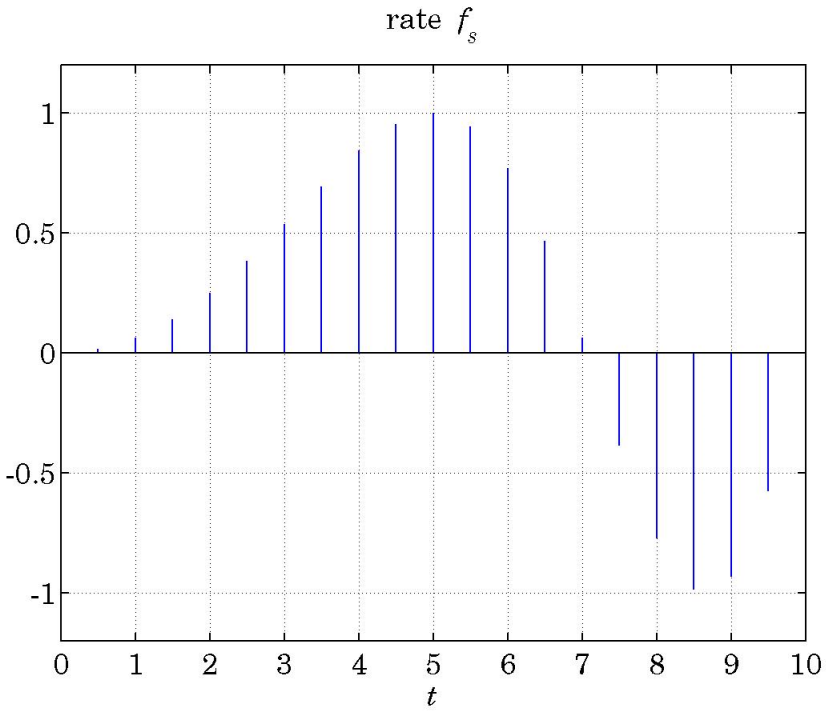
```
Ra = 256;      % input hop-size
N  = 2048;     % FFT frame length

[x,FS] = audioread('flute2.wav'); x = x(:,1).';

r = 2;

y = pitchmod(x,r,Ra,N);   % pitch-shifted signal

soundsc([x, zeros(1,FS), y], FS);  % play sounds

figure; stftgram(x,Ra,N,FS);
xaxis(0,3,0:3);  yaxis(0,1800, 0:400:1800);
title('spectrogram,  original signal')

Rs = round(Ra/r);          % output hop-size

figure;  stftgram(y,Rs,N,FS);
xaxis(0,3,0:3);  yaxis(0,1800, 0:400:1800);
title('spectrogram,  pitch-shifted,  {\itr} = 2')
```

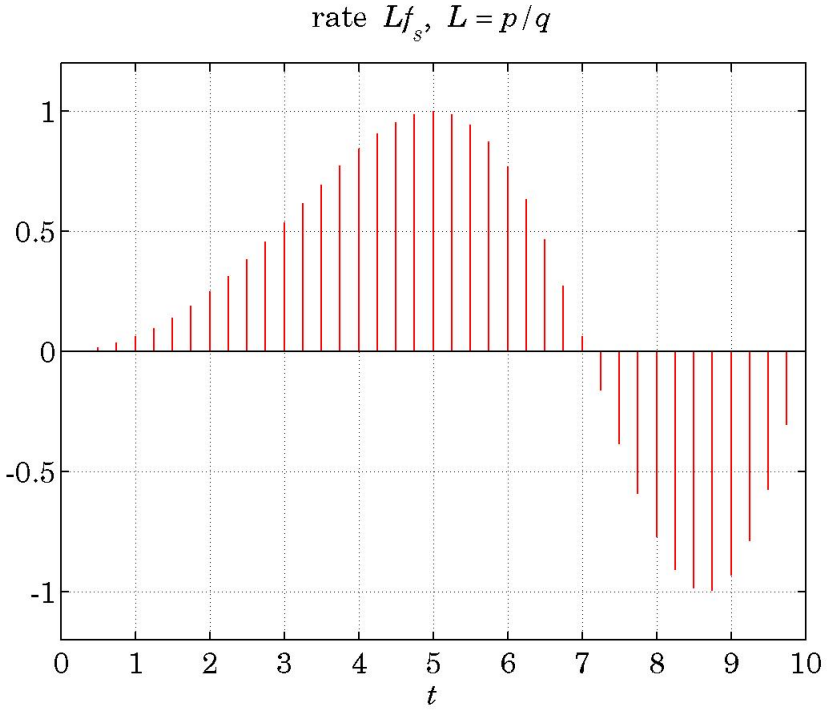
resampling example

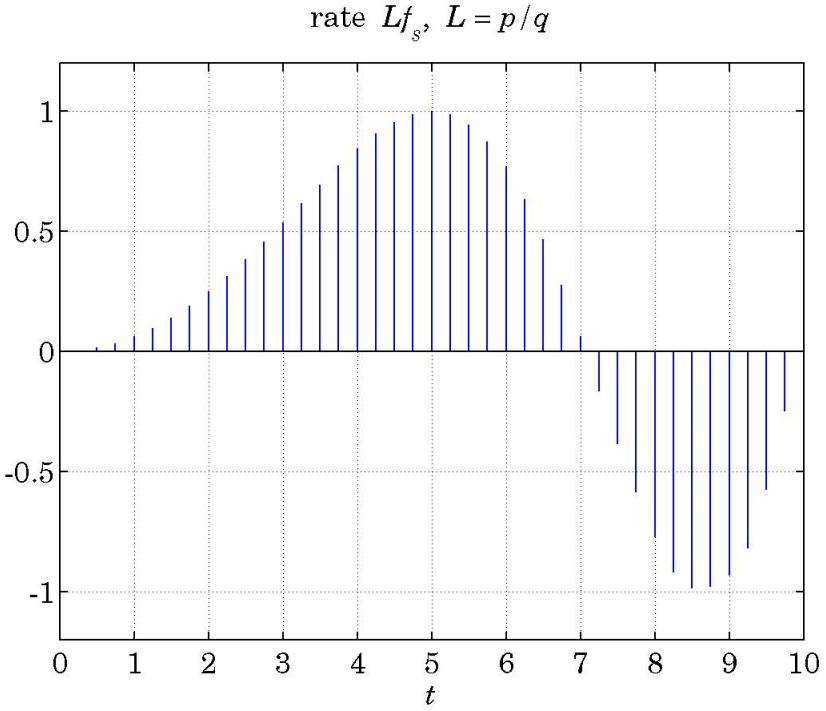
```
T = 10;                 % total duration, sec
fs = 2; Ts = 1/fs;      % sampling interval Ts = 0.5 sec

p=2; q=1;               % resampling factor L = p/q
% p=1; q=2;

L=p/q; fpq = L*fs; Tpq = Ts/L;     % Tpq = 0.25 sec

t = 0:Ts:T;                    % t sampled at fs
tpq = 0:Tpq:T;                 % t sampled at fpq = fs * p/q

a = 2*pi/50;                   % a = chirp parameter
x = sin(a/2 * t.^2);           % sampled at fs
xpq = sin(a/2 * tpq.^2);       % sampled at fpq

figure; stems(t,x,'b');
title('rate  {\itf_s}');

figure; stems(tpq,xpq,'r');
title('rate  {\itLf_s},  {\itL = p/q}');

y = resample(x,p,q);
ty = (0:length(y)-1)*Tpq;

figure; stems(ty,y,'b');
title('rate  {\itLf_s},  {\itL = p/q}');
```



slight differences due to the internal FIR filtering in **resample**


rate f_s
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
t


rate Lf_s, $L = p/q$
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
t


rate Lf_s, $L = p/q$
1
0.5
0
-0.5
-1
0 1 2 3 4 5 6 7 8 9 10
t