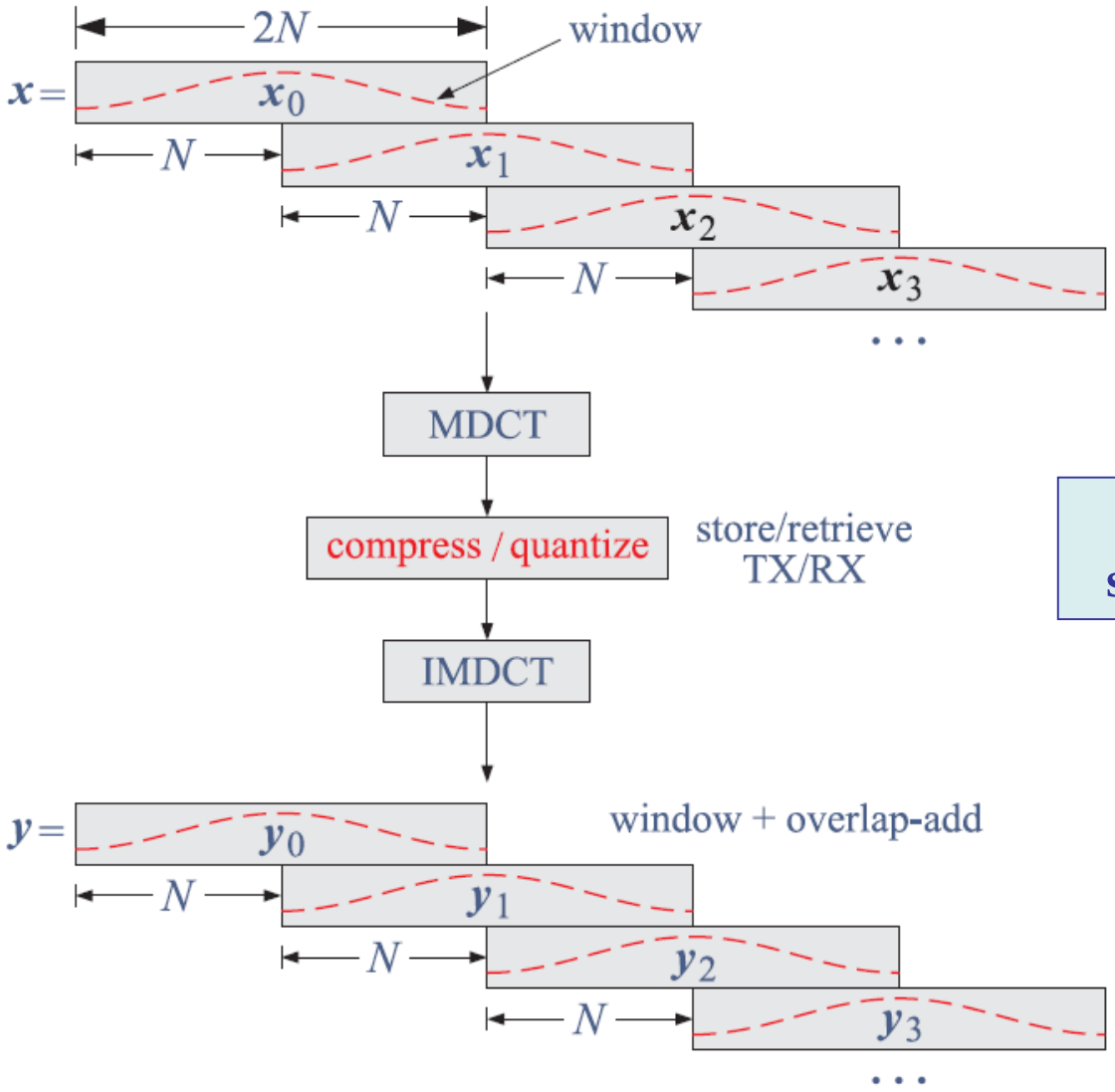
DSA – March 22, 2021

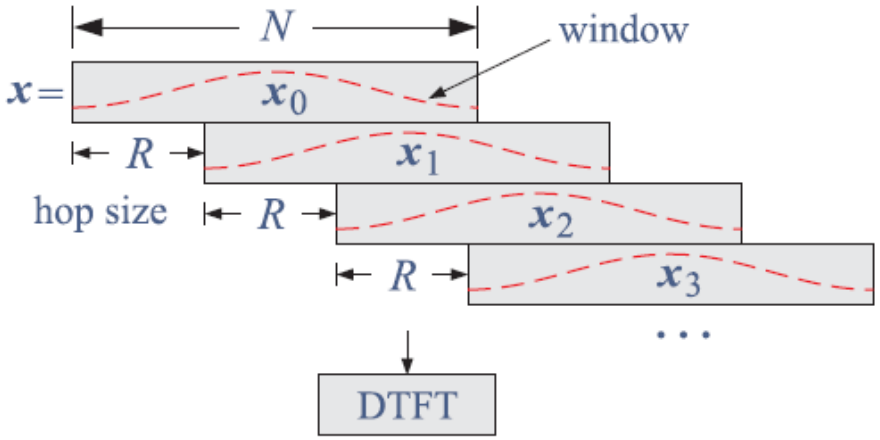
Topics: STFT review, DCT, IDCT, fast DCT, data compression with DCT, MDCT, IMDCT, TDAC property, Pricen-Bradley windows, data compression with MDCT.



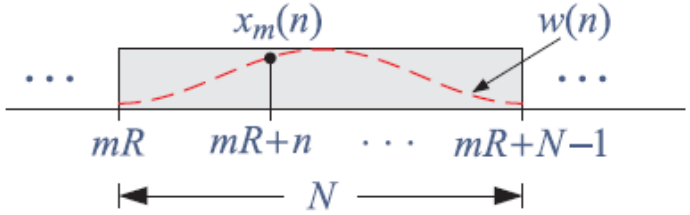
MDCT / TDAC
signal compression system

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 $x(n)$.

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,

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

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

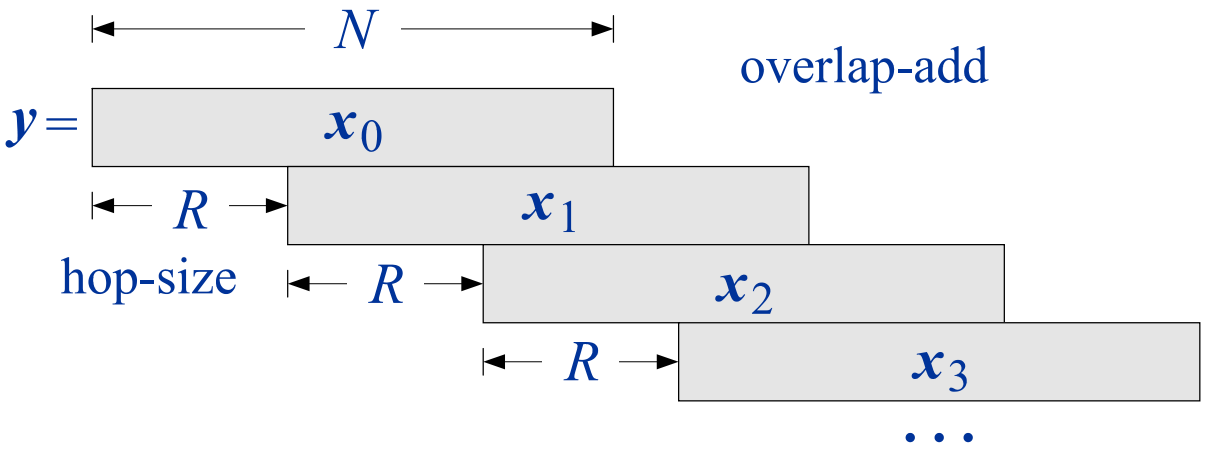
However, solving for $x(mR+n)$ requires division by $w(n)$, which is typically very small near its end points, and this would cause the amplification of even small amounts of noise that might be present.

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

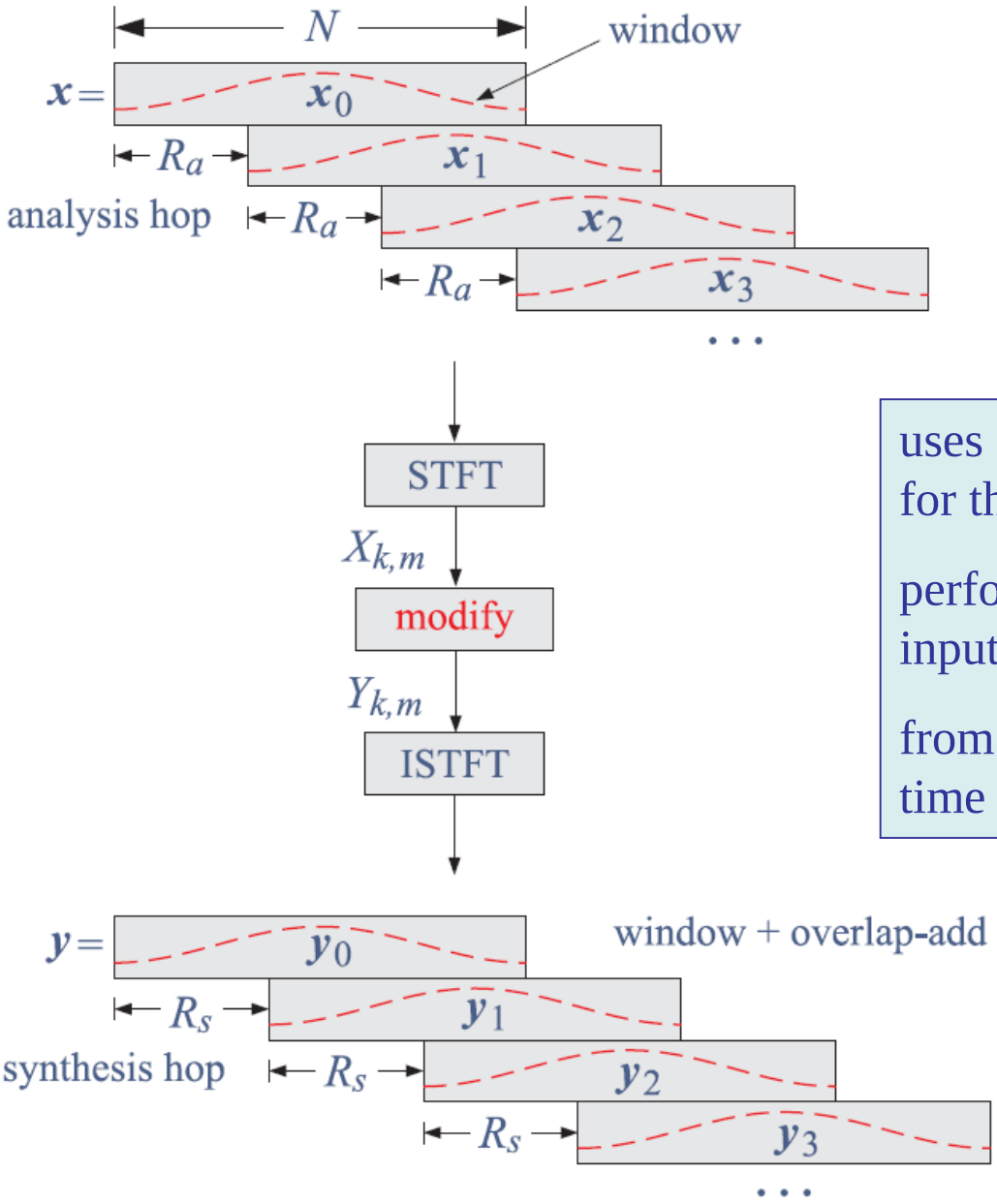
$$W(\omega_r) = 0\,, \quad r = 1, 2, \ldots, R-1$$

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
```

Project 7 – Data Compression with DCT
MDCT and Time-Domain Aliasing Cancellation

The purpose of project-7 is to study the data compression properties of the discrete cosine transform (DCT), implement a DCT compression system in MATLAB, and apply it to audio and image data.

In addition, a compression system based on the modified DCT (MDCT) and its inverse (IMDCT) is studied and implemented in MATLAB, including the related time-domain aliasing cancellation (TDAC) property, which is key in most current audio compression systems, such as MP3, AAC, WMA, Vorbis, and others.

The original reference on the DCT is:

> N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete Cosine Transform," *IEEE Trans. Computers*, **C-23**, 90 (1974).
>
> See also, N. Ahmed, "How I Came Up With the Discrete Cosine Transform," *Dig. Sig. Proc.* **1**, 4 (1991).
>
> For a more complete set of references and reviews of data compression methods, see the articles in the zip-file of project-7.

DFT

We recall the N-point DFT of an N-point signal,

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

or, component-wise,

$$\text{(DFT):} \qquad X_k = \sum_{n=0}^{N-1} x_n e^{-2\pi jkn/N}, \qquad k = 0, 1, \ldots, N-1$$

$$\text{(IDFT):} \qquad x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{2\pi jkn/N}, \quad n = 0, 1, \ldots, N-1$$

and in matrix form, define the $N \times N$ symmetric DFT matrix,

$$A_{kn} = e^{-2\pi jkn/N}, \qquad \begin{array}{l} k = 0, 1, \ldots, N-1 \\ n = 0, 1, \ldots, n-1 \end{array}$$

satisfying the unitarity and inversion properties,

$$\frac{1}{N} A A^* = I_N \quad \Rightarrow \quad A^{-1} = \frac{1}{N} A^*$$

so that the forward and inverse DFTs can be expressed compactly as,

$$\boxed{\mathbf{X} = A\mathbf{x} \quad \Rightarrow \quad \mathbf{x} = \frac{1}{N} A^* \mathbf{X}}$$

The DCT is a similar orthogonal transform that can be viewed as a real-valued version of the DFT.

Data Compression with DCT

The DCT takes as input a length-N real-valued signal vector, $\mathbf{x}$, and produces a length-N real-valued vector of DCT coefficients, $\mathbf{C}$. The inverse DCT reverses the process, recovering the original signal vector,

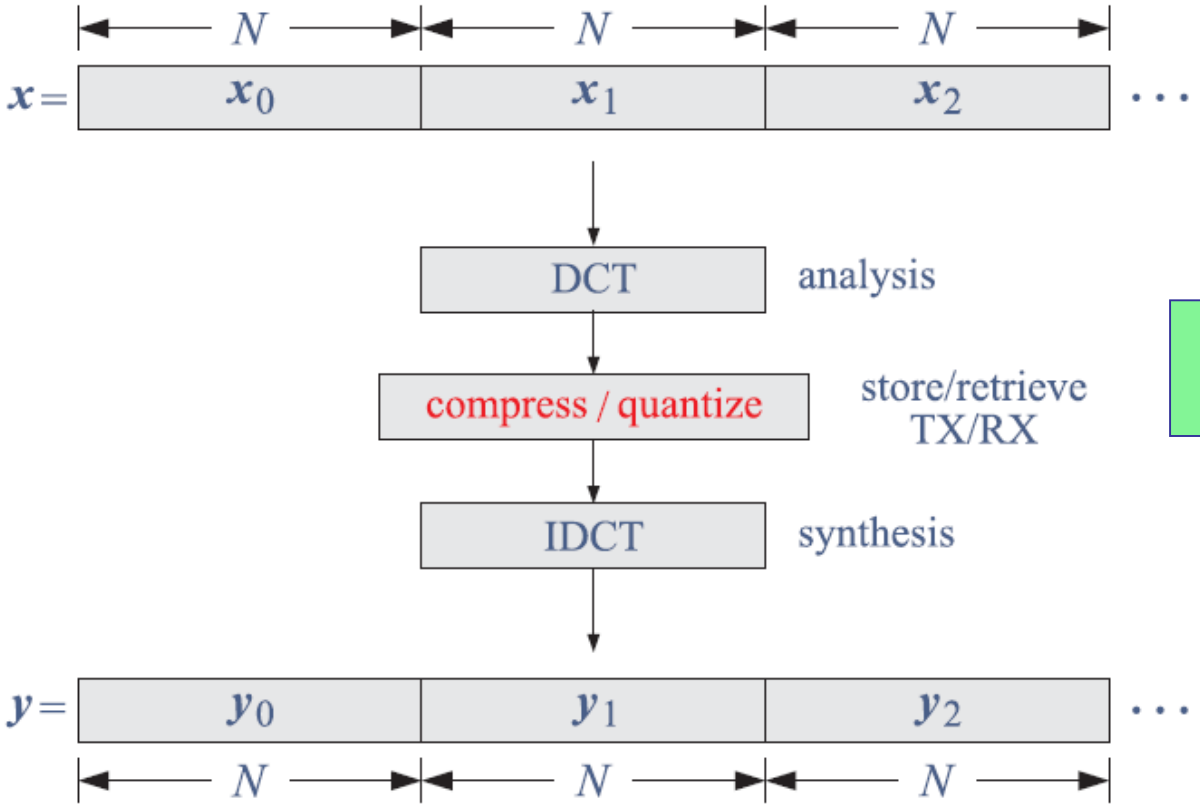
$$\mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \quad \xrightarrow{\text{DCT}} \quad \mathbf{C} = \begin{bmatrix} C_0 \\ C_1 \\ \vdots \\ C_{N-1} \end{bmatrix} \quad \xrightarrow{\text{IDCT}} \quad \mathbf{x} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{N-1} \end{bmatrix} \qquad (1)$$

In a typical DCT-based data compression system, a large fraction (e.g., 80-90%) of the DCT coefficients C_k are dropped, retaining only a few of the most significant ones, which are then quantized and stored or transmitted.

The signal recovered from the few retained DCT coefficients—while not identically equal to the original one—is close enough to the original to be perceptually indistinguishable from it.

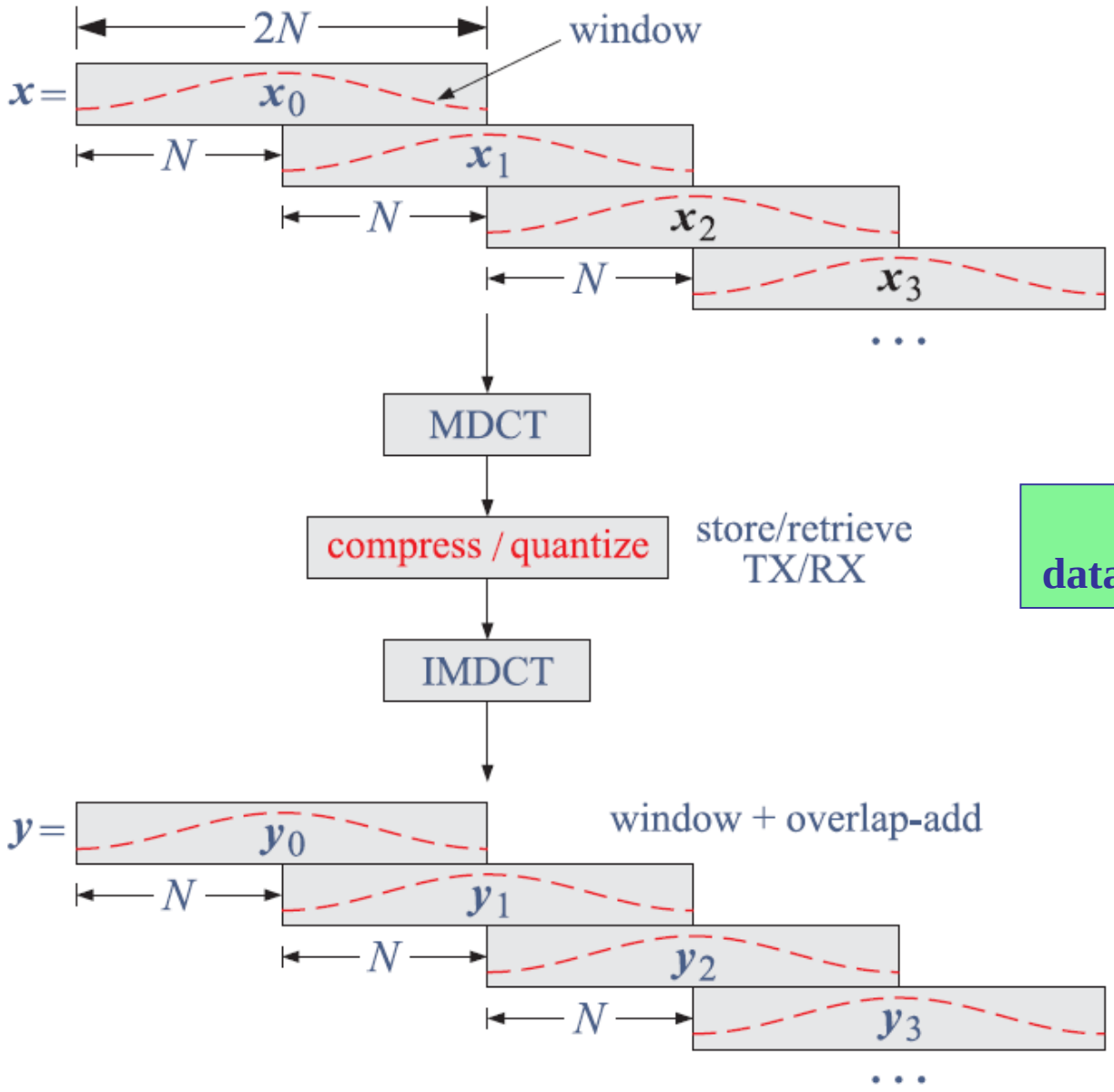
Such process is referred to as *lossy compression* since the recovered signal is slightly different from the original one as, for example, in MP3 audio or JPEG images. In audio and image applications, the DCT coefficient quantization process takes into account the psychophysical properties of the hearing or visual system, and is beyond the scope of project-7.

A simplified DCT compression system is shown below in which a long input signal is divided into contiguous length-N blocks or frames, the N-point DCT of each frame is computed and compressed, then, the corresponding inverse DCTs of the frames are computed and the recovered signal blocks are pieced together to form the output signal.



simplified DCT compression system

To avoid possible artifacts that may be introduced by the blocking process, a more refined approach divides the input signal into overlapping blocks.



MDCT / TDAC data compression system

Such system is based on the modidied DCT (MDCT) and divides the input into blocks that are 50% overlapping, then windows each block, and calculates its MDCT, compresses the MDCT coefficients, and takes the inverse MDCT, windows the resulting blocks again, and finally overlaps and adds the results.

The window functions must be chosen properly (i.e., satisfying the so-called Princen-Bradley conditions) so that the overlap/add operation correctly implements the time-domain aliasing cancellation (TDAC) property that allows the faithful reconstruction of the input signal.

MDCT-based compression systems are used in current audio compression formats, such as MP3, AAC, WMA, Vorbis, and others.

DCT

There exist eight versions of the DCT, but the type-2 is the most widely used and is the default version in MATLAB's **dct** function and can be implemented efficiently using an FFT. The type-2 DCT is defined as follows,

$$C_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad k = 0, 1, \ldots, N-1$$

and its inverse,

$$x_n = \frac{1}{N} C_0 + \frac{2}{N} \sum_{k=1}^{N-1} C_k \cos\left(\frac{\pi k}{N}\left(n+\tfrac{1}{2}\right)\right), \quad n = 0, 1, \ldots, N-1$$

The DCT pair satisfies the following Parseval-like identity,

$$\sum_{n=0}^{N-1} x_n^2 = \frac{1}{N}\left[C_0^2 + 2\sum_{k=1}^{N-1} C_k^2\right] \quad \text{(Parseval)}$$

DCT IDCT

In MATLAB, and other literature, a normalized version of the DCT coefficients is used, defined as follows, where δ_k denotes the Kronecker delta,

$$D_k = \frac{1}{s_k} C_k\,, \quad \text{where} \quad s_k = \sqrt{\frac{N}{2}\left(\delta_k + 1\right)}\,, \quad k = 0, 1, \ldots, N-1 \quad (2)$$

so that,

$$s_0 = \sqrt{N}\,, \qquad s_k = \sqrt{\frac{N}{2}}\,, \quad k = 1, 2, \ldots, N-1$$

$$D_0 = \sqrt{\frac{1}{N}}\, C_0\,, \quad D_k = \sqrt{\frac{2}{N}}\, C_k\,, \quad k = 1, 2, \ldots, N-1$$

With this definition, the normalized DCT and its inverse read as follows,

$$\boxed{D_k = \frac{1}{s_k} \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi k}{N}\left(n + \tfrac{1}{2}\right)\right)\,, \quad k = 0, 1, \ldots, N-1} \quad \text{(DCT)}$$

$$\boxed{x_n = \sum_{k=0}^{N-1} \frac{1}{s_k} D_k \cos\left(\frac{\pi k}{N}\left(n + \tfrac{1}{2}\right)\right)\,, \quad n = 0, 1, \ldots, N-1} \quad \text{(IDCT)}$$

$$\boxed{\sum_{n=0}^{N-1} x_n^2 = \sum_{k=0}^{N-1} D_k^2} \quad \text{(Parseval)}$$

The above relationships can be understood more simply by expressing them in matrix form. Let us define the $N \times N$ matrix of DCT coefficients by its kn matrix element,

$$B_{kn} = \cos\left(\frac{\pi k}{N}\left(n + \tfrac{1}{2}\right)\right), \qquad \begin{matrix} k = 0, 1, \ldots, N-1 \\ n = 0, 1, \ldots, N-1 \end{matrix}$$

Then, the forward DCT and its inverse can be written in matrix form,

$$\mathbf{C} = B\mathbf{x} \quad \Rightarrow \quad \mathbf{x} = B^{-1}\mathbf{C}$$

Define the diagonal matrix of the scale factors s_k,

$$S = \mathrm{diag}\big([s_0, s_1, \ldots, s_{N-1}]\big) = \begin{bmatrix} s_0 & 0 & 0 & \cdots & 0 \\ 0 & s_1 & 0 & \cdots & 0 \\ 0 & 0 & s_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_{N-1} \end{bmatrix}$$

Then, it can be shown that the matrix B satisfies the following orthogonality-like property from which its inverse can be determined,

$$B\,B^T = S^2 = \mathrm{diag}\big([s_0^2, s_1^2, \ldots, s_{N-1}^2]\big)$$

$$B^{-1} = B^T S^{-2} = B^T\,\mathrm{diag}\big([s_0^{-2}, s_1^{-2}, \ldots, s_{N-1}^{-2}]\big)$$

Thus, the forward and inverse DCTs can be written in the following matrix forms, for the un-normalized and normalized versions, since, $\mathbf{D} = S^{-1}\mathbf{C}$,

$$\boxed{\begin{aligned}\mathbf{C} &= B\mathbf{x}\\ \mathbf{x} &= B^T S^{-2}\mathbf{C}\end{aligned}} \quad \Rightarrow \quad \boxed{\begin{aligned}\mathbf{D} &= S^{-1}B\mathbf{x} = A\mathbf{x}\\ \mathbf{x} &= B^T S^{-1}\mathbf{D} = A^T\mathbf{D}\end{aligned}}$$

DCT
IDCT

where we defined the rescaled DCT matrix,

$$\boxed{A = S^{-1}B} \quad \text{(DCT matrix)}$$

which is *orthogonal*, that is,

$$AA^T = I_N \quad \Rightarrow \quad A^{-1} = A^T$$

Indeed, we have from the orthogonality of B,

$$AA^T = S^{-1}BB^T S^{-1} = S^{-1}S^2 S^{-1} = I_N$$

The normalized DCT matrix A can be computed for any N by the following one-line anonymous MATLAB function:

```
A = @(N) sqrt(2/N) * ...
         [ones(1,N)/sqrt(2); cos(pi*(1:N-1)'*((0:N-1)+1/2)/N)];
```

Although the above matrix formulation is very efficient in MATLAB for moderate sizes (i.e., $N \leq 1024$), there is an even faster implementation based on computing the N-point DCT vector $\mathbf{C}$ from a related $2N$-point FFT.

The computational steps are summarized as follows. Define the extended signal and DCT column vectors of length-$2N$ obtained by appending to $\mathbf{x}$ its reversed version, and similarly, appending its negative reversed version to the DCT,

$$\mathbf{y} = [x_0, x_1, \ldots, x_{N-1}, x_{N-1}, \ldots, x_1, x_0]^T$$

$$\mathbf{C}^{\text{ext}} = [C_0, C_1, \ldots, C_{N-2}, C_{N-1}, 0, -C_{N-1}, -C_{N-2}, \ldots, -C_1]^T$$

Then, it can be shown that the $2N$-point DFT of $\mathbf{y}$ is related to the extended DCT vector by,

$$Y_k = 2e^{j\pi k/2N} C_k^{\text{ext}}, \quad k = 0, 1, \ldots, 2N-1, \quad \text{or,}$$

$$C_k^{\text{ext}} = \frac{1}{2} e^{-j\pi k/2N} Y_k, \quad k = 0, 1, \ldots, 2N-1$$

FFT-based fast DCT

where we have the $2N$-point DFT pair,

$$Y_k = \sum_{n=0}^{2N-1} y_n e^{-2\pi jk/2N}, \quad k = 0, 1, \ldots, 2N-1$$

$$y_n = \frac{1}{2N} \sum_{n=0}^{2N-1} Y_k e^{2\pi jk/2N}, \quad n = 0, 1, \ldots, 2N-1$$

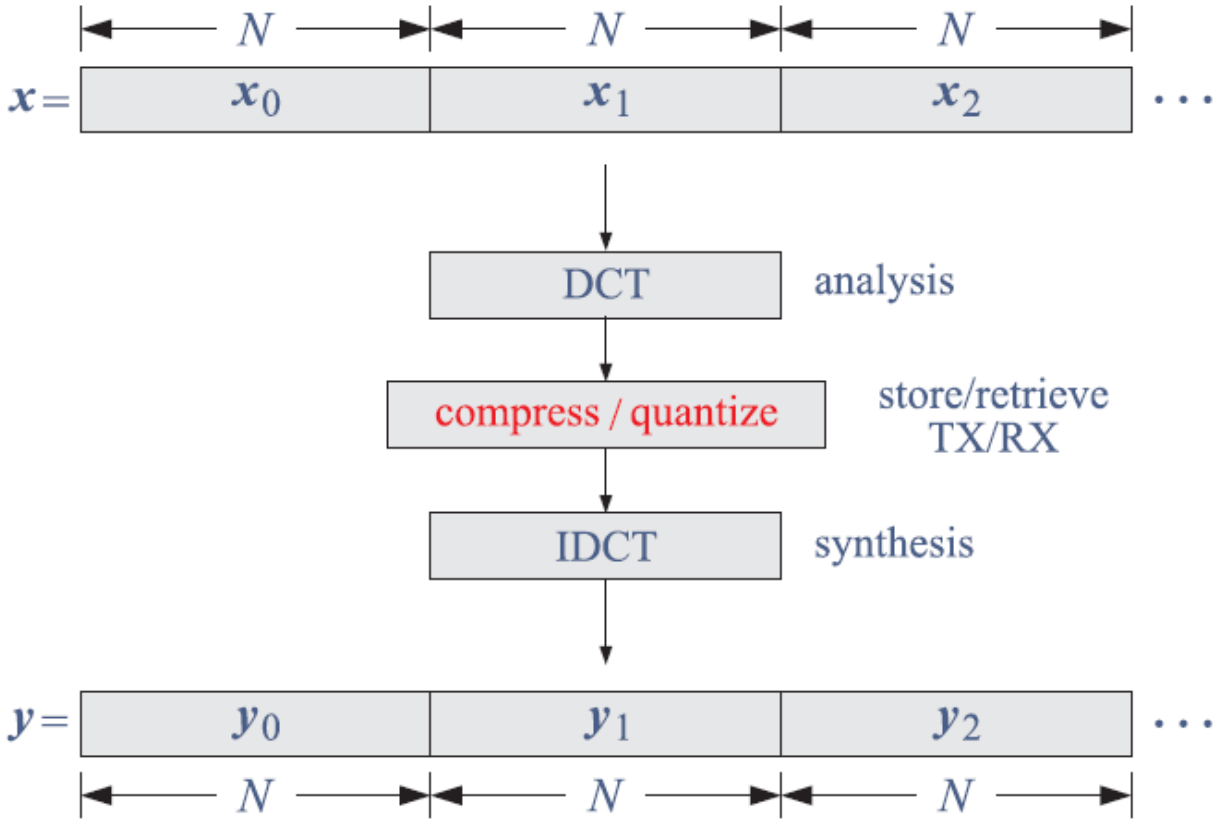
This leads to the following efficient computational algorithm for the DCT:

(a) extend the data vector **x** to **y** as above
(b) compute the FFT of **y**, that is, Y_k, $k = 0, 1, \ldots, 2N-1$
(c) construct the extended C_k^{ext} vector as above, and
(d) retain the first N elements, $C_k = C_k^{\text{ext}}$, $k = 0, 1, \ldots, N-1$
(e) renormalize the result, $D_k = C_k / s_k$, $k = 0, 1, \ldots, N-1$

For the inverse DCT, the above steps are entirely reversible, that is, starting with D_k, $k = 0, 1, \ldots, N-1$, first undo the normalization scale factors, $C_k = s_k D_k$, then form the length-$2N$ extended vector C_k^{ext}, and evaluate the DFT values Y_k, then, perform an inverse $2N$-point FFT to recover **y** and retain its the first N elements, $x_n = y_n$, $n = 0, 1, \ldots, N-1$.

Simplified DCT Compression System

To clarify the operations in the simplified DCT compression system shown below, and the possible methods of compressing the DCT coefficients, we discuss a small example, implemented in MATLAB.



Define a length-40 signal and divided it up in 4 frames of length $N = 10$, with the help of the **buffer** function, and compute the DCT of all frames,

```
L = 40; t = (0:L-1)/L;     %
x = sin(10*t.^2) + 2*t;    % length L=40

N = 10;                    % frame length
X = buffer(x,N);           % 10x4 matrix of frames
D = dct(X);                % 10x4 matrix of DCT coefficients
```

X					D			
0	1.0851	1.5985	0.8883		1.2623	4.9797	2.6349	6.5811
0.0562	1.2362	1.4259	1.2766		-0.9433	-0.7086	1.2451	-1.2616
0.1250	1.3833	1.2163	1.7165		0.1282	-0.3166	0.4891	-1.4116
0.2062	1.5205	0.9861	2.1495	DCT	-0.0959	-0.0124	-0.0916	0.1412
0.2998	1.6408	0.7575	2.5086	==>	0.0286	-0.0736	0.1147	-0.2160
0.4056	1.7365	0.5577	2.7305		-0.0310	-0.0039	-0.0281	0.0275
0.5231	1.7996	0.4164	2.7699		0.0108	-0.0280	0.0432	-0.0768
0.6515	1.8224	0.3622	2.6134		-0.0125	-0.0016	-0.0111	0.0097
0.7894	1.7986	0.4175	2.2892		0.0041	-0.0106	0.0163	-0.0286
0.9349	1.7241	0.5943	1.8686		-0.0035	-0.0004	-0.0031	0.0026

We consider two methods of compression. In method-1, we pick a compression factor, $r < 1$, which defines the number of DCT coefficients to be kept from each length-N frame,

$$N_r = \text{round}(rN)\,, \qquad r_{\text{actual}} = \frac{N_r}{N} \tag{3}$$

then, sort the absolute values of the DCT coefficients in each column in descending order, and keep the highest N_r of them, setting the rest of the coefficients to zero. Because of the rounding process, the actual realized compression factor, r_{actual}, may be slightly different from r. The sorting and construction of the new DCT coefficient matrix could be done as follows,

```
M = size(D,2);                          % no. of frames
Nr = round(r*N);                        % no. of kept coefficients per frame

[~, Ir] = sort(abs(D),'descend');       % sort column-wise, descending order
Ir = Ir(1:Nr,:);                        % Ir = Nr x M matrix of sorting indices

C = zeros(size(D));                     % initialize kept DCT coefficients

for m = 1:M                             % rebuild DCT from the kept coefficients
    Dr(:,m) = D(Ir(:,m),m);             % Dr = Nr x M matrix, sorted coefficients
    C(Ir(:,m),m) = Dr(:,m);             % C = new DCT coefficients
end
```

In a storage/retrieval or transmitting/receiving system, one would store or transmit both the index and coefficient matrices, I_r, D_r, in order to be able to re-position the kept coefficients in their original order, therefore, the compression factor would be, $2r$, in this case. For example, with, $r = 0.4$, we have the sorted coefficients and sorting indices,

Ir				Dr			
1	1	1	1	1.2623	4.9797	2.6349	6.5811
2	2	2	3	-0.9433	-0.7086	1.2451	-1.4116
3	3	3	2	0.1282	-0.3166	0.4891	-1.2616
4	5	5	5	-0.0959	-0.0736	0.1147	-0.2160

X					D			
0	1.0851	1.5985	0.8883		1.2623	4.9797	2.6349	6.5811
0.0562	1.2362	1.4259	1.2766		-0.9433	-0.7086	1.2451	-1.2616
0.1250	1.3833	1.2163	1.7165		0.1282	-0.3166	0.4891	-1.4116
0.2062	1.5205	0.9861	2.1495	DCT	-0.0959	-0.0124	-0.0916	0.1412
0.2998	1.6408	0.7575	2.5086	==>	0.0286	-0.0736	0.1147	-0.2160
0.4056	1.7365	0.5577	2.7305		-0.0310	-0.0039	-0.0281	0.0275
0.5231	1.7996	0.4164	2.7699		0.0108	-0.0280	0.0432	-0.0768
0.6515	1.8224	0.3622	2.6134		-0.0125	-0.0016	-0.0111	0.0097
0.7894	1.7986	0.4175	2.2892		0.0041	-0.0106	0.0163	-0.0286
0.9349	1.7241	0.5943	1.8686		-0.0035	-0.0004	-0.0031	0.0026

In method-2, we pick a threshold factor, $r_{\text{thr}} < 1$, which defines a threshold value D_{thr} for the DCT coefficients below which the coefficients are discarded, and construct a new DCT matrix that only has coefficients such that, $|D| \geq D_{\text{thr}}$, where,

$$D_{\text{thr}} = r_{\text{thr}} \cdot |D|_{\max} \tag{4}$$

This method can be implemented by the example code:

```
Dthr = r_thr * max(max(abs(D)));    % DCT threshold

I = find(abs(D) < Dthr);            % indices of DCT coeffs to be zeroed

C = D;                              % initialize with C = D

C(I) = 0;                           % discard coefficients below Dthr

ra = 1-length(I)/(N*M);             % realized compression factor

% ra = length(find(C)) / prod(size(C));   % alternative calculation
```

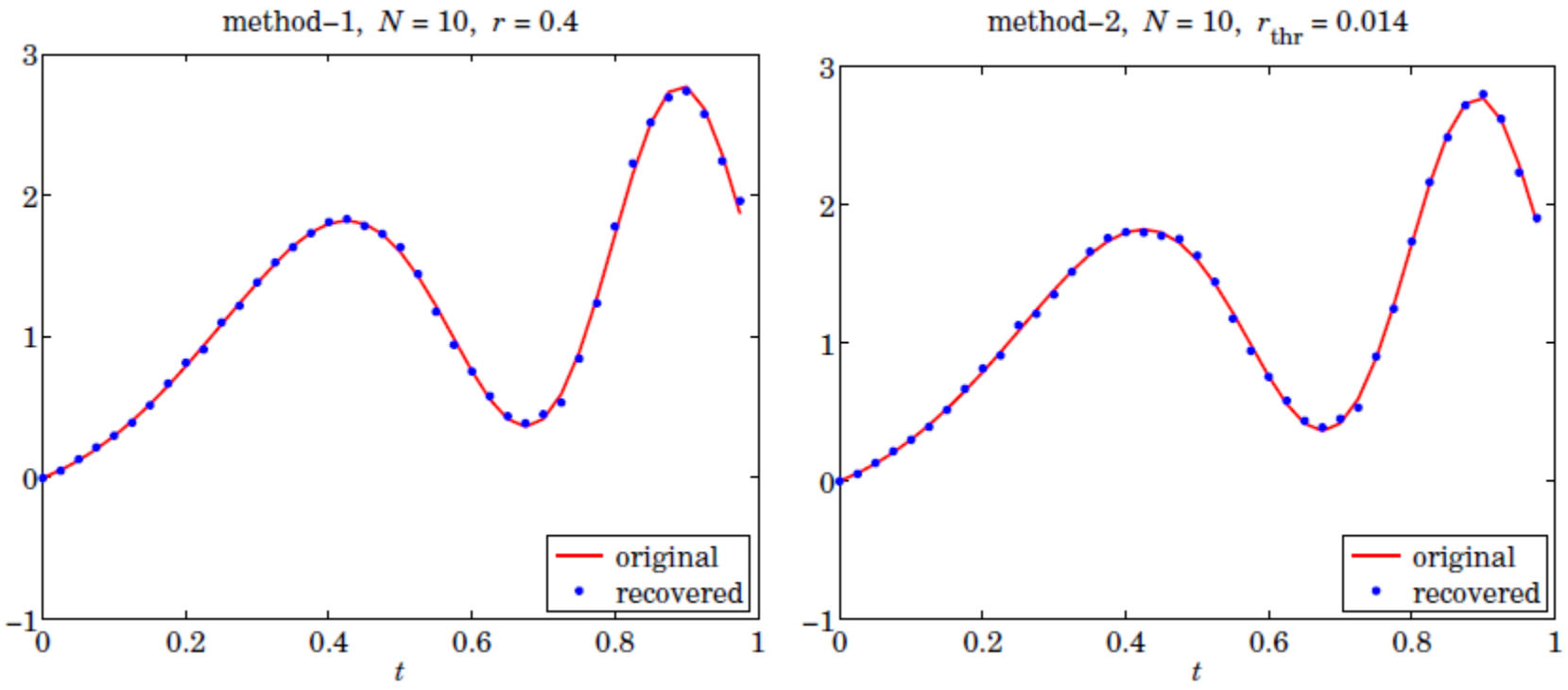
Applying the two methods to the DCT matrix D of the above example, we obtain the following new DCT matrices C, where we used, $r = 0.4$, for method-1, and, $r_{\text{thr}} = 0.014$, for method-2 (chosen such that both methods achieve the same actual compression ratio, $r_{\text{actual}} = 0.4$.)

method-1 C				method-2 C			
1.2623	4.9797	2.6349	6.5811	1.2623	4.9797	2.6349	6.5811
-0.9433	-0.7086	1.2451	-1.2616	-0.9433	-0.7086	1.2451	-1.2616
0.1282	-0.3166	0.4891	-1.4116	0.1282	-0.3166	0.4891	-1.4116
-0.0959	0	0	0	-0.0959	0	0	0.1412
0	-0.0736	0.1147	-0.2160	0	0	0.1147	-0.2160
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

We note that for method-1 there are, $N_r = rN = 0.4 \cdot 10 = 4$, coefficients per frame, but that number is variable in method-2 for which the computed threshold was, $D_{\text{thr}} = r_{\text{thr}}|D|_{\text{max}} = 0.014 \cdot 6.5811 = 0.0921$, with all coefficients with magnitudes less than that set to zero.

The actual compression factor, representing the fraction non-zero DCT coefficients, was the same in the two cases. The final reconstructed output is finally obtained by performing an inverse DCT on C and concatenating the resulting frames, with the results plotted below.

```
Y = idct(C);     % IDCT of all frames
y = Y(:);        % concatenate frames
y = y(1:L);      % make x,y lengths equal (in case buffer had extended X)

plot(t,x,'r-', t,y,'b.');      % plot original and compressed signals
```



The approach also works with images, using a 2D-DCT (which is equivalent to taking the 1D-DCT of each column followed by the 1D-DCT of each row of the image). For example, try the following MATLAB code based on the documentation of the **dct2** function,

```
X = imread('cameraman.tif');          % read image, 256x256 matrix

D = dct2(X);                          % compute its 2D-DCT

Dmax = max(max(abs(D)));              % Dmax = 30393.4687
Dth = 10;                             % select a threshold
rth = Dth/Dmax;                       % with threshold factor,
                                      % rth = 3.2902e-04
C = D;
C(abs(D )< Dth) = 0;                  % compressed DCT

ra = length(find(C))/prod(size(C))    % actual compression ratio,
                                      % ra = 26617/65536 = 0.4061

Y = idct2(C);                         % inverse 2D-DCT

figure; imshowpair(X,Y,'montage')     % display images side by side
```

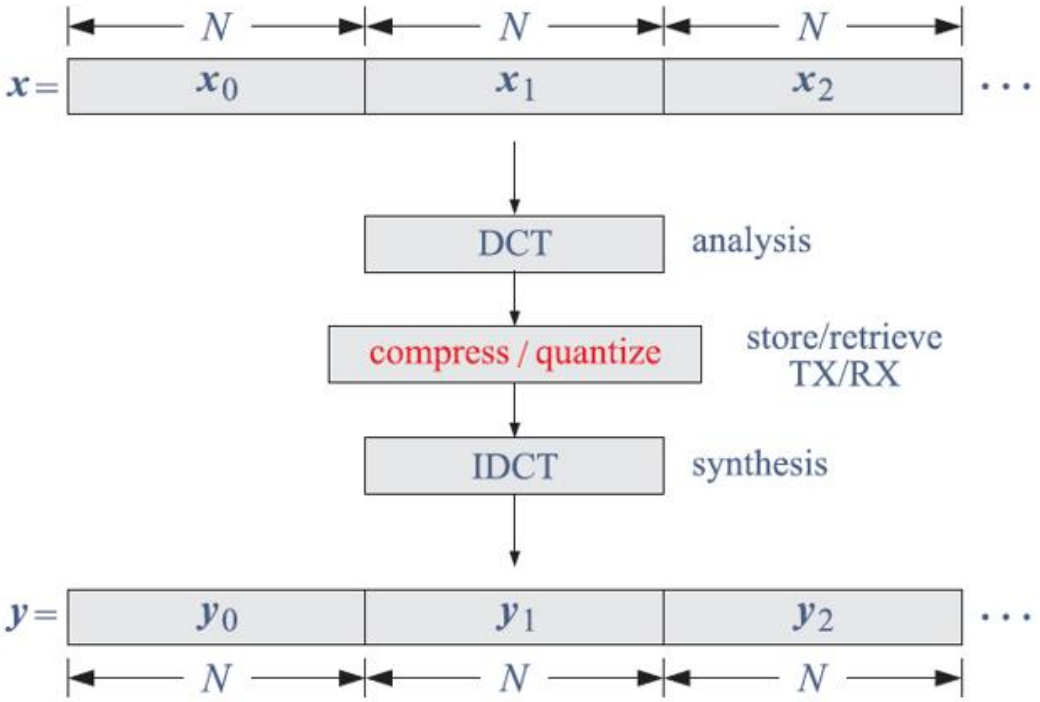
The JPEG image compression standard uses similar DCT operations, but applied to 8x8 sub-blocks of the image, and employing a standardized quantization scheme. JPEG has been replaced by JPEG2000, which uses wavelet compression.

Summary – Simplified DCT compression system

In project-7, you will need to write a MATLAB function, **dctcompr**, that implements the simplified DCT compression scheme that incorporates the two compression methods outlined above.

```
[y,ra] = dctcompr(x,N,r,method);

% x = signal to be compressed
% N = frame length
% r = compression ratio or threshold factor (r<1)
% method = 1,2, compression method, default is 1
%
% y = compressed signal, same length as x
% ra = actual compression ratio achieved
```

In this function, you may use either the dct/idct built-in functions, or your own FFT-based fast versions. The function should put together the following operations,



$X = \mathbf{buffer}(\mathbf{x}, N)$

$D = \mathbf{dct}(X)$

$C = f(D, r) =$ method 1 or 2

$Y = \mathbf{idct}(C)$

$\mathbf{y} =$ concatenate Y columns

To clarify the overall sequence of computational steps, we list below the explicit steps for methods 1 & 2 for the previous example

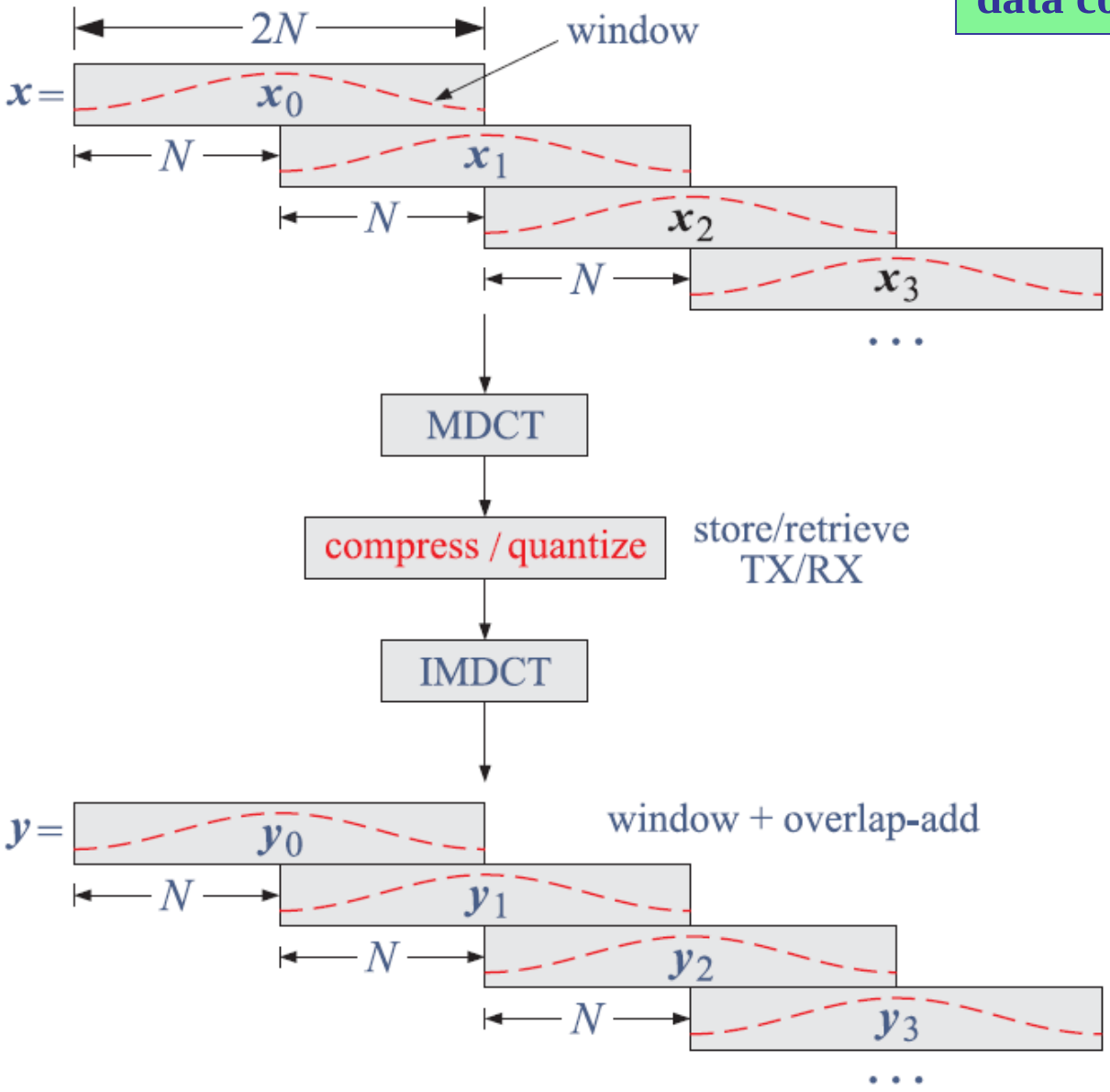
method 1

```
L = 40;
t = (0:L-1)/L;
x = sin(10*t.^2) + 2*t;        % sampled signal

N=10; r=0.4;                   % pick N and r - method 1

X = buffer(x,N);               % NxM matrix of frames

D = dct(X);                    % DCT of each column of X

M = size(D,2);                 % here, M=4

Nr = round(r*N);               % number of kept coefficients per frame
ra = Nr/N;                     % actual compression ratio

[~, Ir] = sort(abs(D), 'descend');   % sort D in descending order
Ir = Ir(1:Nr,:);                     % keep Nr coefficients per frame

C = zeros(size(D));            % kept DCT coefficients

for m = 1:M                    % rebuild DCT from the kept coefficients
    Dr(:,m) = D(Ir(:,m),m);    % Nr x M matrix of sorted coefficients
    C(Ir(:,m),m) = Dr(:,m);    % C = new DCT coefficients
end

Y = idct(C);     % invert compressed DCT
y = Y(:);        % concatenate columns
y = y(1:L);      % compressed signal
```

method 2

```
L = 40;
t = (0:L-1)/L;
x = sin(10*t.^2) + 2*t;        % sampled signal

N=10; r=0.014;                 % pick N and r - method 2

X = buffer(x,N);               % NxM matrix of frames

D = dct(X);                    % DCT of each column of X

M = size(D,2);                 % here, M=4

Dthr = r * max(max(abs(D)));       % DCT threshold

I = find(abs(D) < Dthr);           % indices of DCT coeffs to be zeroed

C = D;                             % initialize with C = D

C(I) = 0;                          % discard coefficients below Dthr

ra = length(find(C)) / prod(size(C));   % realized compression factor

Y = idct(C);     % invert compressed DCT
y = Y(:);        % concatenate columns
y = y(1:L);      % compressed signal
```

MDCT / TDAC
data compression system



MDCT / TDAC data compression system

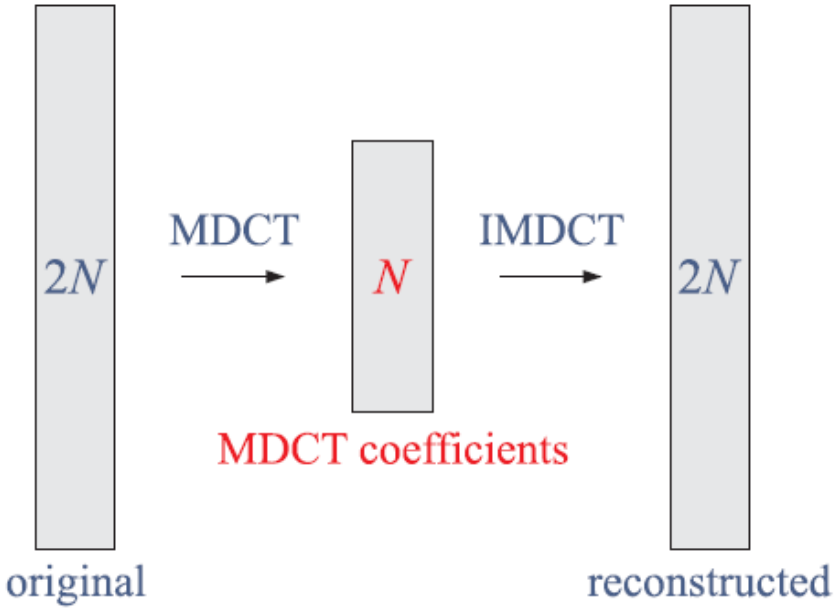
Such system is based on the modidied DCT (MDCT) and divides the input into blocks that are 50% overlapping, then windows each block, and calculates its MDCT, compresses the MDCT coefficients, and takes the inverse MDCT, windows the resulting blocks again, and finally overlaps and adds the results.

The window functions must be chosen properly (i.e., satisfying the so-called Princen-Bradley conditions) so that the overlap/add operation correctly implements the time-domain aliasing cancellation (TDAC) property that allows the faithful reconstruction of the input signal.

MDCT-based compression systems are used in current audio compression formats, such as MP3, AAC, WMA, Vorbis, and others.

MDCT

The modified DCT (MDCT) is not quite an orthogonal or invertible transform as it transforms a length-$2N$ data block into a length-N vector of MDCT coefficients, while the inverse MDCT (IMDCT) transforms the N MDCT coefficients back to a length-$2N$ data block, which is not quite equal to the original block.



However, because the MDCT is used in blocks that are 50% overlapping, the reconstruction error introduced in one block is cancelled by the error introduced by the next block—a property referred to as time-domain aliasing cancellation (TDAC)—so that the original signal is reconstructed correctly.

The N-point MDCT D_k of a $2N$-point signal x_n, and the corresponding $2N$-point inverse MDCT y_n are defined as follows,

$$D_k = \sum_{n=0}^{2N-1} x_n \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right), \quad k = 0, 1, \ldots, N-1$$

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} D_k \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right), \quad n = 0, 1, \ldots, 2N-1$$

The precise relationship of the reconstructed signal to the original one is given below, being expressed more simply by splitting the input and reconstructed length-$2N$ blocks into their upper and lower length-N sub-blocks,

$$\mathbf{x} = \begin{bmatrix}\mathbf{a}\\ \mathbf{b}\end{bmatrix} \quad \xrightarrow{\text{MDCT}} \quad \mathbf{D} \quad \xrightarrow{\text{IMDCT}} \quad \mathbf{y} = \begin{bmatrix}\frac{1}{2}(\mathbf{a}-\mathbf{a}_R)\\ \frac{1}{2}(\mathbf{b}+\mathbf{b}_R)\end{bmatrix}$$

where $\mathbf{a}_R, \mathbf{b}_R$ denote the reversed vectors, e.g., for $N = 4$,

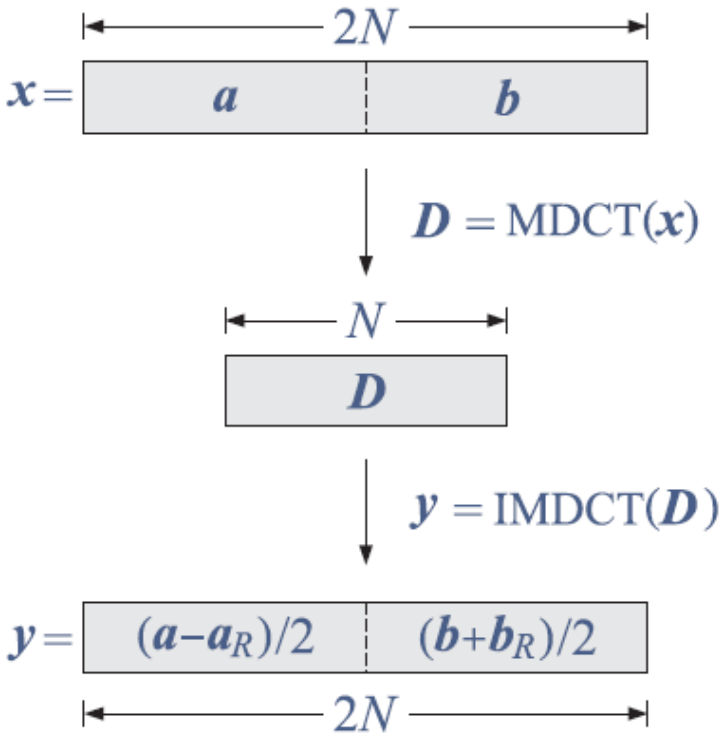
$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} \quad \Rightarrow \quad \mathbf{a}_R = \begin{bmatrix} a_3 \\ a_2 \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix} = J\mathbf{a}$$

where, defining the $N \times N$ reversing matrix J having ones along its anti-diagonal line, one may think of $\mathbf{a}_R$ as the result of the matrix operation, $\mathbf{a}_R = J\mathbf{a}$. Thus, introducing also the $N \times N$ identity matrix I, we have,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \quad \overset{\text{MDCT}}{\longrightarrow} \quad \mathbf{D} \quad \overset{\text{IMDCT}}{\longrightarrow} \quad \mathbf{y} = \begin{bmatrix} \frac{1}{2}(I - J)\mathbf{a} \\ \frac{1}{2}(I + J)\mathbf{b} \end{bmatrix}, \quad \text{or,}$$

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \quad \overset{\text{MDCT}}{\longrightarrow} \quad \mathbf{D} \quad \overset{\text{IMDCT}}{\longrightarrow} \quad \mathbf{y} = \begin{bmatrix} \frac{1}{2}(I - J) & 0 \\ 0 & \frac{1}{2}(I + J) \end{bmatrix} \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix}$$

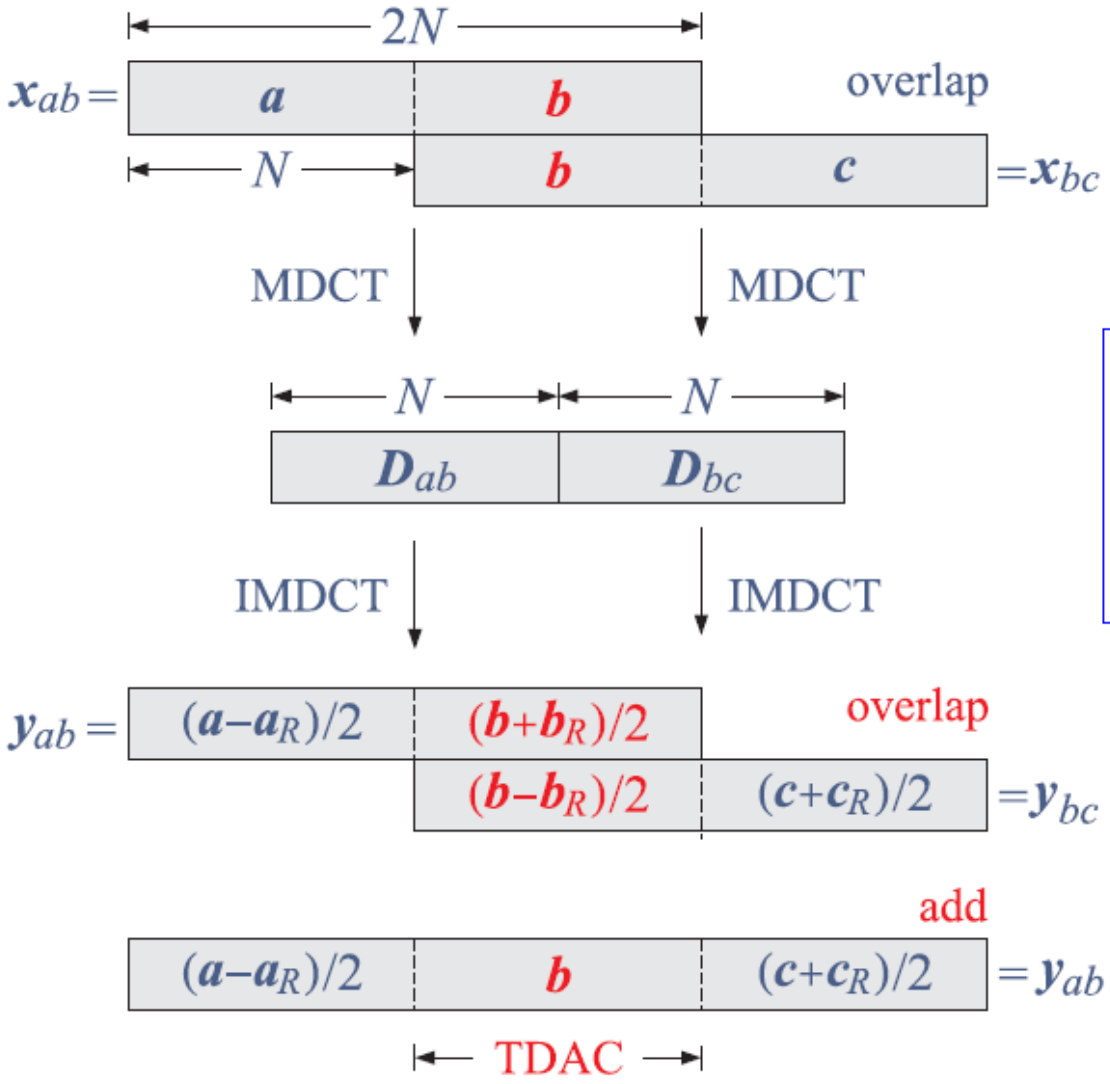
Pictorially, we have for one block,



Applying this property to two blocks that are 50% overlapping, we observe that if the overlapping reconstructed blocks are added, the second-half of the first block is corrected by the first-half of the second block, recovering the overlapping portion of the original blocks, that is,

$$\frac{1}{2}(\mathbf{b} + \mathbf{b}_R) + \frac{1}{2}(\mathbf{b} - \mathbf{b}_R) = \mathbf{b}$$

This is precisely the time-domain aliasing cancellation (TDAC) property.



$$\mathbf{D}_{ab} = \mathrm{MDCT}(\mathbf{x}_{ab})$$
$$\mathbf{D}_{bc} = \mathrm{MDCT}(\mathbf{x}_{bc})$$
$$\mathbf{y}_{ab} = \mathrm{IMDCT}(\mathbf{D}_{ab})$$
$$\mathbf{y}_{bc} = \mathrm{IMDCT}(\mathbf{D}_{bc})$$

For a long signal that is split into several such 50% overlapping blocks, as shown in Fig. 2, the entire signal is recovered correctly, with the exception of the first and last length-N portions that are not overlapping—these can be fixed by padding N zeros at the beginning and end of the original signal prior to MDCT processing. Alternatively, the last and first N outputs can be replaced by NaN's or by the original input samples.

The fundamental TDAC result can be derived by considering the matrix formulation of the MDCT/IMDCT. Let us define the $N \times 2N$ MDCT transformation matrix by its kn matrix element,

$$F_{kn} = \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right), \qquad \begin{matrix} k = 0, 1, \ldots, N-1 \\ n = 0, 1, \ldots, 2N-1 \end{matrix}$$

Then, the MDCT and IMDCT transformations are in matrix form,

$$\boxed{\begin{aligned} \mathbf{D} &= \mathbf{mdct}(\mathbf{x}) = F\mathbf{x} \\ \mathbf{y} &= \mathbf{imdct}(\mathbf{D}) = \frac{1}{N}F^T\mathbf{D} \end{aligned}} \quad \Rightarrow \quad \mathbf{y} = \frac{1}{N}F^T F\mathbf{x}$$

these matrix forms may be used in project-7

We may split F into its two $N \times N$ submatrices, $F = [A, B]$, defined by their matrix elements,

$$A_{kn} = F_{kn} = \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{1}{2}N\right)\right)$$
$$B_{kn} = F_{k,n+N} = \cos\left(\frac{\pi}{N}\left(k+\tfrac{1}{2}\right)\left(n+\tfrac{1}{2}+\tfrac{3}{2}N\right)\right)$$

with $k = 0, 1, \ldots, N-1$, $n = 0, 1, \ldots, N-1$

Then, it is left as part of project-7 to show that the submatrices A, B satisfy the relationships,

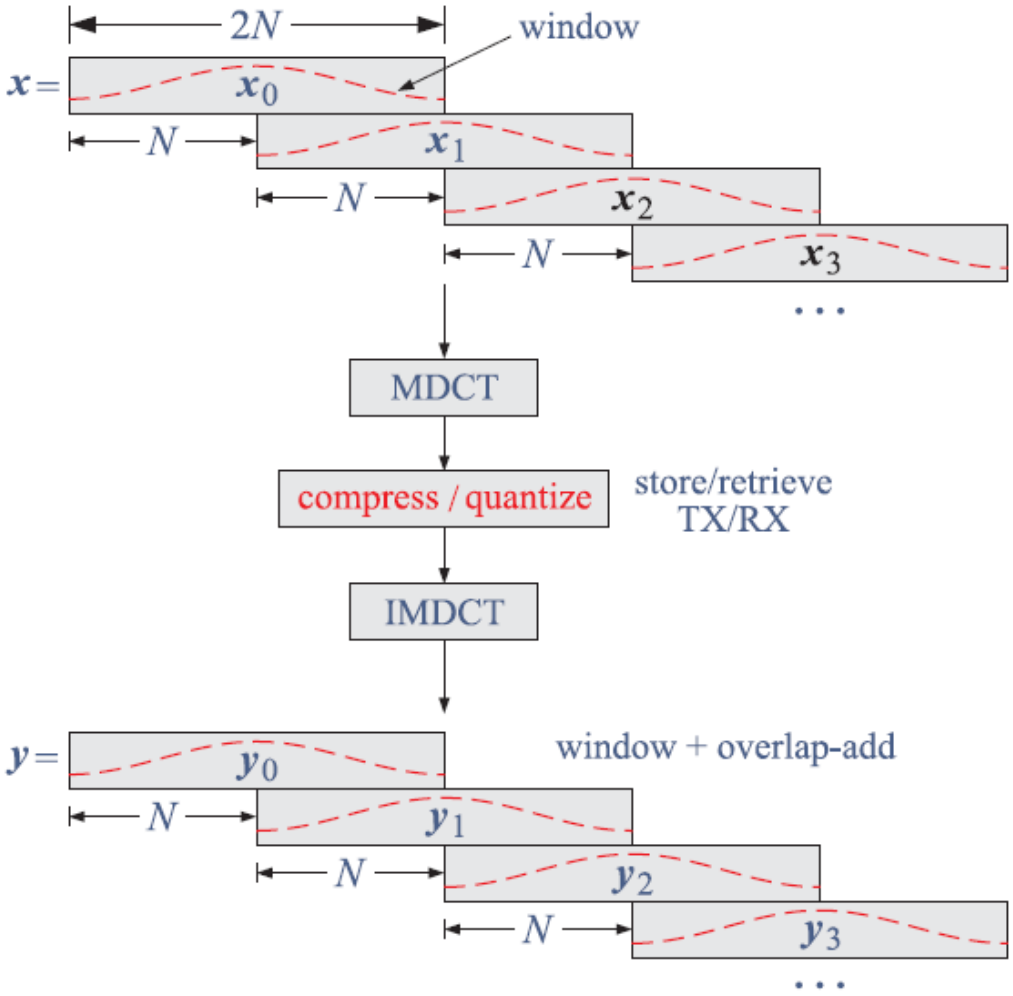
$$\boxed{\begin{aligned} \frac{1}{N} A^T A &= \frac{1}{2}(I - J) \\ \frac{1}{N} B^T B &= \frac{1}{2}(I + J) \\ A^T B &= B^T A = 0 \end{aligned}}$$

And, these imply the IMDCT property because,

$$\frac{1}{N} F^T F = \frac{1}{N} \begin{bmatrix} A^T \\ B^T \end{bmatrix} [A, B] = \frac{1}{N} \left[\begin{array}{c|c} A^T A & A^T B \\ \hline B^T A & B^T B \end{array}\right] = \left[\begin{array}{c|c} \frac{1}{2}(I - J) & 0 \\ \hline 0 & \frac{1}{2}(I + J) \end{array}\right]$$

Princen-Bradley Windows

As depicted below, each frame of length-$2N$ of the input and reconstructed output is windowed with a length-$2N$ window in order to reduce the blocking effects near the endpoints of the frames.



Such windows are chosen to be symmetric about their middle with their second length-N half being the reversed version of their first half,

$$\hat{\mathbf{w}} = [\underbrace{w_0, w_1, \ldots, w_{N-1}}_{\mathbf{w}}, \underbrace{w_{N-1}, \ldots, w_1, w_0}_{\mathbf{w}_R}]$$

and moreover, in order to preserve the TDAC property, the windows must satisfy the following so-called Princen-Bradley condition expressed in terms of the length-$2N$ window $\hat{w}_n$, or, in terms of its half portion, w_n, for, $n = 0, 1, \ldots, N-1$,

$$\hat{w}_n^2 + \hat{w}_{n+N}^2 = 2 \quad \Rightarrow \quad w_n^2 + w_{N-1-n}^2 = 2$$

Define the diagonal matrices of the windows,

$$\hat{W} = \mathrm{diag}(\hat{\mathbf{w}}) = \left[\begin{array}{c|c} W & 0 \\ \hline 0 & W_R \end{array}\right], \quad W = \mathrm{diag}(\mathbf{w}), \quad W_R = \mathrm{diag}(\mathbf{w}_R) = JWJ$$

for example, for $N = 4$, and $\mathbf{w} = [w_0, w_1, w_2, w_3]$, we have,

$$W = \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_2 & 0 \\ 0 & 0 & 0 & w_3 \end{bmatrix}$$

$$W_R = \begin{bmatrix} w_3 & 0 & 0 & 0 \\ 0 & w_2 & 0 & 0 \\ 0 & 0 & w_1 & 0 \\ 0 & 0 & 0 & w_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_0 & 0 & 0 & 0 \\ 0 & w_1 & 0 & 0 \\ 0 & 0 & w_2 & 0 \\ 0 & 0 & 0 & w_3 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Then, the windowing operation on a length-$2N$ block, assumed to be a column, can be expressed as a matrix multiplication by the diagonal window matrices,

$$\mathbf{x} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \end{bmatrix} \quad \Rightarrow \quad \hat{W}\mathbf{x} = \begin{bmatrix} W\mathbf{a} \\ W_R\mathbf{b} \end{bmatrix}$$

The combined operations of windowing an input block, computing its MDCT followed by an IMDCT, and windowing the result with the same window are as follows,

$$\mathbf{x} \xrightarrow{\text{window}} \hat{W}\mathbf{x} = \begin{bmatrix} W\mathbf{a} \\ W_R\mathbf{b} \end{bmatrix} \xrightarrow{\text{MDCT/IMDCT}} \begin{bmatrix} \frac{1}{2}(W\mathbf{a} - W_R\mathbf{a}_R) \\ \frac{1}{2}(W_R\mathbf{b} + W\mathbf{b}_R) \end{bmatrix} \xrightarrow{\text{window}}$$

$$\xrightarrow{\text{window}} \begin{bmatrix} \frac{1}{2}(W^2\mathbf{a} - WW_R\mathbf{a}_R) \\ \frac{1}{2}(W_R^2\mathbf{b} + W_RW\mathbf{b}_R) \end{bmatrix} = \hat{W}\mathbf{y}$$

where we made use of the properties, $W_R = JWJ$ and $J^2 = I$, which imply,

$$(W\mathbf{a})_R = JW\mathbf{a} = JWJJ\mathbf{a} = W_R\mathbf{a}_R$$
$$(W_R\mathbf{b})_R = JW_R\mathbf{b} = J^2WJ\mathbf{b} = W\mathbf{b}_R$$

When two 50% overlapping blocks are subjected to these operations and added, then the overlapping portions (i.e., second half of the first block and first half of the second) will combine as,

$$\frac{1}{2}(W_R^2\mathbf{b} + W_R W\mathbf{b}_R) + \frac{1}{2}(W^2\mathbf{b} - WW_R\mathbf{b}_R) = \frac{1}{2}(W^2 + W_R^2)\mathbf{b}$$

where $\mathbf{b}_R$ term was cancelled because the diagonal matrices W, W_R commute. Thus, in order to guarantee the TDAC property, the W matrix must satisfy the following condition, which is equivalent to the Princen-Bradley condition,

$$\frac{1}{2}(W^2 + W_R^2) = I$$

A couple of window examples are as follows,

$$\text{(sine):} \quad \hat{w}_n = \sqrt{2}\,\sin\left(\frac{\pi}{2N}\left(n+\tfrac{1}{2}\right)\right), \quad n = 0, 1, \ldots, 2N-1$$

$$\text{(vorbis):} \quad \hat{w}_n = \sqrt{2}\,\sin\left[\frac{\pi}{2}\sin^2\left(\frac{\pi}{2N}\left(n+\tfrac{1}{2}\right)\right)\right], \quad n = 0, 1, \ldots, 2N-1$$

The first is used in MP3 and MPEG-2 AAC formats, and the second in Vorbis.

A more general procedure for constructing such windows begins with a typical length-$(N+1)$ window, say, f_k, $k = 0, 1, \ldots, N$, that is symmetric about its middle (i.e., about $\frac{1}{2}N$), such as a Kaiser or a Hamming window, and constructs the first length-N half of the window by forming the square-root of the cumulative sum of f_k,

$$\hat{w}_n = w_n = \left[\frac{2}{S}\sum_{k=0}^{n} f_k\right]^{1/2}, \quad n = 0, 1, \ldots, N-1, \quad \text{where} \quad S \equiv \sum_{k=0}^{N} f_k$$

then, the second length-N half of the window is taken to be the reversed version of w_n, obtained by replacing n by $N-1-n$,

$$\hat{w}_{n+N} = w_{N-1-n} = \left[\frac{2}{S}\sum_{m=0}^{N-1-n} f_m\right]^{1/2} = \left[\frac{2}{S}\sum_{k=n+1}^{N} f_{N-k}\right]^{1/2}$$

where we changed summation variables from m to $N-k$. Exploiting the assumed symmetry of the f_k window, i.e., $f_{N-k} = f_k$, we obtain,

$$\hat{w}_{n+N} = w_{N-1-n} = \left[\frac{2}{S}\sum_{k=n+1}^{N} f_k\right]^{1/2}, \quad n = 0, 1, \ldots, N-1$$

Together the above equations define a length-$2N$ window w_n that satisfies the Prince-Bradley condition, indeed,

$$\hat{w}_n^2 + \hat{w}_{n+N}^2 = \frac{2}{S}\left[\sum_{k=0}^{n} f_k + \sum_{k=n+1}^{N} f_k\right] = \frac{2}{S}\left[\sum_{k=0}^{N} f_k\right] = \frac{2}{S}\,S = 2$$

An example of such window is the so-called Kaiser-Bessel derived (KBD) window, which is used in AAC and Dolby AC-3 formats.

It is generated by the above procedure from an ordinary Kaiser window of length $(N+1)$ and shape parameter β,

$$f_k = I_0\left(\beta\sqrt{1-\left(\frac{k-N/2}{N/2}\right)^2}\right), \quad k = 0, 1, \ldots, N$$

Similarly, a Hamming window would have,

$$f_k = 0.54 - 0.46\cos\left(\frac{2\pi k}{N}\right), \quad k = 0, 1, \ldots, N$$

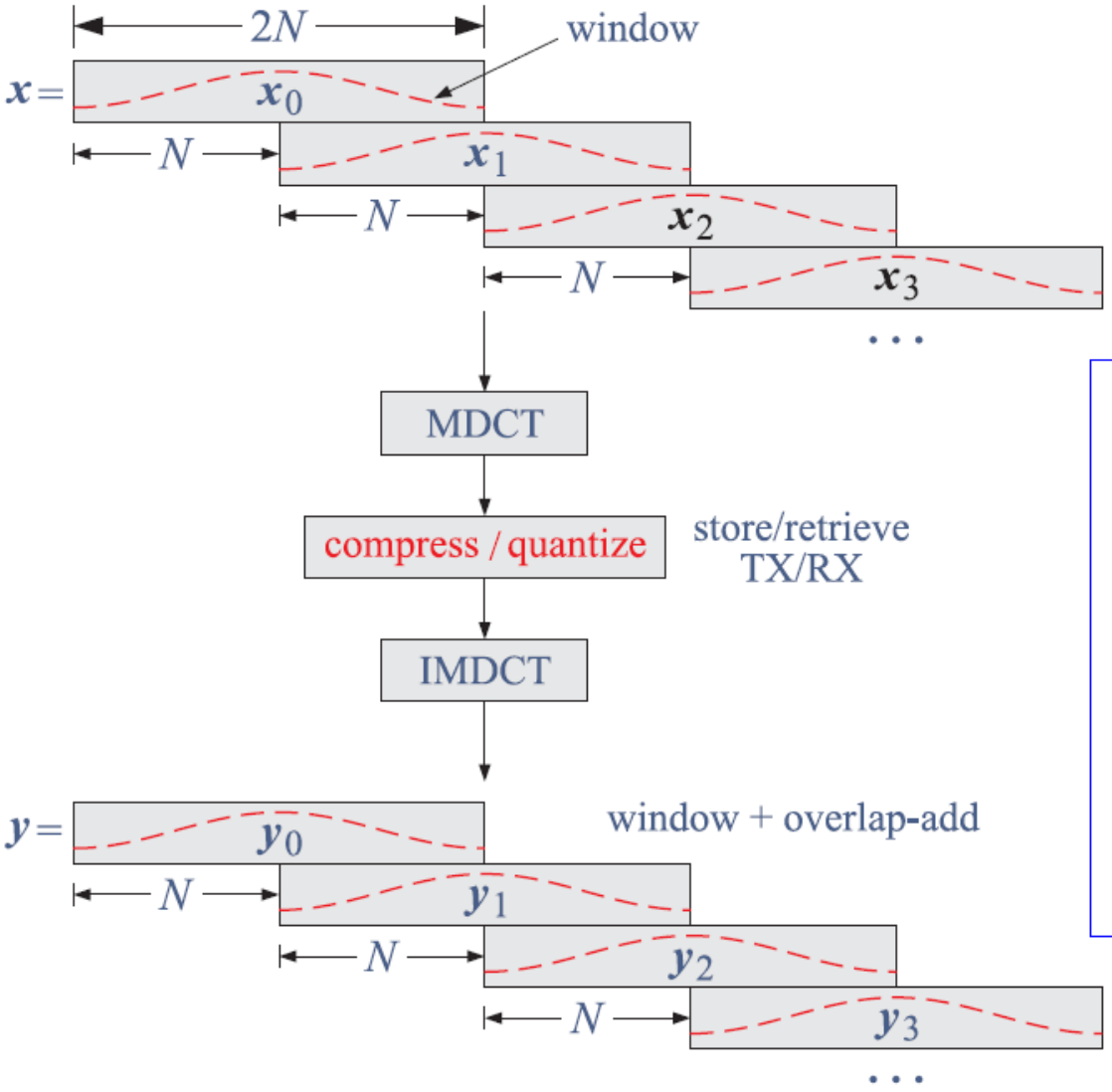
```
% MATLAB code for constructing such windows

f = ...                                % choose f(k), n=0,1,...,N
                                       % length-(N+1) column vector

w = sqrt(2*cumsum(f)/sum(f));          % left-half of window, w(n)

w = w(1:end-1);                        % keep only, w(n), n=0,1,...,N-1

w = [w; flip(w)];                      % add right-half, symmetrically
```

Summary –MDCT compression system

In project-7, you will need to write a funrction, **mdctcompr**, as well as functions **mdct**, **imdct**, **pbwin**, that implement the MDCT/TDAC compression scheme.

```
[y,ra] = mdctcompr(x,N,rth,win,beta);

% x = signal to be compressed
% N = MDCT length, frame length = 2*N
% rth = compression threshold, rth<1, (rth=0 for no compression)
% win = 0,1,2,3, for rectangular, sine, vorbis, KBD windows
% beta = Kaiser shape parameter when win = 3
%
% y = compressed signal, same size as x
% ra = actual compression ratio achieved
```

Your function must incorporate the following operations,



$X = \mathbf{buffer}(\mathbf{x}, 2N, N, \texttt{'nodelay'})$

$\mathbf{w} = \mathbf{pbwin}(N, \texttt{win}, \beta)$

$W = \mathbf{repmat}(\mathbf{w}, 1, M)$

$D = \mathbf{mdct}(W .* X)$

$C = f(D, r_{\text{thr}}) = \text{compressed MDCT}$

$Y = W .* \mathbf{imdct}(C)$

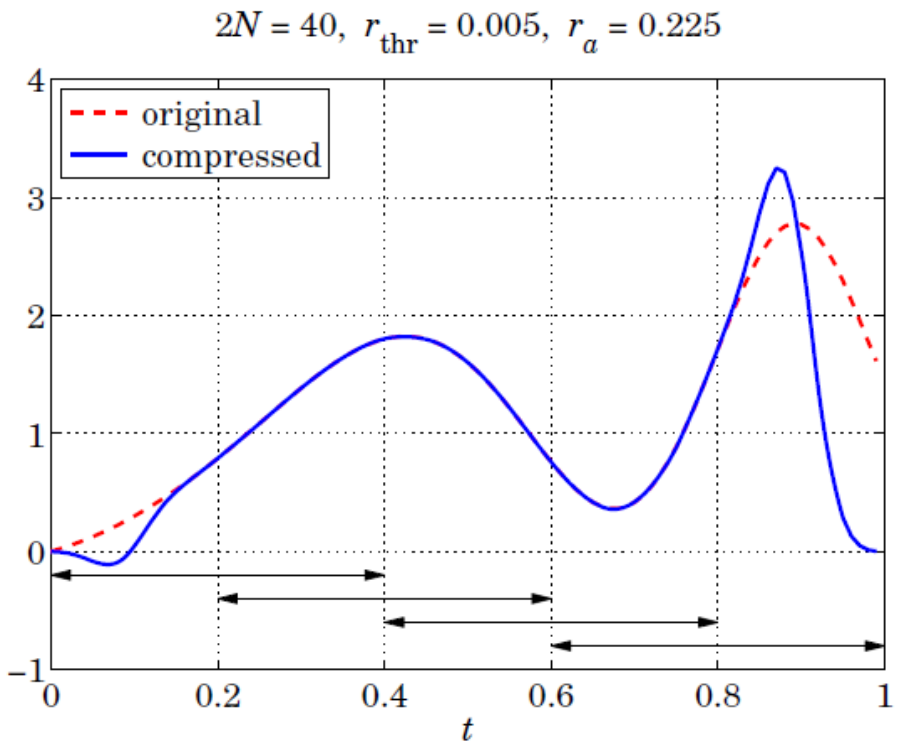
$\mathbf{y} = \mathbf{ola}(Y, N)$

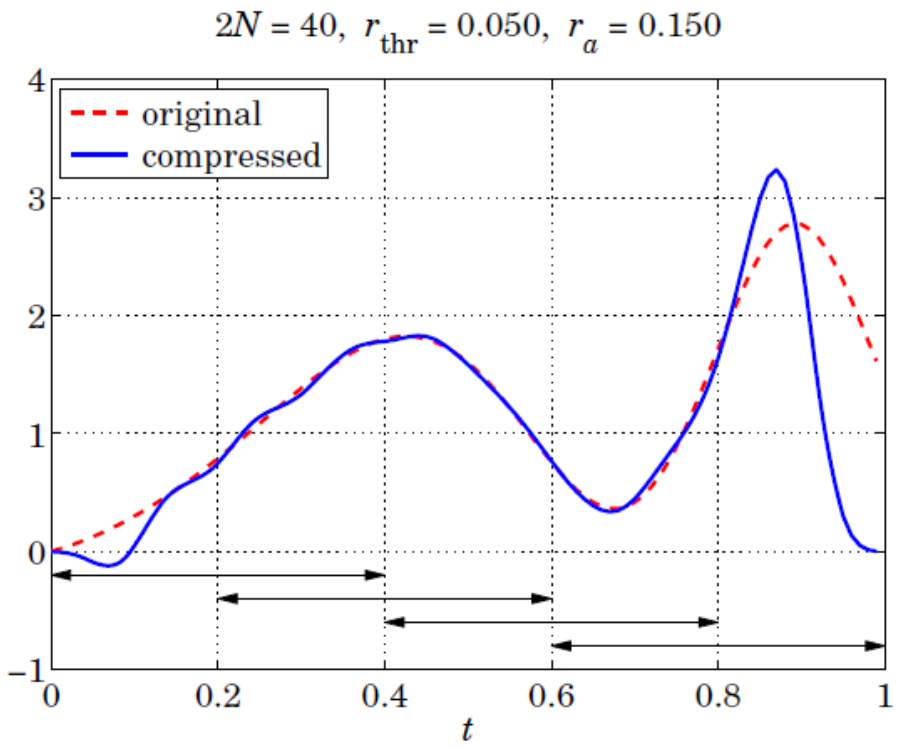
$\mathbf{y} = \mathbf{y}\big(1 : \text{length}(\mathbf{x})\big)$

method-2

example

```
N = 20; M = 4; L = N*M + N;

t = (0:L-1)'/L;
x = sin(10*t.^2) + 2*t;

beta = 15;
rthr = 0.005;          % try also, rthr = 0.05

[y,ra] = mdctcompr(x,N,rthr,3,beta);

figure; plot(t,x,'r--', t,y,'b-');
```





To clarify the overall sequence of computational steps, we list below the explicit steps for the same example, such steps can be the basis of constructing the function, **mdctcompr**

example

```
N = 20; M = 4; L = N*M + N;

t = (0:L-1)'/L;
x = sin(10*t.^2) + 2*t;

beta = 15;
rthr = 0.005;              % try also, rthr = 0.05
```

```
w = pbwin(N,3,beta);                    % KBD window

X = buffer(x,2*N,N,'nodelay');          % 50% overlapping segments

M = size(X,2);                          % no. of segments
W = repmat(w,1,M);                      % replicate window M times

D = mdct(W.*X);                         % MDCT of all windowed frames

Dmax = max(max(abs(D)));                % max MDCT coefficient
Dthr = rthr * Dmax;                     % MDCT threshold

D(abs(D) < Dthr)=0;                        % discard all |D| < Dthr
ra = length(find(D))/prod(size(D));     % actual compression ratio

Y = W.*imdct(D);                        % IMDCT following by windowing

y = ola(Y,N);                           % OLA with hop-size N
y = y(1:length(x));                     % make lengths equal

figure; plot(t,x,'r--', t,y,'b-');
```

To display more clearly the TDAC mechanism, we compute and list the output frames separately, for a simple signal and KBD window, assuming no compression so that TDAC will be exact.

```
N = 4; M = 6; L = N*M + N

x = (1:L)';

w = pbwin(N,3,6);          % KBD, with beta=6

W = sparse(diag(w));       % sparsify when N is large

X = W * buffer(x,2*N,N,'nodelay');

D = mdct(X);

Y = W * imdct(D);

y = ola(Y,N);              % OLA with hop size N

Ym = zeros(N*M+N,M);       % list frames separately

for m=1:M,
    Ym((m-1)*N + (1:2*N), m) = Y(:,m);
end

num2str([x, Ym, y], '% 8.2f')
```

x	y1	y2	y3	y4	y5	y6	y
1	-0.34	0.00	0.00	0.00	0.00	0.00	-0.34
2	-0.80	0.00	0.00	0.00	0.00	0.00	-0.80
3	1.39	0.00	0.00	0.00	0.00	0.00	1.39
4	3.88	0.00	0.00	0.00	0.00	0.00	3.88
5	5.65	-0.65	0.00	0.00	0.00	0.00	5.00
6	7.53	-1.53	0.00	0.00	0.00	0.00	6.00
7	4.34	2.66	0.00	0.00	0.00	0.00	7.00
8	0.49	7.51	0.00	0.00	0.00	0.00	8.00
9	0.00	9.97	-0.97	0.00	0.00	0.00	9.00
10	0.00	12.27	-2.27	0.00	0.00	0.00	10.00
11	0.00	7.07	3.93	0.00	0.00	0.00	11.00
12	0.00	0.86	11.14	0.00	0.00	0.00	12.00
13	0.00	0.00	14.28	-1.28	0.00	0.00	13.00
14	0.00	0.00	17.00	-3.00	0.00	0.00	14.00
15	0.00	0.00	9.80	5.20	0.00	0.00	15.00
16	0.00	0.00	1.24	14.76	0.00	0.00	16.00
17	0.00	0.00	0.00	18.59	-1.59	0.00	17.00
18	0.00	0.00	0.00	21.73	-3.73	0.00	18.00
19	0.00	0.00	0.00	12.53	6.47	0.00	19.00
20	0.00	0.00	0.00	1.61	18.39	0.00	20.00
21	0.00	0.00	0.00	0.00	22.91	-1.91	21.00
22	0.00	0.00	0.00	0.00	26.46	-4.46	22.00
23	0.00	0.00	0.00	0.00	15.26	7.74	23.00
24	0.00	0.00	0.00	0.00	1.99	22.01	24.00
25	0.00	0.00	0.00	0.00	0.00	27.22	27.22
26	0.00	0.00	0.00	0.00	0.00	31.20	31.20
27	0.00	0.00	0.00	0.00	0.00	17.99	17.99
28	0.00	0.00	0.00	0.00	0.00	2.36	2.36