



## BLM3620 Digital Signal Processing

Dr. Ali Can KARACA

*[ackaraca@yildiz.edu.tr](mailto:ackaraca@yildiz.edu.tr)*

Yıldız Technical University – Computer Engineering

# Lecture #11 – Fast Fourier Transform and Windowing

- Fast Fourier Transform
- Examples
- Windowing
- MATLAB Applications

## ***Important Materials:***

- James H. McClellan, R. W. Schafer, M. A. Yoder, *DSP First Second Edition*, Pearson, 2015.
- Lizhe Tan, Jean Jiang, *Digital Signal Processing: Fundamentals and Applications*, Third Edition, Academic Press, 2019.

## ***Auxiliary Materials:***

- Prof. Sarp Ertürk, *Sayısal İşaret İşleme*, Birsen Yayınevi.
- Prof. Nizamettin Aydın, DSP Lecture Notes.
- J. G. Proakis, D. K. Manolakis, *Digital Signal Processing Fourth Edition*, Pearson, 2014.
- J. K. Perin, *Digital Signal Processing, Lecture Notes*, Stanford University, 2018.

# Syllabus



| Week | Lectures  |
|------|---|
| 1    | Introduction to DSP and MATLAB                  |
| 2    | Sinuzoids and Complex Exponentials              |
| 3    | Spectrum Representation                         |
| 4    | Sampling and Aliasing                           |
| 5    | Discrete Time Signal Properties and Convolution |
| 6    | Convolution and FIR Filters                     |
| 7    | Frequency Response of FIR Filters               |
| 8    | Midterm Exam                                    |
| 9    | Discrete Time Fourier Transform and Properties  |
| 10   | Discrete Fourier Transform and Properties       |
| 11   | Fast Fourier Transform and Windowing            |
| 12   | z- Transforms                                   |
| 13   | FIR Filter Design and Applications              |
| 14   | IIR Filter Design and Applications              |
| 15   | Final Exam                                      |

For more details -> Bologna page: <http://www.bologna.yildiz.edu.tr/index.php?r=course/view&id=5730&aid=3>

# Review & Recall



**Discrete Fourier Transform (DFT)**

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn}$$

**Inverse DFT**

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j(2\pi/N)kn}$$

**Discrete-time Fourier Transform (DTFT)**

$$X(e^{j\hat{\omega}}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\hat{\omega}n}$$

**Inverse DTFT**

$$x[n] = \frac{1}{2\pi} \int_0^{2\pi} X(e^{j\hat{\omega}}) e^{j\hat{\omega}n} d\hat{\omega}$$

# Recap: MATLAB Code for DFT



```
clc; clear all;  
%%  
x = [1 2 2 1 1 2 3 4];  
N = length(x);  
X = zeros(1,N);
```

```
for k = 0:N-1  
    for n = 0:N-1  
        X(k+1) = X(k+1) + x(n+1)*exp(-j*(2*pi/N)*k*n);  
    end  
end
```

```
X  
fft(x)
```

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j(2\pi/N)kn}$$

What is the algorithm complexity here?

# Fast Fourier Transform (FFT)



**FFT** is an algorithm that allows to obtain the same DFT results with a faster way.

Most of applications in real world use FFT instead of DFT.

Algorithm complexity of DFT is  $O(N^2)$ . It requires  $N$  complex-multiplication for each  $k$  sample.

Therefore, if  $N=2^{10} \rightarrow N^2 = 2^{20}$ . Too much computational complexity!!!

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

# Fast Fourier Transform (FFT)



Algorithm complexity of DFT is  $O(N^2)$ .

**Algorithm complexity of FFT is  $\frac{N}{2} O(\log_2 N)$**

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

Main Idea:

Divide N-length signal into two N/2-length sub-parts (even and odd indices) until reaching 2-length signals.

*Can be calculated using two ways:*

- 1) Decimation in time
- 2) Decimation in frequency





# Fast Fourier Transform (FFT)



$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

Let's define a Phase Factor:

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

Using the symmetry and periodicity properties of DFT, we can write:

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

$$W_N^{k+N} = e^{-j\frac{2\pi}{N}(k+N)} = W_N^k$$

Benefitting from these properties, we can calculate FFT in a faster way.

Decimation in time algorithm is also known as Radix-2 algorithm.

# Derivation of FFT Equations



N-point DFT:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{N-1} x[n] W_N^{kn}$$

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

Decompose the signal into even part ( $x[2n]$ ) and odd part ( $x[2n+1]$ ):

$$X[k] = \sum_{n=0}^{N/2-1} x[2n] W_N^{k2n} + \sum_{n=0}^{N/2-1} x[2n+1] W_N^{k(2n+1)}$$

$$W_N^{k(2n)} = e^{-j\frac{2\pi 2kn}{N}} = e^{-j\frac{2\pi kn}{N/2}} = W_{N/2}^{kn}$$

By using this property, we can rewrite the equation as:

$$X[k] = \sum_{n=0}^{N/2-1} x[2n] W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1] W_{N/2}^{kn}$$

# Derivation of FFT Equations



$$X[k] = \sum_{n=0}^{N/2-1} x[2n] W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x[2n+1] W_{N/2}^{kn}$$

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

Here, we can rename the signals as  $x_1[n] = x[2n]$  ve  $x_2[n] = x[2n+1]$

$$X[k] = \sum_{n=0}^{N/2-1} x_1[n] W_{N/2}^{kn} + W_N^k \sum_{n=0}^{N/2-1} x_2[n] W_{N/2}^{kn}$$

This bring us to the final equation:

$$X[k] = X_1[k] + W_N^k X_2[k]$$

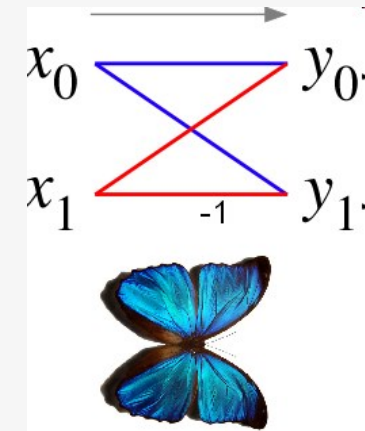
$$X[k + N/2] = X_1[k] - W_N^k X_2[k]$$

Conjugate symmetry property

# Derivation of FFT Equations

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

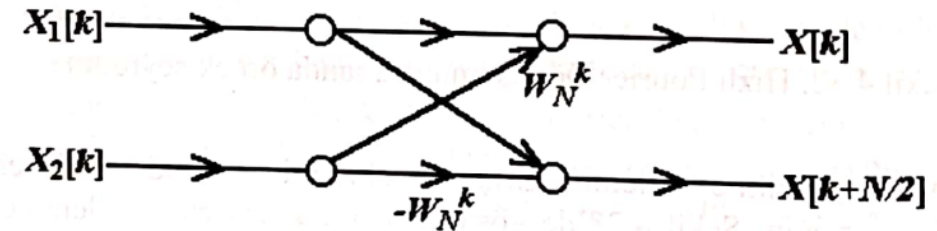
Butterfly  
FFT



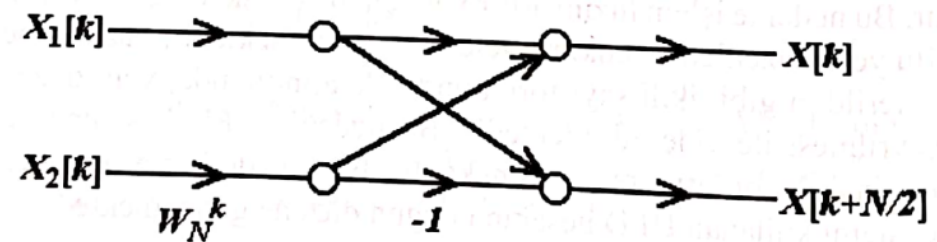
$$X[k] = X_1[k] + W_N^k X_2[k]$$

$$X[k + N/2] = X_1[k] - W_N^k X_2[k]$$

*Temel kelebek:*



*Basit kelebek:*



# Algorithm of FFT



## 8-pt FFT algorithm:

- Önce tek ve çift örnekleri belirleyerek iki parçaya ayır  $\rightarrow (x_t, x_{\bar{c}})$
- Sonra tek ve çift örnekleri de ikiye ayır  $\rightarrow (x_{tt}, x_{t\bar{c}}, x_{\bar{c}t}, x_{\bar{c}\bar{c}})$
- Her ikili çift için AFD'sini hesapla.  $\rightarrow (X_{tt}, X_{t\bar{c}}, X_{\bar{c}t}, X_{\bar{c}\bar{c}})$
- Aşağıdaki denklemleri kullanarak tek ve çift örneklerin AFD'sini bul.  $\rightarrow (X_t, X_{\bar{c}})$

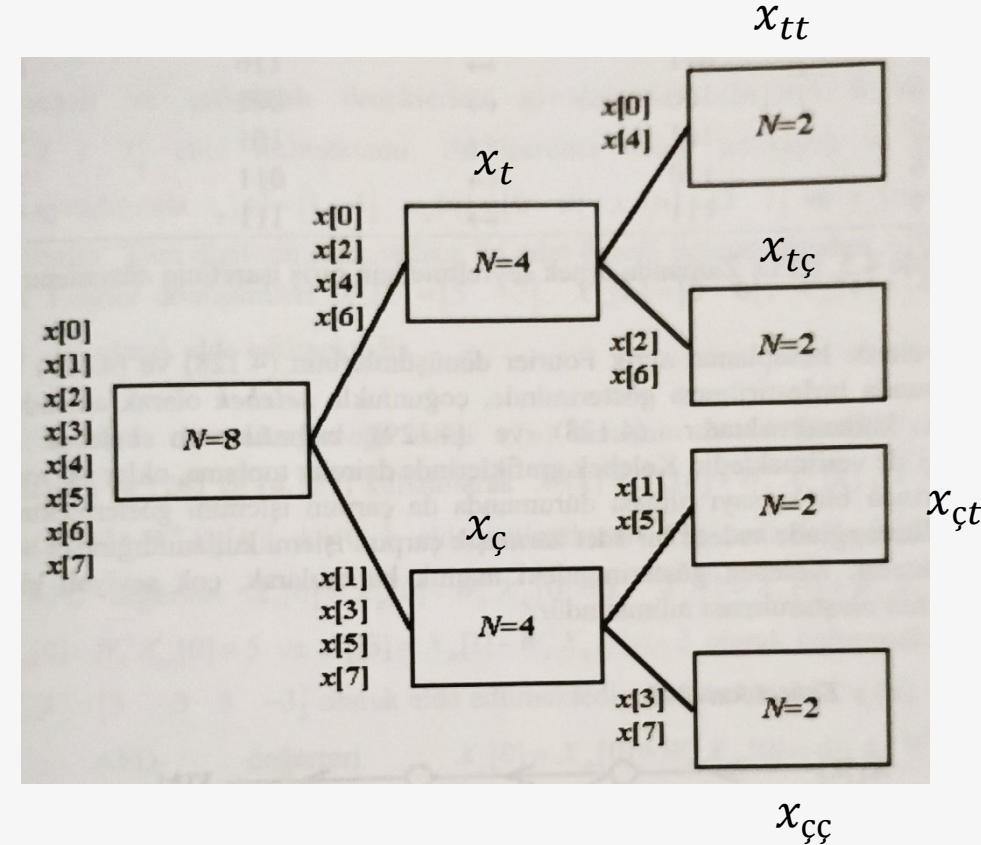
$$X_t[k] = X_{tt}[k] + W_4^k X_{t\bar{c}}[k]$$

$$X_t[k + N/2] = X_{tt}[k] - W_4^k X_{t\bar{c}}[k]$$

- Tek ve çift örneklerin AFD'sini kullanarak denklemlerle sonucu bul.  $\rightarrow (X)$

$$X[k] = X_t[k] + W_8^k X_{\bar{c}}[k]$$

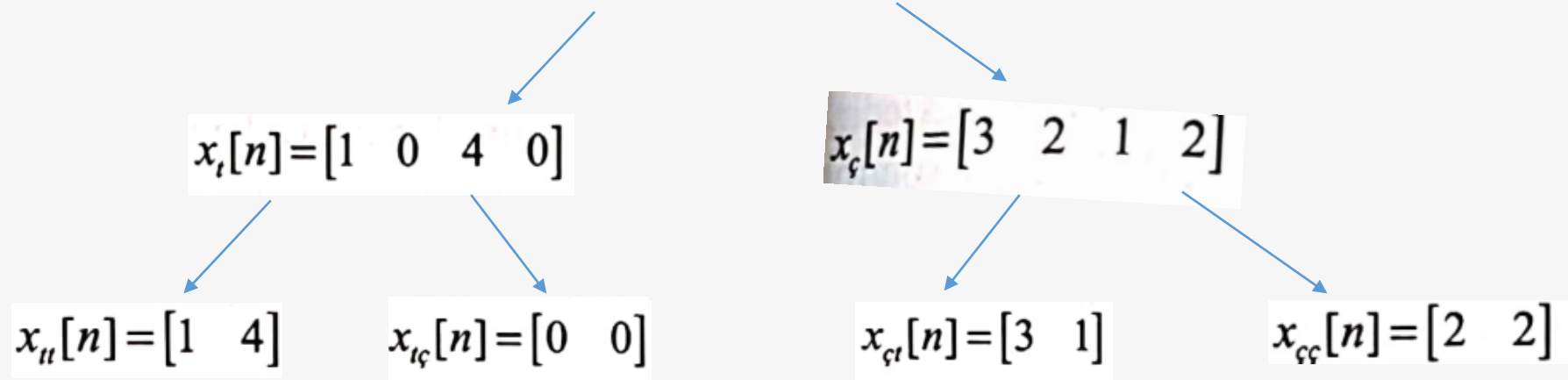
$$X[k + N/2] = X_t[k] - W_8^k X_{\bar{c}}[k]$$



# Example:

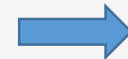


Hızlı Fourier dönüşümü kullanarak  $x[n] = [1 \ 3 \ 0 \ 2 \ 4 \ 1 \ 0 \ 2]$  işaretinin ayrık Fourier dönüşümünü hesaplayınız.



$$X_{tt}[0] = \sum_{n=0}^1 x_{tt}[n] e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^1 x_{tt}[n] e^{-j\pi 0n} = 5$$

$$X_{tt}[1] = \sum_{n=0}^1 x_{tt}[n] e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^1 x_{tt}[n] e^{-j\pi n} = 1 - 4 = -3$$



$$X_{tt}[k] = [5 \ -3]$$

$$X_{t\tau}[k] = [0 \ 0]$$

$$X_{\tau t}[k] = [4 \ 2]$$

$$X_{\tau\tau}[k] = [4 \ 0]$$

# Example:



$$\begin{array}{cccc} X_{tt}[k] = [5 & -3] & X_{t\zeta}[k] = [0 & 0] & X_{\zeta t}[k] = [4 & 2] & X_{t\zeta}[k] = [4 & 0] \\ & \searrow & \swarrow & & \searrow & \swarrow & \\ & X_t[k] = ? & & & X_\zeta[k] = ? & & \end{array}$$

$$X_t[k] = X_{tt}[k] + W_4^k X_{t\zeta}[k], \quad k = 0,1$$

$$X_t[k+2] = X_{tt}[k] - W_4^k X_{t\zeta}[k], \quad k = 0,1$$

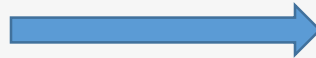
$$W_4^k = e^{-j\frac{2\pi}{N}k} = e^{-j\frac{\pi}{2}k}$$

$$X_t[0] = X_{tt}[0] + W_4^0 X_{t\zeta}[0] = 5$$

$$X_t[1] = X_{tt}[1] + W_4^1 X_{t\zeta}[1] = -3$$

$$X_t[0+2] = X_{tt}[0] - W_4^0 X_{t\zeta}[0] = 5$$

$$X_t[1+2] = X_{tt}[1] - W_4^1 X_{t\zeta}[1] = -3$$



$$X_t[k] = [5 \quad -3 \quad 5 \quad -3]$$

$$X_\zeta[k] = [8 \quad 2 \quad 0 \quad 2]$$

# Example:



$$X_t[k] = [5 \quad -3 \quad 5 \quad -3]$$

$$X_c[k] = [8 \quad 2 \quad 0 \quad 2]$$



$$X[k] = ?$$

$$X[k] = X_t[k] + W_8^k X_c[k], \quad k = 0, 1, 2, 3$$

$$X[k + 4] = X_t[k] - W_8^k X_c[k], \quad k = 0, 1, 2, 3$$

$$W_8^k = e^{-j\frac{2\pi}{N}k} = e^{-j\frac{\pi}{4}k}$$

$$X[0] = X_t[0] + W_8^0 X_c[0] = 5 + 8e^{-0.25\pi 0} = 13$$

$$X[1] = X_t[1] + W_8^1 X_c[1] = -3 + 2e^{-0.25\pi 1} = 1.5858 - j1.4142$$

$$X[2] = X_t[2] + W_8^2 X_c[2] = 5 + 0e^{-0.25\pi 2} = 5$$

$$X[3] = X_t[3] + W_8^3 X_c[3] = -3 + 2e^{-0.25\pi 3} = -4.4142 - j1.4142$$

$$X[4] = X_t[0] - W_8^0 X_c[0] = 5 - 8e^{-(j0.25\pi)0} = -3,$$

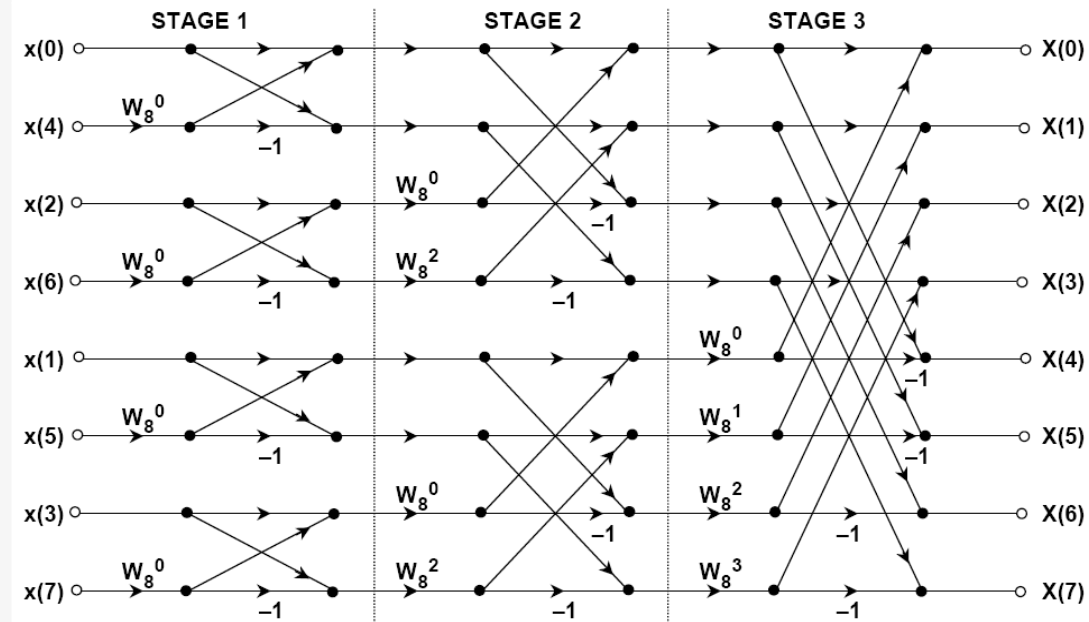
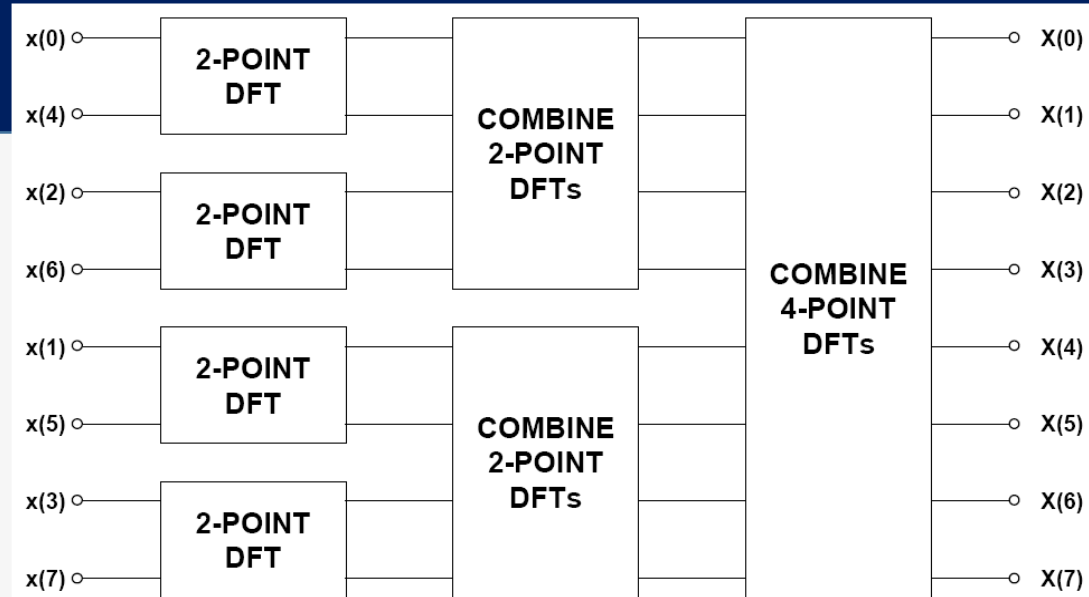
$$X[5] = X_t[1] - W_8^1 X_c[1] = -3 - 2e^{-(j0.25\pi)1} = -4.4142 + j1.4142,$$

$$X[6] = X_t[2] - W_8^2 X_c[2] = 5 - 0e^{-(j0.25\pi)2} = 5,$$

$$X[7] = X_t[3] - W_8^3 X_c[3] = -3 - 2e^{-(j0.25\pi)3} = -1.5858 + j1.4142$$



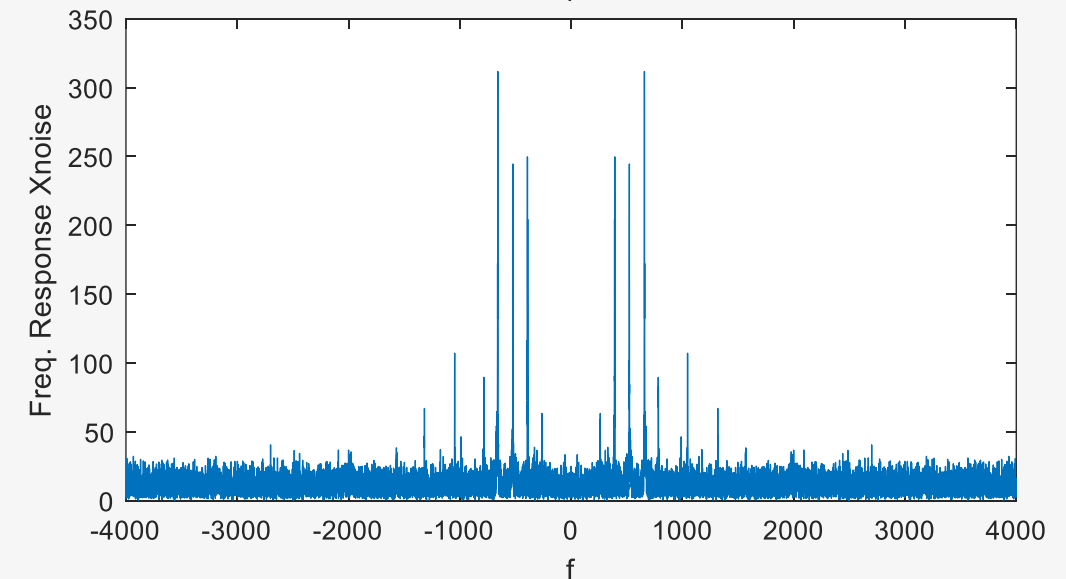
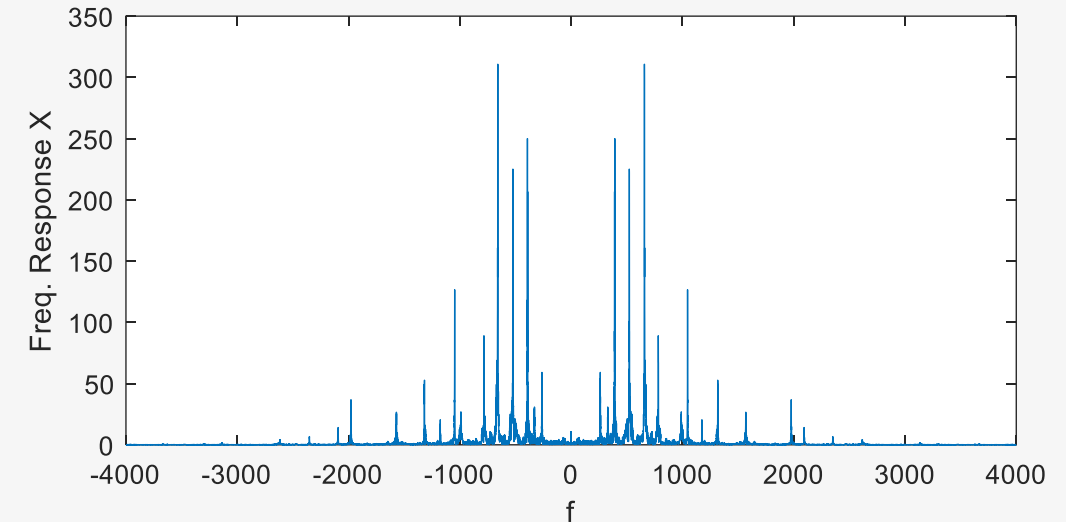
- Input signal must be properly re-ordered using a *bit reversal* algorithm
- In-place computation
- Number of stages:  $\log_2 N$
- Stage 1: all the twiddle factors are 1
- Last Stage: the twiddle factors are in sequential order



# Application – 1: FFT of a noisy sound



```
clc; clear all;
%% Load Sound
load ('piano2.mat');
N = 2^14; x = x(1:N);
%% FFT of the signal
X = fftshift(fft(x,N));
wHat = (-N/2:N/2-1) * Fs/N;
plot(wHat,abs(X)); xlabel('f');
ylabel('Freq. Response X');
%% Add Noise
Xnoise = awgn(x,20);
XnoiseF = fftshift(fft(Xnoise,N));
figure(2);
plot(wHat,abs(XnoiseF));
xlabel('f'); ylabel('Freq. Response
Xnoise');
```



# Application – 2: Filtering in Fourier Domain

```

N = 2^14; x = x(1:N);
f = (-N/2:N/2-1) * Fs/N;

%% FFT of the signal
X = fftshift(fft(x,N));
Xnoise = awgn(x,35);
Xnoisef = fftshift(fft(Xnoise,N));
figure(1);
plot(f,abs(Xnoisef));
xlabel('f'); ylabel('Freq. Response Xnoise');
sound(real(Xnoise));

%% Generate any signal in FFT domain
H = zeros(size(X));
H(4097:12279)=1;
figure(2); plot(f,abs(Xnoisef.*H)); xlabel('f'); ylabel('Freq. Response Result');
y = ifft(ifftshift(Xnoisef.*H));
sound(real(y));

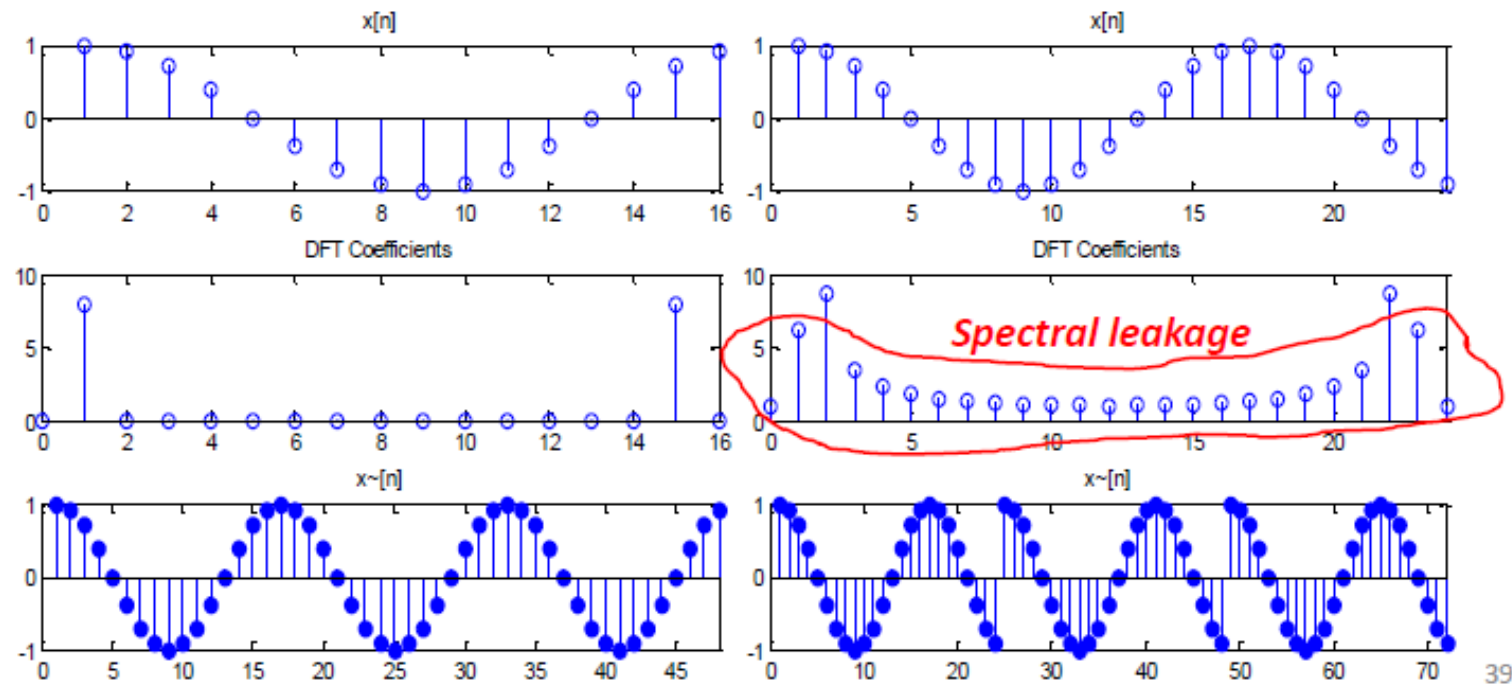
```

|             |                                     |            |
|-------------|-------------------------------------|------------|
| Convolution | $\sum_{m=0}^{N-1} h[m]x[((n-m))_N]$ | $H[k]X[k]$ |
|-------------|-------------------------------------|------------|

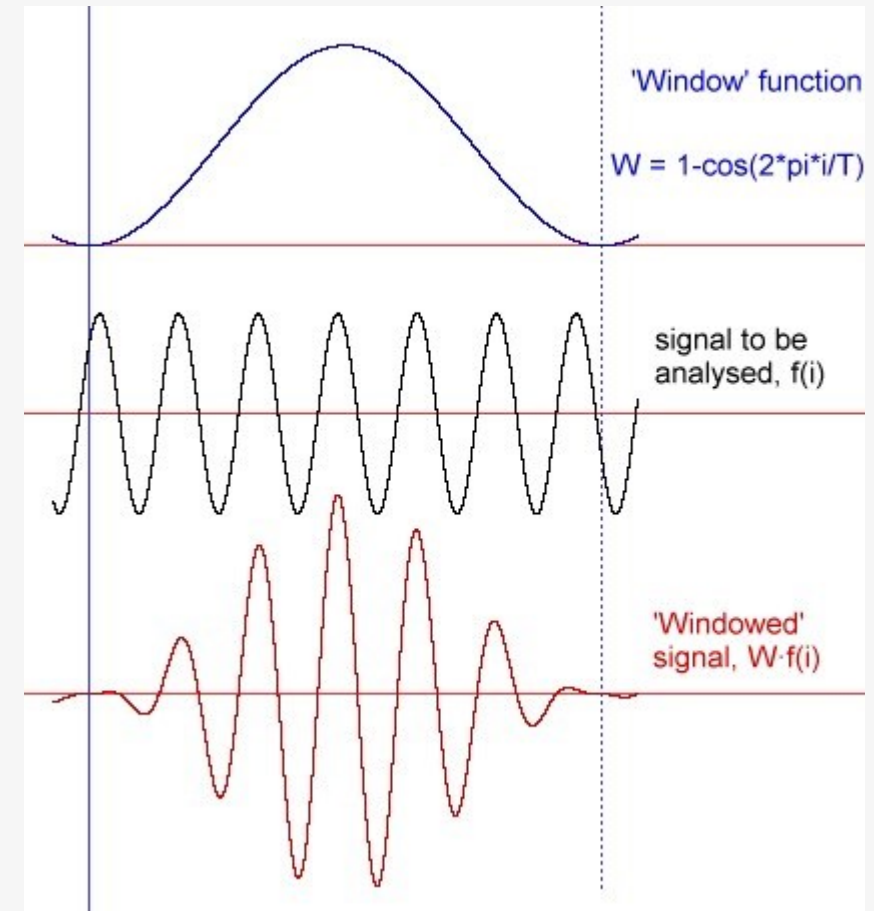
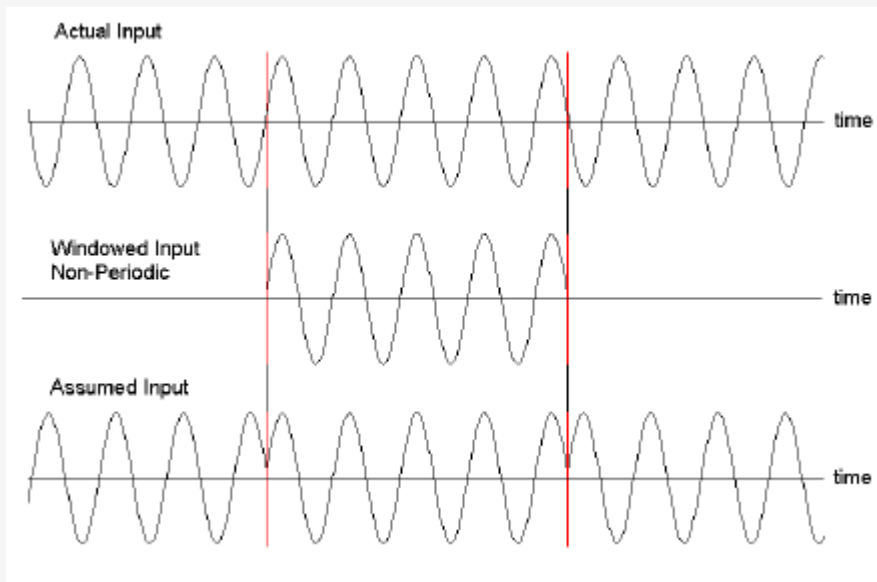
Convolution in time = Multiplication in freq.

# Windowing

- *Periodicity* causes unwanted spectral effects in frequency domain
- This issue is called as ***spectral leakage***.



# Windowing



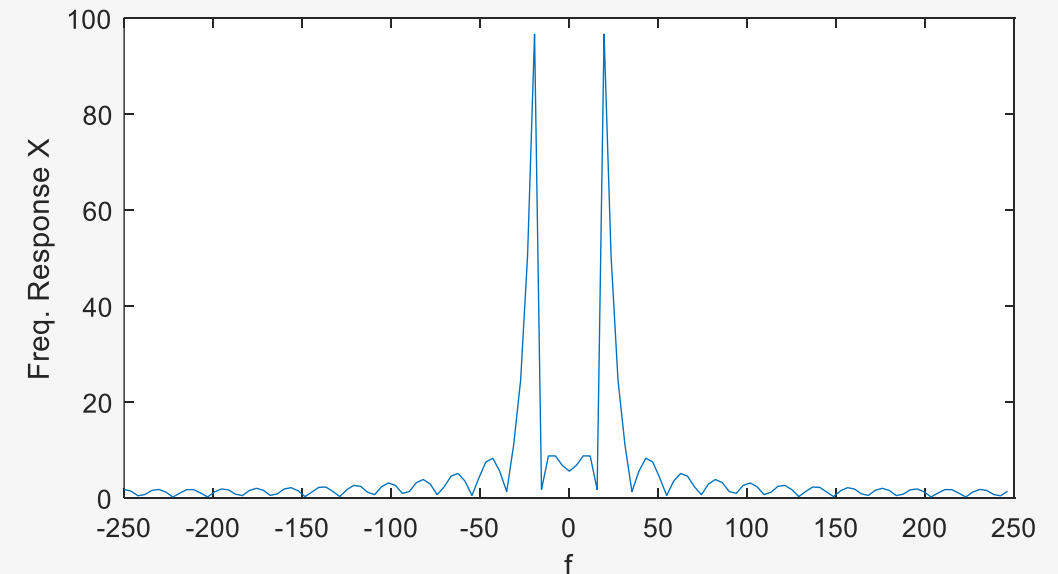
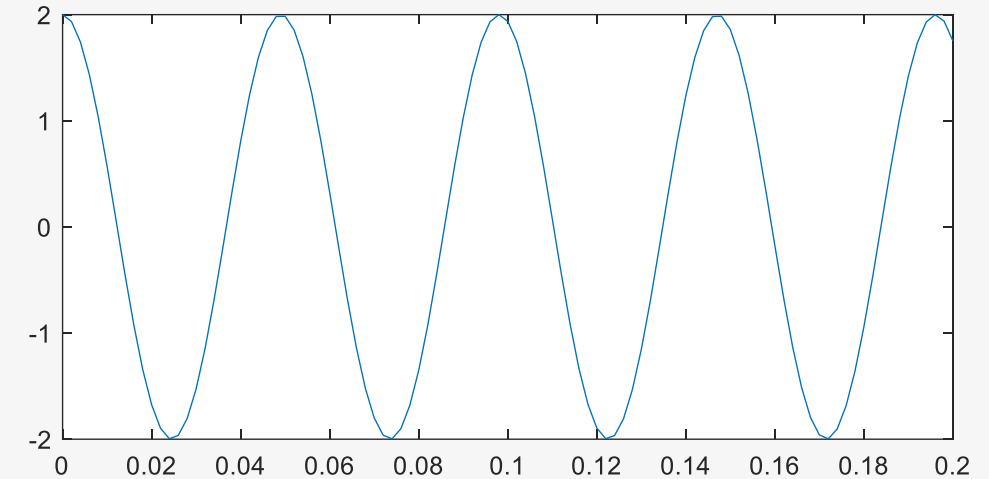
# Example



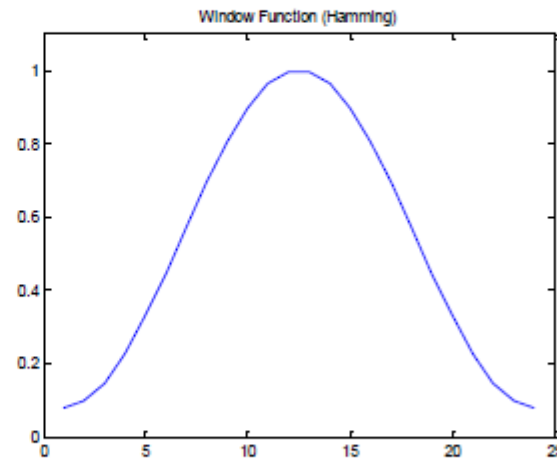
```
clc; clear all;

%% Load Sound
Fs = 500;
n = 0:1/Fs:0.2;
x = 2*cos(40.8*pi*n);
plot(n,x);
N = 128;

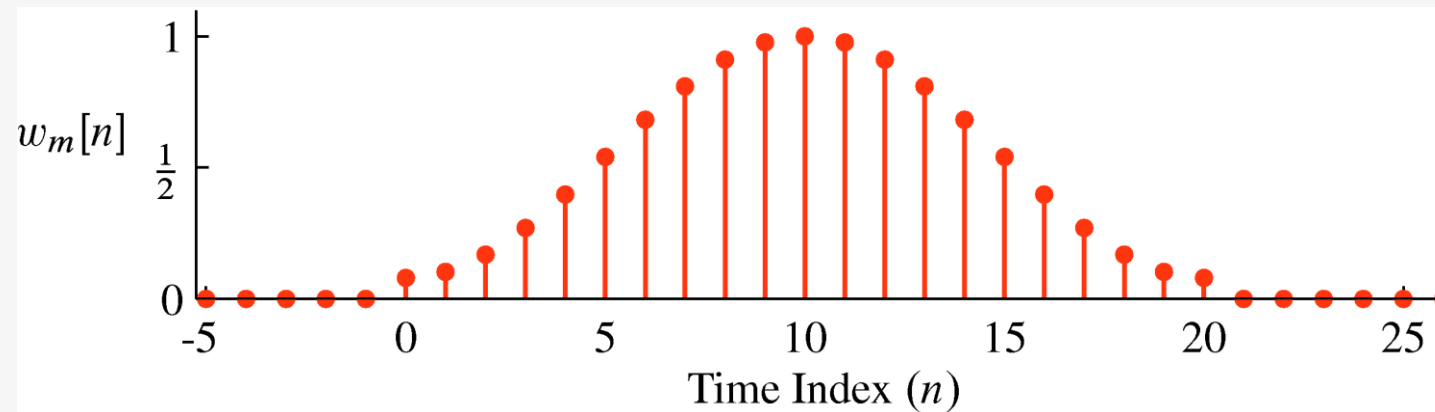
%% FFT of the signal
X = fftshift(fft(x,N));
f = (-N/2:N/2-1) * Fs/N;
plot(f,abs(X)); xlabel('f'); ylabel('Freq. Response X');
```



- Windowing is utilized to overcome this effect.
- Well known window functions:
  - Triangular
  - Trapezoid
  - Hamming
  - Hanning
  - Blackman
  - Parzen
  - Welch
  - Nuttall
  - Kaiser



- Plot of Length-21 Hamming window



Hamming Window

$$w_m[n] = \begin{cases} 0 & n < 0 \\ 0.54 - 0.46 \cos(2\pi(n)/(L-1)) & 0 \leq n < L \\ 0 & n \geq L \end{cases}$$



# Example



```
clc; clear all;
```

```
%% Load Sound
```

```
Fs = 500;
```

```
n = 0:1/Fs:0.2;
```

```
x = 2*cos(40.8*pi*n);
```

```
plot(n,x);
```

```
N = 128;
```

```
%% FFT of the signal
```

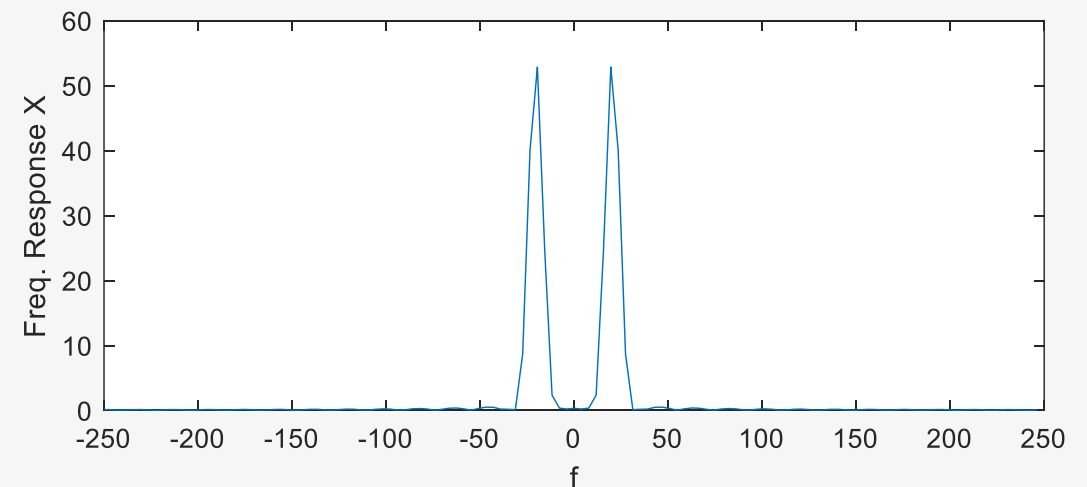
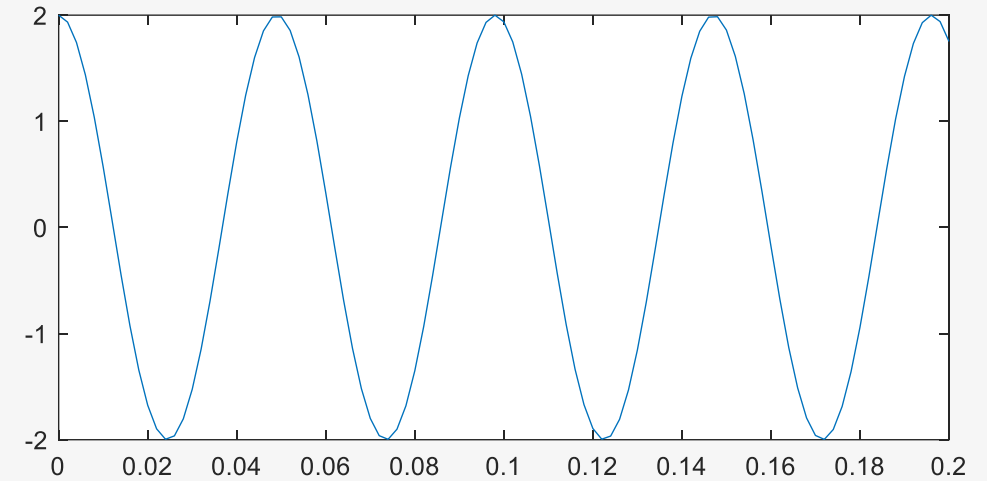
```
w = hamming(101)';
```

```
x = x.*w;
```

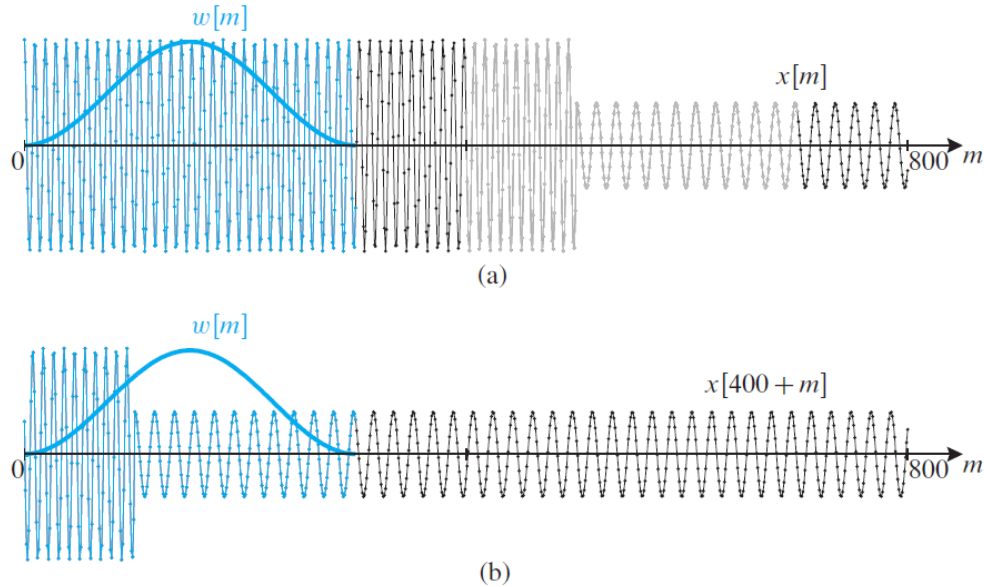
```
X = fftshift(fft(x,N));
```

```
f = (-N/2:N/2-1) * Fs/N;
```

```
plot(f,abs(X)); xlabel('f'); ylabel('Freq.  
Response X');
```



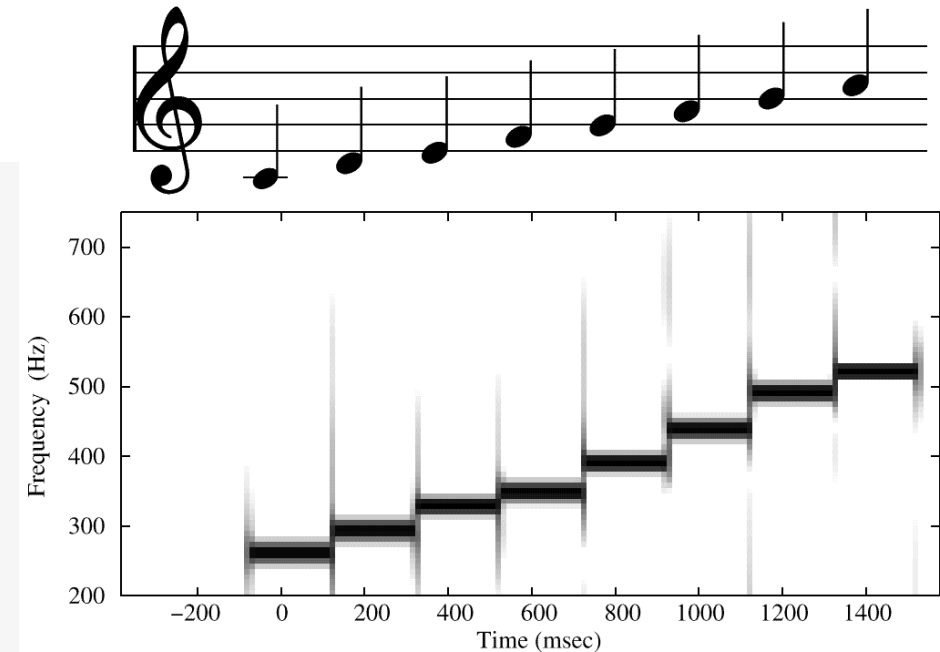
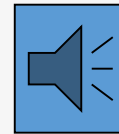
# Homework: Spectrogram (aka Short-Time FT)



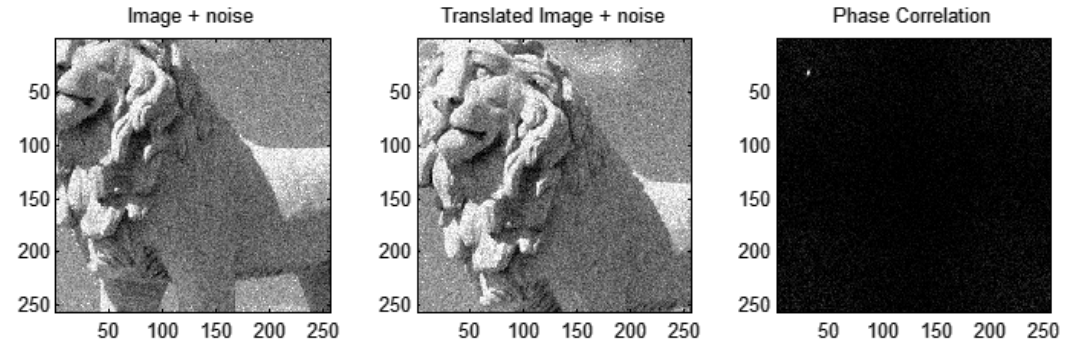
**Figure 8-19** Time-dependent Hann windowing of a signal using a length-301 window. (a) The signal  $x[0+m] = x[m]$  and fixed window  $w[m]$ . (b) Fixed window  $w[m]$  with signal shifted by 400 samples (i.e.,  $x[400+m]$ ).

Write a code that generates spectrogram of a short song:

- 1- User can select the window length ( $L$ ),
- 2- You must indicate the Frequency and Time on figure,
- 3- You must analyze the effect of window length.



# Application-3: Video Stabilization



Given two input images  $g_a$  and  $g_b$ :

Apply a [window function](#) (e.g., a [Hamming window](#)) on both images to reduce edge effects (this may be optional depending on the image characteristics). Then, calculate the discrete 2D [Fourier transform](#) of both images.

$$\mathbf{G}_a = \mathcal{F}\{g_a\}, \mathbf{G}_b = \mathcal{F}\{g_b\}$$

Calculate the [cross-power spectrum](#) by taking the [complex conjugate](#) of the second result, multiplying the [Fourier transforms](#) together elementwise, and normalizing this product elementwise.

$$R = \frac{\mathbf{G}_a \circ \mathbf{G}_b^*}{|\mathbf{G}_a \circ \mathbf{G}_b^*|}$$

Where  $\circ$  is the [Hadamard product](#) (entry-wise product) and the absolute values are taken entry-wise as well. Written out entry-wise for element index  $(j, k)$ :

$$R_{jk} = \frac{G_{a,jk} \cdot G_{b,jk}^*}{|G_{a,jk} \cdot G_{b,jk}^*|}$$

Obtain the normalized cross-correlation by applying the inverse Fourier transform.

$$\mathbf{r} = \mathcal{F}^{-1}\{R\}$$

Determine the location of the peak in  $\mathbf{r}$ .

$$(\Delta x, \Delta y) = \arg \max_{(x,y)} \{r\}$$

# Image Registration using Phase Correlation

- 1- Select two image
- 2- Apply windowing to each image
- 3- Compute FFTs of Im1 and Im2
- 4- Obtain Phase Correlation Matrix
- 5- Find the Location of Max. Point
- 6- Stabilize the selected image Im2

```
% PC  
Ga = fft2(win.*I1);  
Gb = fft2(win.*I2);  
R = ifft2(Ga.*conj(Gb) ./ abs(Ga.*conj(Gb)));
```

$$R_{jk} = \frac{G_{a,jk} \cdot G_{b,jk}^*}{|G_{a,jk} \cdot G_{b,jk}^*|}$$

Obtain the normalized cross-correlation by applying the inverse Fourier transform.

$$r = \mathcal{F}^{-1}\{R\}$$

# First Try in a Shifted Image

```
clc; clear all;
I1 = double(imread ('cameraman.tif'));
I2 = I1(5:end,6:end);
I1 = I1(1:end-4,1:end-5);

figure(1); imshow(I1, []);
figure(2); imshow(I2, []);

%% FFT
win =
hamming(size(I1,1))*hamming(size(I1,2))';
Ga = fft2(win.*I1);
Gb = fft2(win.*I2);
R = ifft2(Ga.*conj(Gb)./abs(Ga.*conj(Gb)));
[dX, dY] = find(R==max(max(R)));

%% Crop Im.
I2 = cropIm(I2,dX,dY);
figure(3); imshow(I2, []);
```

I1



I2



I2Corr



# Video Stabilization



- 1- Select two sequential image
- 2- Apply windowing to each image
- 3- Compute FFTs of Im1 and Im2
- 4- Obtain Phase Correlation Matrix
- 5- Find the Location of Max. Point
- 6- Stabilize the selected image Im2

```
%% Read video frames
k=1;
v = VideoReader('18AF.avi');
vidHeight = v.Height;
vidWidth = v.Width;
s = struct('cdata',zeros(vidHeight,vidWidth,'uint8'),
'colormap',[]);

while hasFrame(v)
    s(k).cdata = readFrame(v);
    k = k+1;
end
```

```
I2rgb = s(k).cdata;
```

