IN THE NAME OF ALLAH

Fast Fourier Transform

Algorithm design and analysis

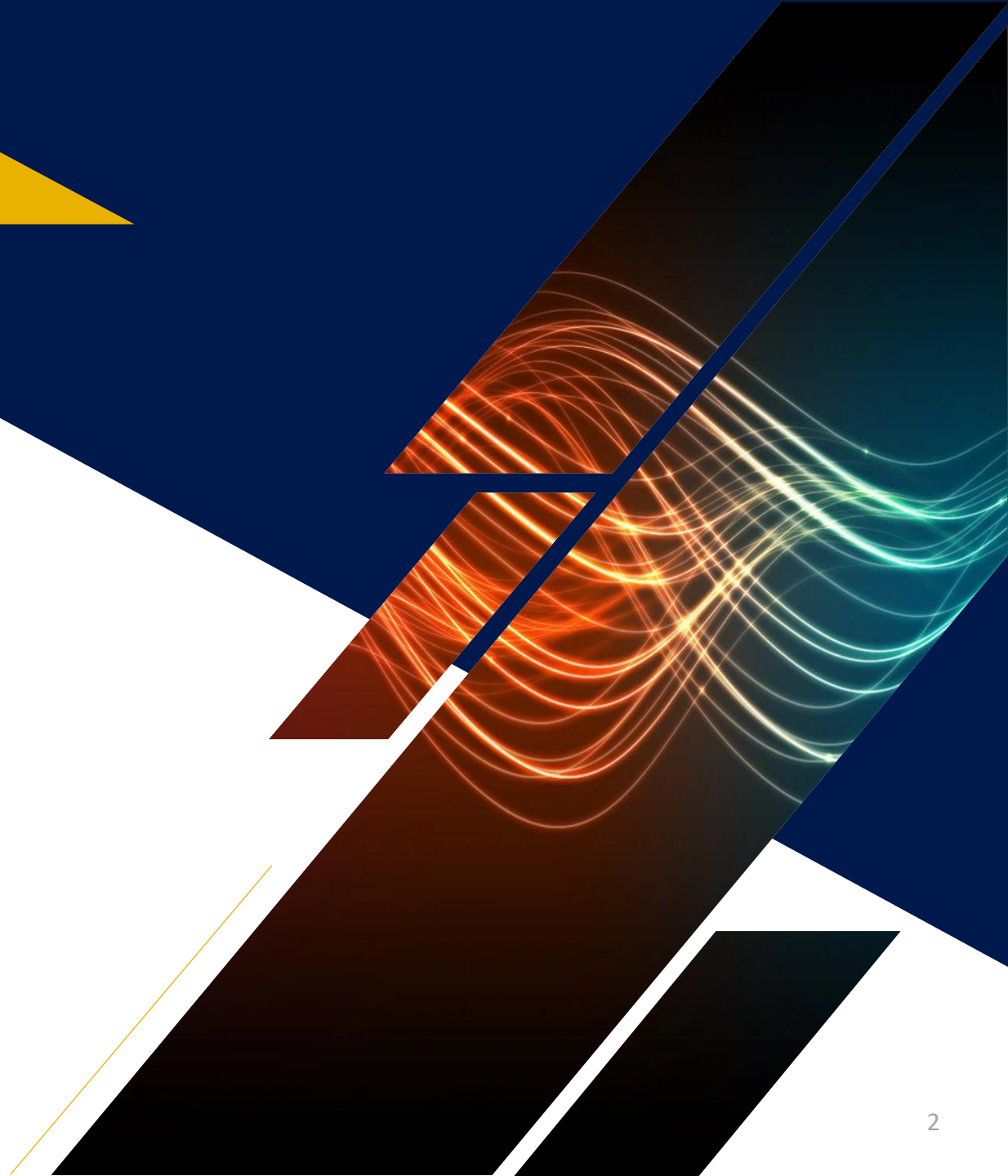
Seyed Mahdi Mahdavi Mortazavi

Seyed Mohsen Razavi Zadegan Jahromi

Contents

Part 1 – Signal analysis

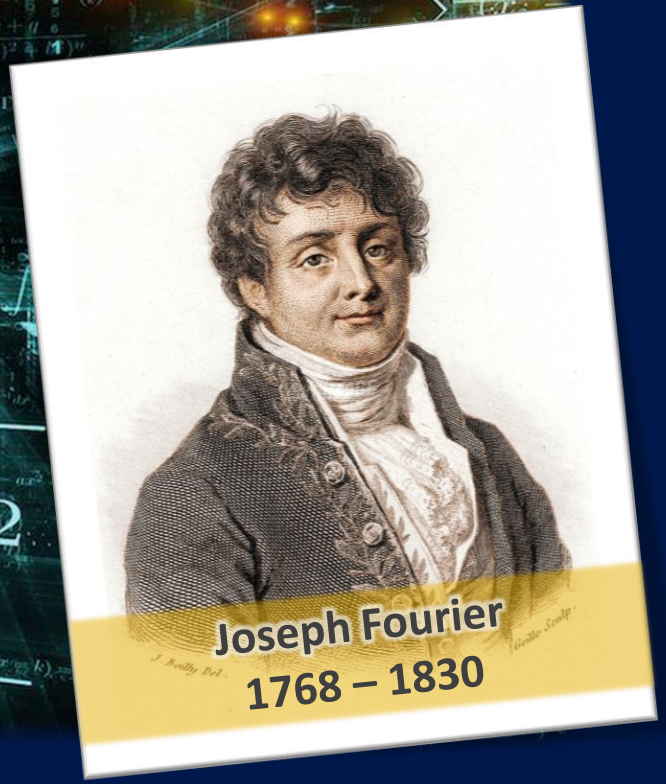
- Discrete signals
- Fourier Transform (FT)
- Discrete Fourier Transform (DFT)
- Applications of Fourier Transform (Frequency modulation (FM) and noise reduction (filtering))



Contents

Part 2 – Algorithm analysis

Part 1 – Signal analysis



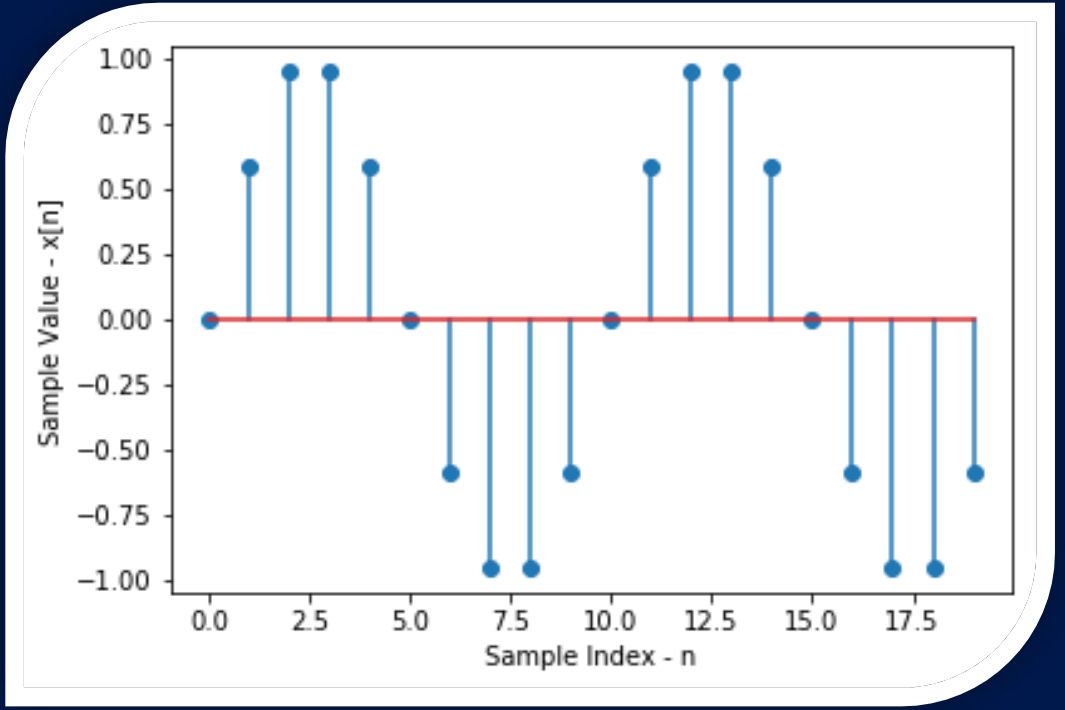
Joseph Fourier
1768 – 1830

Discrete signal

- A discrete signal is a signal (function) that does not have a continuous behavior for all values on the time axis. Actually, the discrete signal is only limited in time (as independent variable), but not limited in value (as dependent variable).
- A special form of this signal is the digital signal that we deal with in computers.

Actually, a discrete signal is only limited in time (as independent variable), but not limited in value (as dependent variable).

$$n \rightarrow \textbf{Independent variable}$$
$$\mathbf{X}[n] \rightarrow \textbf{Independent variable}$$

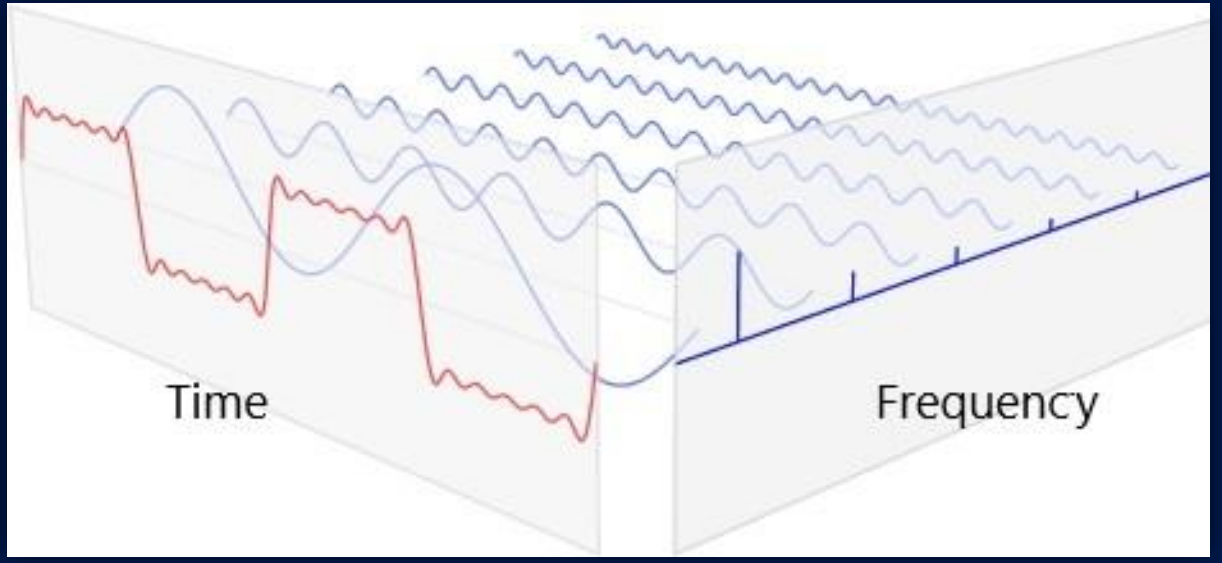


Fourier Transform (FT)

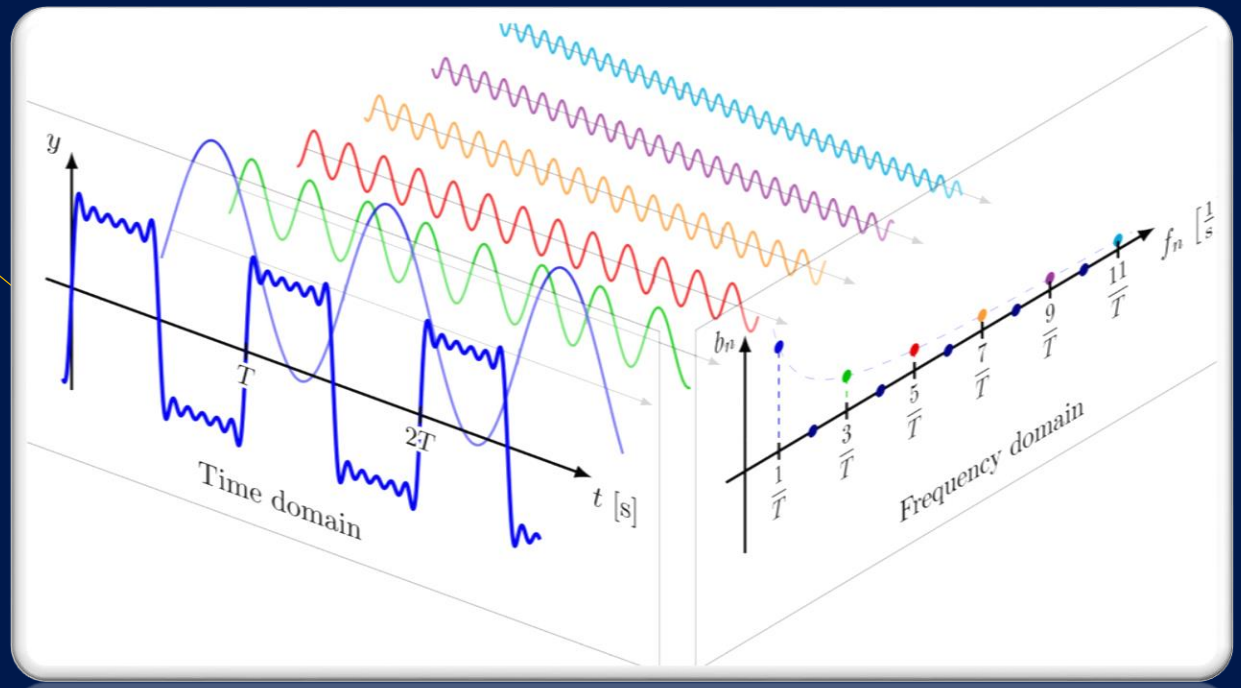
At first, let's check the concept of Fourier Transform:

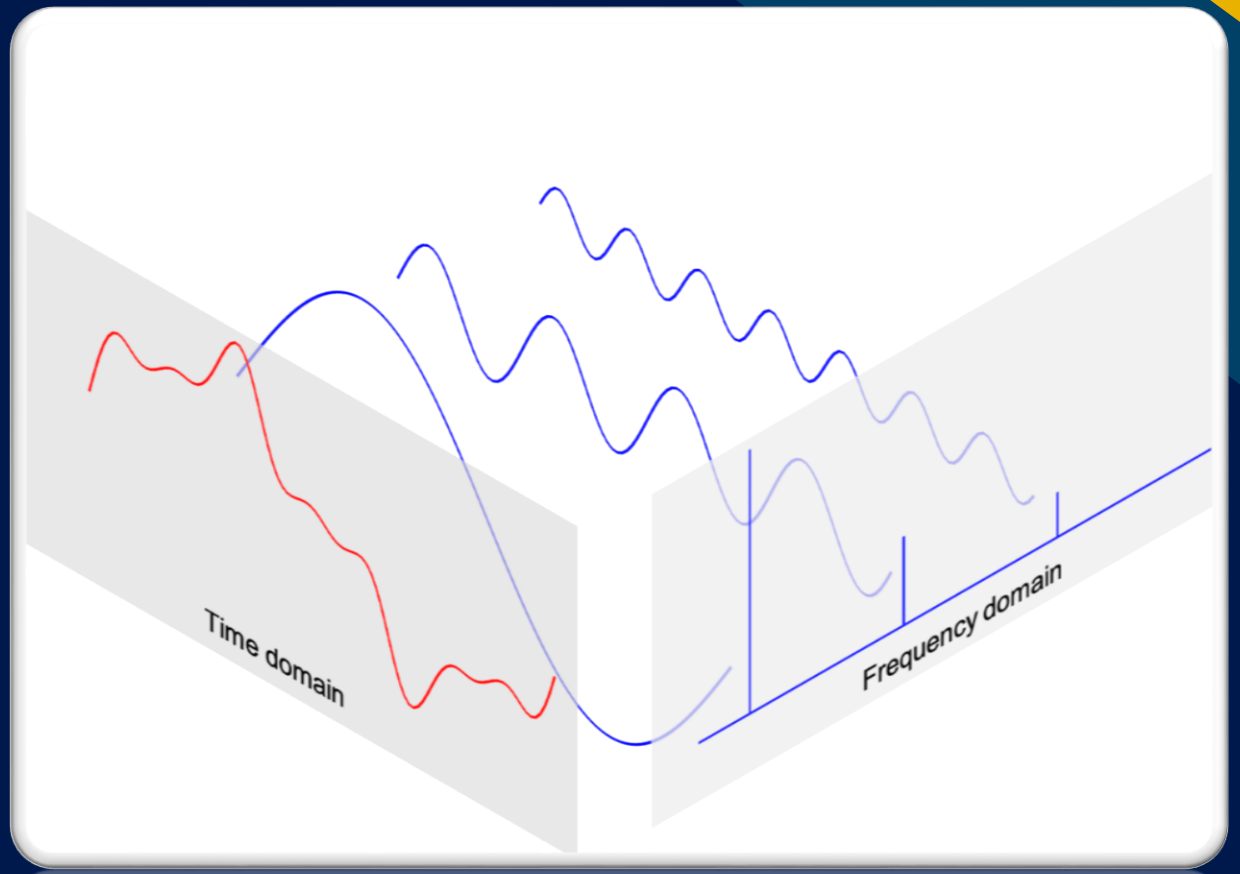
- **Fourier transform** is usually used to convert a signal in the time spectrum to a signal in the frequency spectrum. Examples of waves in the time spectrum are audio signals, electrical signals, and mechanical vibrations.
- The Fourier transform actually uses the principle that **any nonlinear function can be represented by an infinite sum of sine waves.** The image below shows the concept of displaying a signal with the sum of sinusoidal signals.

The Fourier transform decomposes a time-domain signal and provides information about the frequencies of all the sine waves needed to construct the time-domain signal.



Fourier Transform (other examples)





Discrete Fourier Transform (**DFT**)

An important concept: Differences between Discrete-time Fourier Transform (DTFT) and Discrete Fourier Transform (DFT):

- The **Discrete-time Fourier Transform (DTFT)** is the (conventional) Fourier transform of a discrete-time signal. Its output is continuous in frequency and periodic; For example: to find the spectrum of the sampled version $x(j\omega)$ of a continuous-time signal $x(t)$ or $x(e^{j\omega})$ of a discrete-time signal of $x[n]$.
- The **Discrete Fourier Transform (DFT)** can be seen as the sampled version (in frequency-domain) of the DTFT output. It's used to calculate the frequency spectrum of a discrete-time signal with a computer, because computers can only handle a finite number of values. It is periodic as well and can therefore be continued infinitely.

Method	Input	Output
DTFT	Discrete, Infinite	Continuous, Periodic
DFT	Discrete, Finite	Discrete, Finite

DFT vs **DTFT** (Formula)

Differences between the formula of DTFT and DFT (mathematical review):

DTFT (and IDTFT) Formula

DTFT :

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

Inverse of DTFT :

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

DFT (and IDFT) Formula

DFT :

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N}$$

Inverse DFT (IDFT) :

$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j2\pi nk/N}$$

In one period we have :

$$\hat{x}[n] \triangleq \begin{cases} x[n] & \text{for } 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases}$$

Applications of Fourier Transform

Perhaps the most important reason for using the Fourier transform is to reduce the Noise (denoising).

- In signal processing, **Noise** is a general term for unwanted (in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing, or conversion.
- Two of the best ways to remove signal noise (denoising) are **Modulation** and **Filtering**.

Modulation is the encoding of information of a signal in a carrier wave by varying (usually increasing) one of the characteristics of our signal wave;

- **Wave characteristics**: Amplitude, Frequency, Wavelength, Time Period and Speed; Speed of a ways = Frequency × Wavelength ($v = f.\lambda$).
- One of the best modulations is **Frequency Modulation (FM)**; As you know, most music is played through FM (Frequency Modulation) radio in which we call **FM wave**.

Advantages of **Frequency Modulation** (FM)

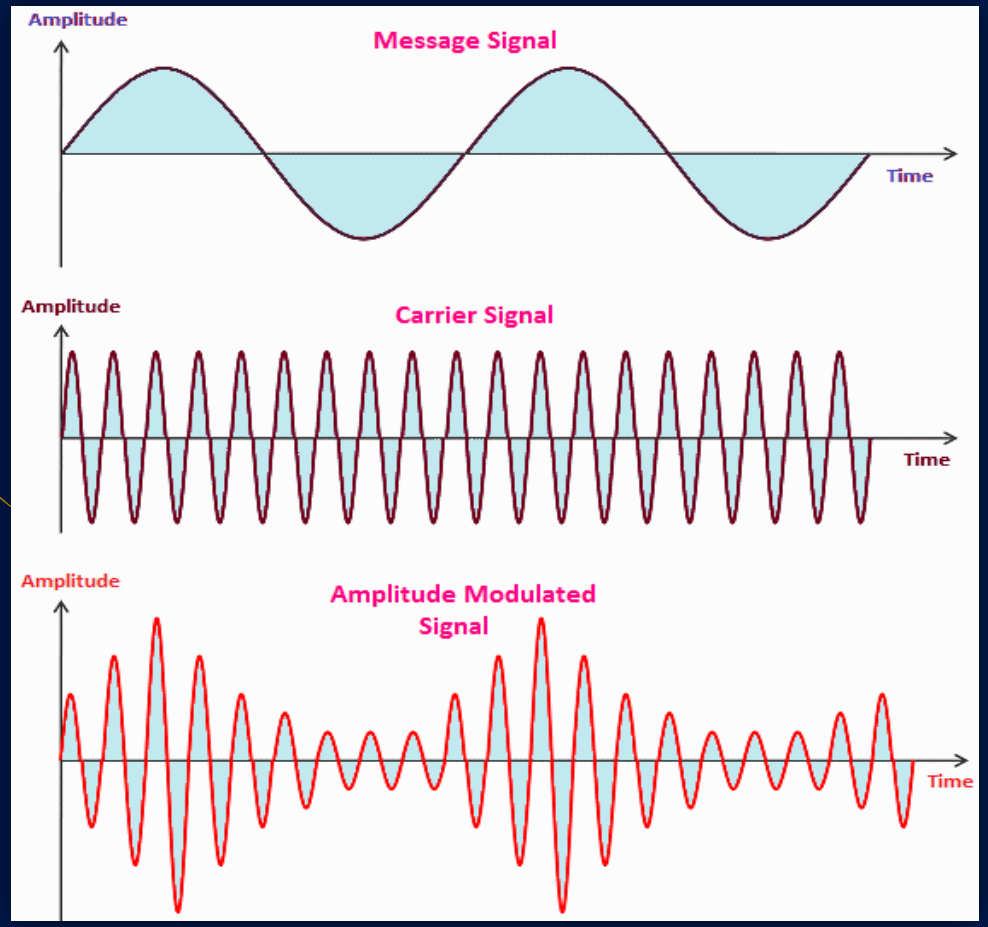
Frequency modulation (FM) is the encoding of information in a carrier wave by varying (usually increasing) the frequency of the wave.

- In radio transmission, frequency modulation has a good advantage over other modulation; Actually, Frequency Modulation decreases the noise; hence, there is a significant increase in the **signal-to-noise ratio (SNR)*;** Meaning it will reject radio frequency interferences much better than other modulations.
- It also reduces the interference by the adjacent channels through guard bands.
- Due to this major reasons, most music is played through **FM (Frequency Modulation)** radio (compared to other modulations: **Amplitude modulation (AM)** and **Phase modulation (PM)**).

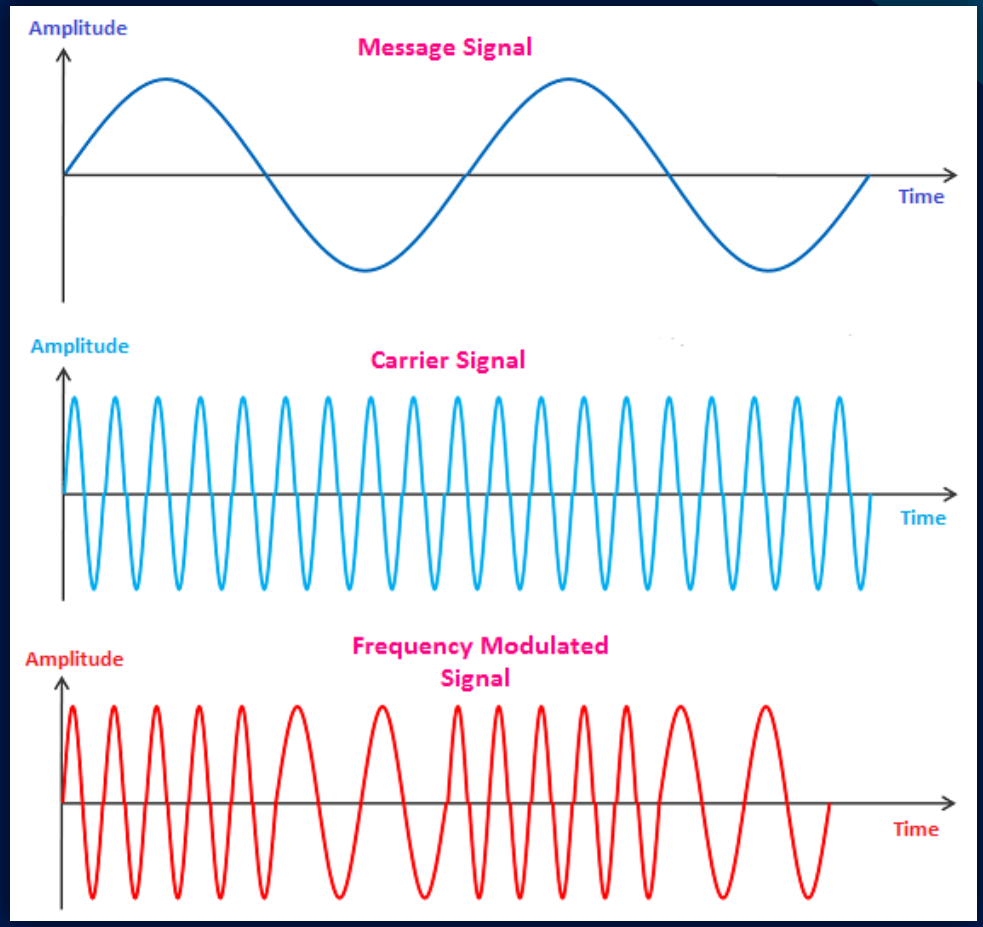
$$* SNR = \frac{Number\ of\ Signal\ points}{Number\ of\ Noisy\ points}$$

Two Types of **Modulation**

Amplitude Modulation (AM)



Frequency Modulation (FM)



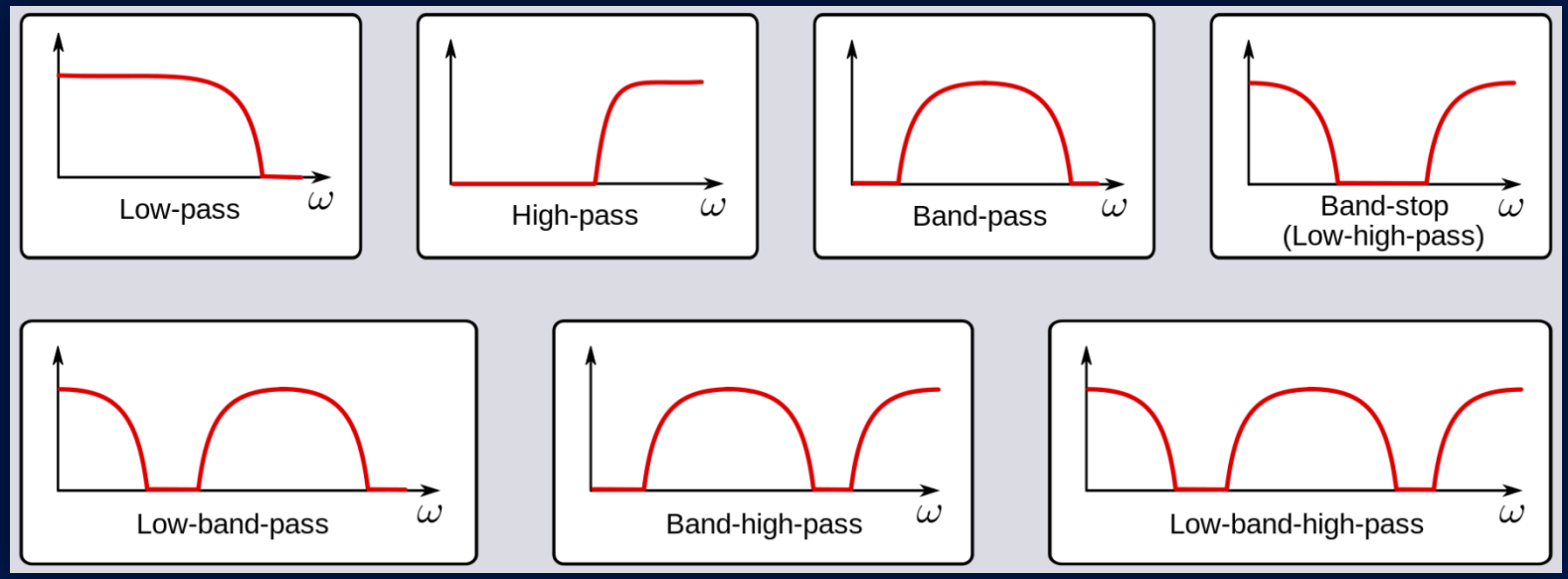
Noise Filtering

Review of different noise reduction (filtering) methods; Low, Band and High pass filters. In signal processing, noise reduction can be achieved in both the time domain as well as frequency domain; In case of the latter, Fourier Transform or Wavelet Transform of the observed signal is obtained and subsequently an appropriate filter is applied. A way to denoising a signal is removing the noised points in frequency domain; How? With using low, band and high pass filters with an index frequency and reject lower or higher frequencies than index frequency (or reject signals between two index frequencies).

Charts are in the frequency domain (continuous); ω is the Angular frequency.

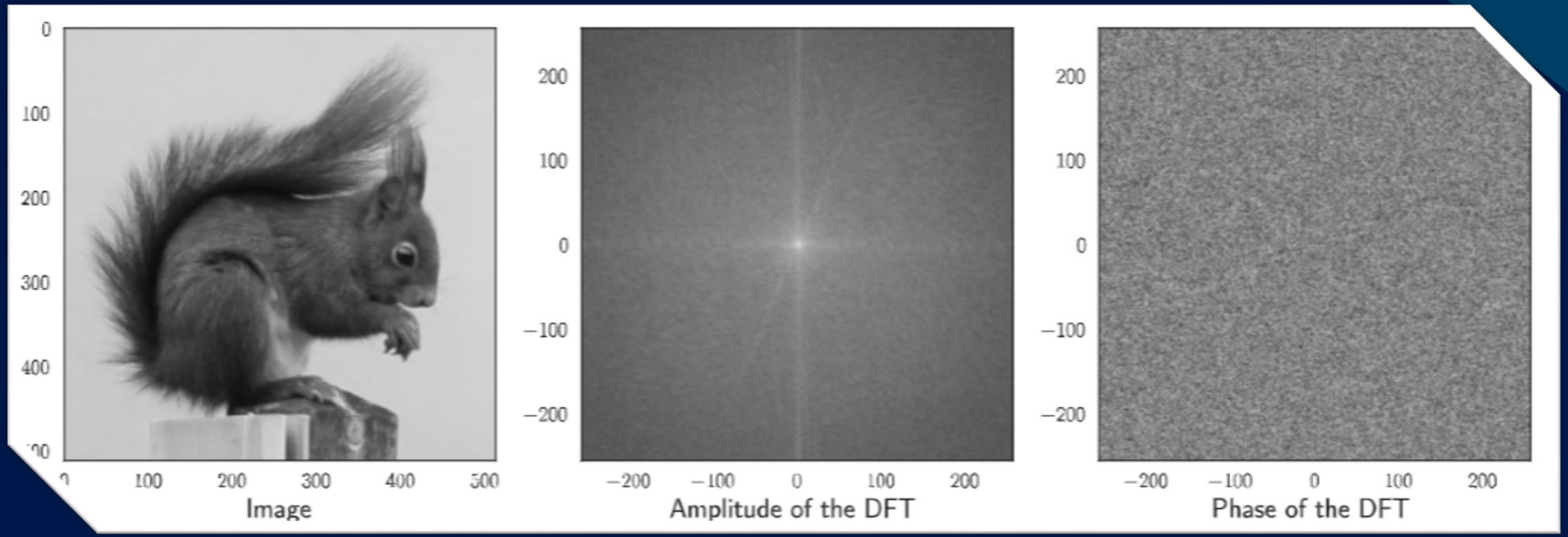
$$\omega = \frac{2\pi}{T} = 2\pi f$$

Where T is Period; And f is frequency.



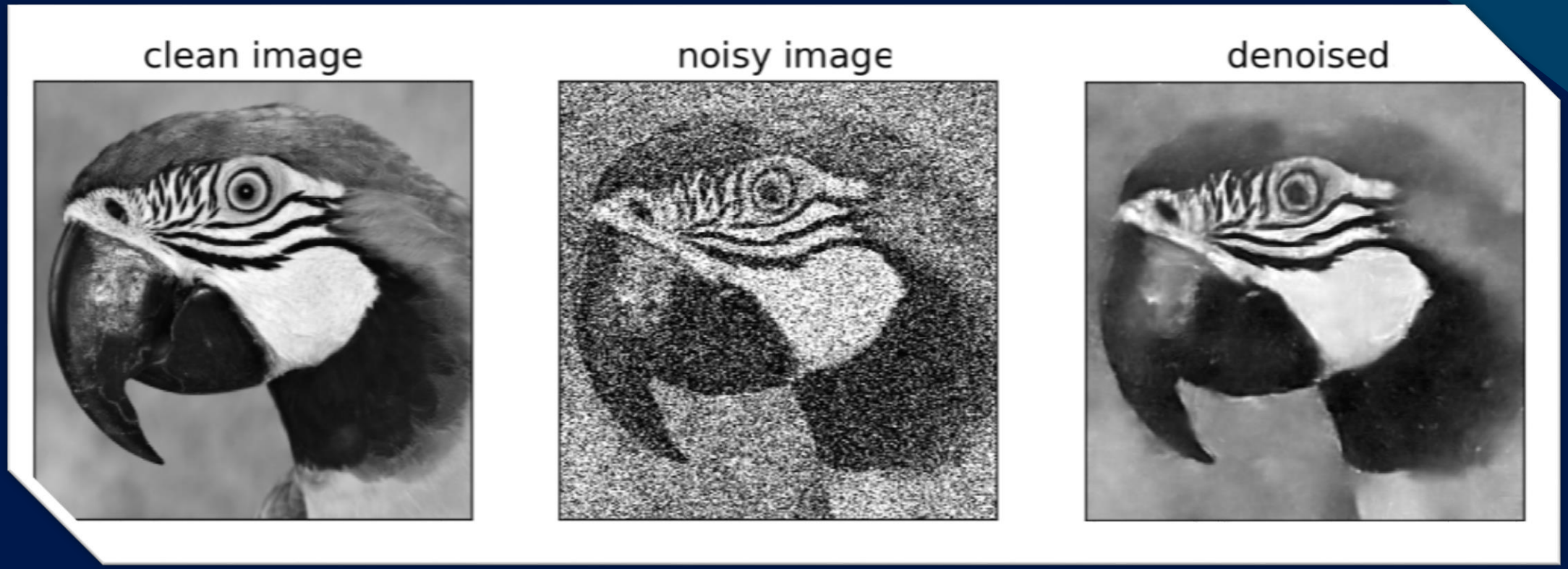
Review some **real examples #1**

- A picture with its Fourier transform (using DFT) signal (Amplitude and Phase)



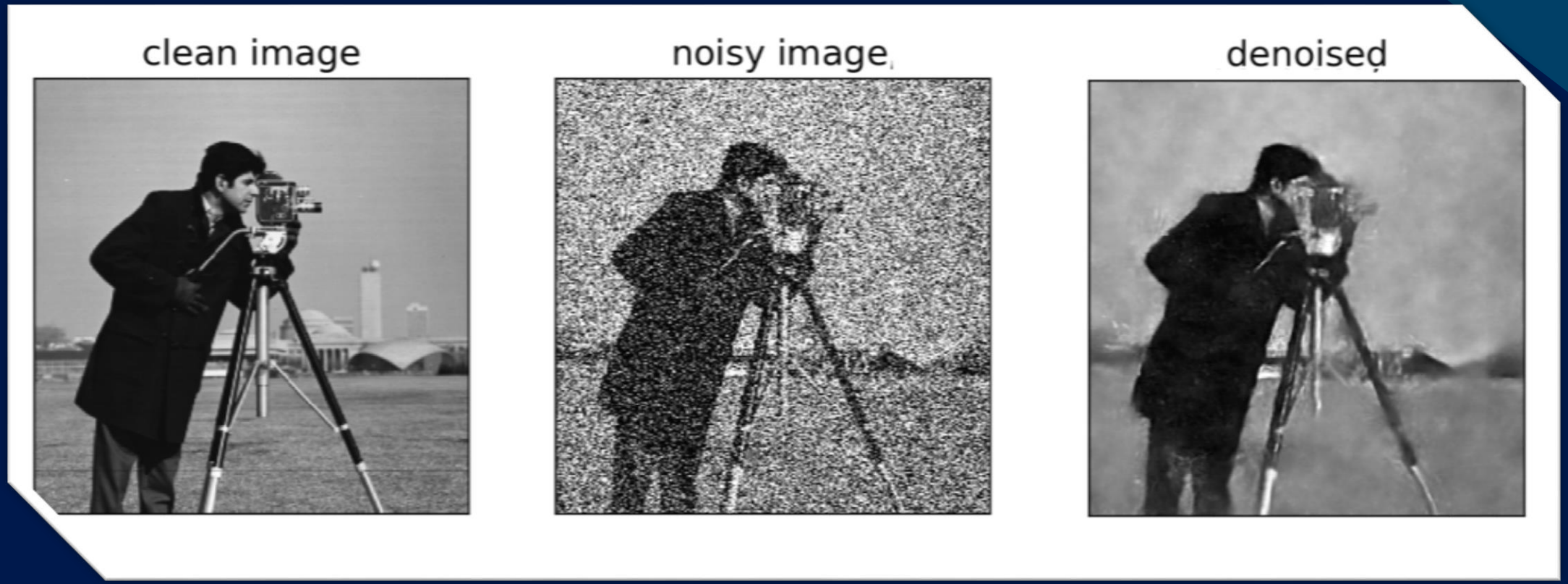
Review some **real examples #2**

- A picture in different states (Clean, Noisy, Denoised) #1

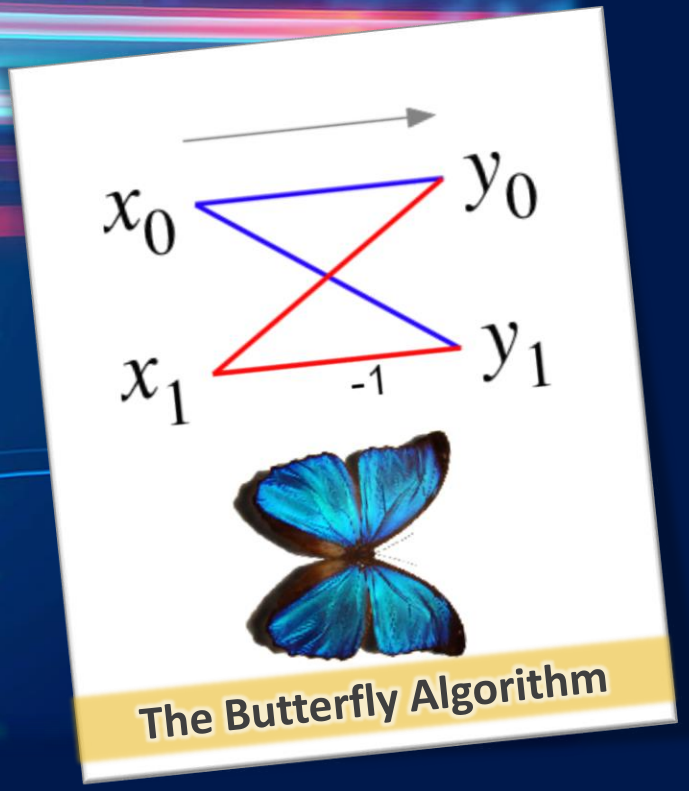


Review some **real examples #3**

- A picture in different states (Clean, Noisy, Denoised) #2



Part 2 – Algorithm analysis



Disadvantages of DFT

Let's review the Discrete Fourier Transform (DFT) formula. As you can see, it has very high time complexity (for many points) and complex calculations.

Discrete Fourier transform (DFT) implementation requires high computational resources and time; A computational complexity of order $O(N^2)$ for a signal of size N. This time complexity is very high for a large number of points.

DFT :

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}$$

Inverse DFT (IDFT) :

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j2\pi nk/N}$$

In one period we have :

$$\hat{x}[n] \triangleq \begin{cases} x[n] & \text{for } 0 \leq n \leq N-1 \\ 0 & \text{otherwise} \end{cases}$$

Complexity of DFT

These transformations (with DFT and IDFT) are quite time-consuming and to understand this, we will describe DFT formula in more detail:

$$X(0)=x(0)+x(1)+\ldots+x(N-1)$$

$$X(1)=x(0)+x(1)e^{-j\frac{2\pi}{N}}+\ldots+x(N-1)e^{-j\frac{2\pi(N-1)}{N}}$$

$$\ldots$$

$$X(N-1)=x(0)+x(1)e^{-j\frac{2\pi(N-1)}{N}}+\ldots+x(N-1)e^{-j\frac{2\pi(N-1)(N-1)}{N}}$$

DFT :

$$X[k]=\sum_{n=0}^{N-1}x[n]e^{-j2\pi nk/N}$$

Inverse DFT (IDFT) :

$$x[n]=\frac{1}{N}\sum_{k=0}^{N-1}X[k]e^{j2\pi nk/N}$$

In one period we have :

$$\hat{x}[n]\triangleq\begin{cases}x[n] & \text{for } 0\le n\le N-1\\ 0 & \text{otherwise}\end{cases}$$

$$\begin{cases}\boldsymbol{n: 0\rightarrow (N-1)}\\ \boldsymbol{k: 0\rightarrow (N-1)}\end{cases}\rightarrow \boldsymbol{Order\ O(N^2)}$$

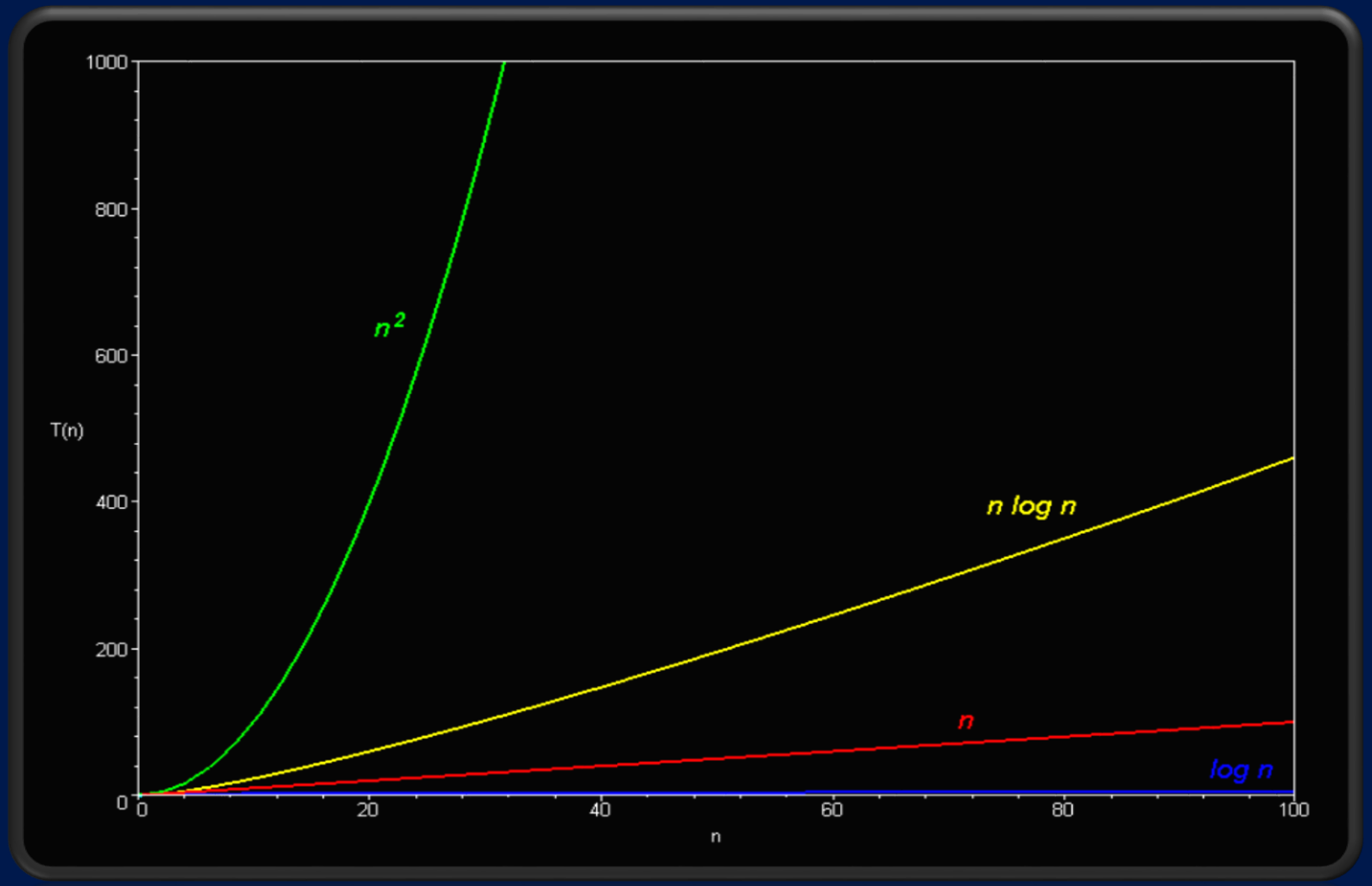
Fast Fourier Transform (FFT)

Often in **Digital Signal Processing** applications it is necessary to estimate the signal spectrum, or vice versa, knowing the signal spectrum calculate the signal itself. For this purpose digital technology uses the Discrete Fourier Transform (DFT) and Inverse Discrete Fourier Transform (IDFT), which are described by formulas of DFT and IDFT.

- **Fast Fourier Transform** (**FFT**) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence, or its inverse (IDFT);
- An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from $\boldsymbol{O(n^2)}$ **to** $\boldsymbol{O(nlogn)}$ which arises if we simplify the definition of DFT, where $\boldsymbol{n}$ is the data size.
- The difference in speed can be enormous, especially for long data sets where $\boldsymbol{n}$ may be in the thousands or millions. many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly.
- There are many different FFT algorithms based on a wide range of published theories; **Here we review one of them that named Butterfly Algorithm (using Divide & Conquer).**

Comparison of **Complexities**

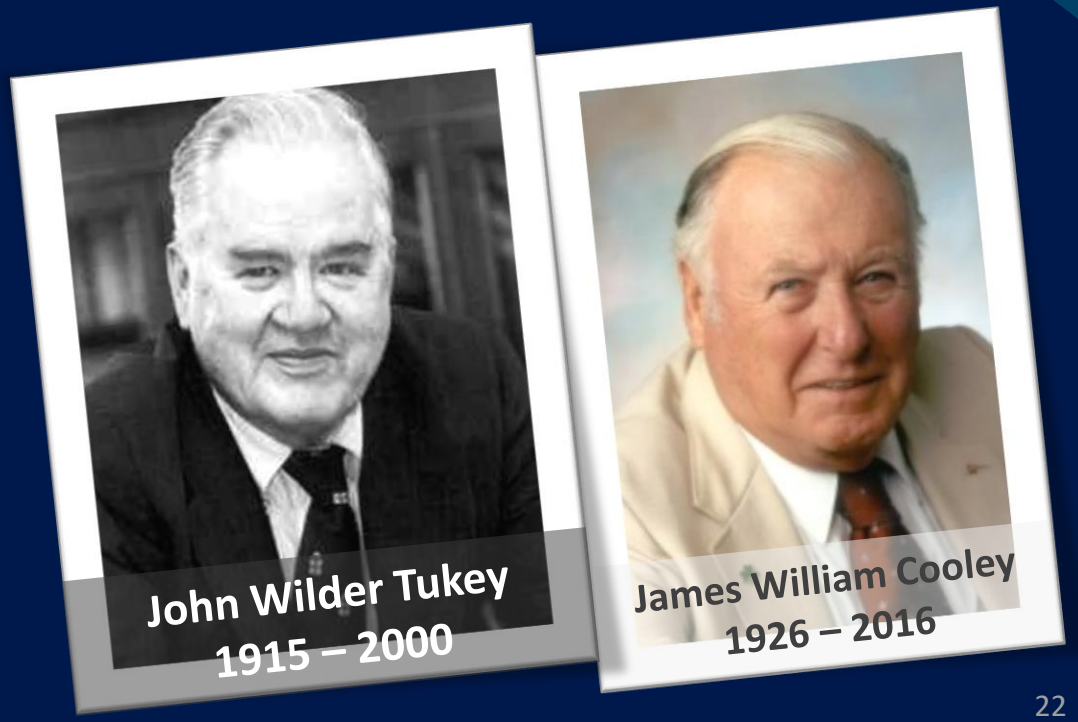
- Comparison between $O(N^2)$ and $O(NlogN)$



Butterfly algorithm

In the context of **fast Fourier transform algorithms**, a **butterfly** is a portion of the computation that combines the results of smaller discrete Fourier transforms (DFTs) into a larger DFT, or vice versa (breaking a larger DFT up into subtransforms).

- The name **"butterfly"** comes from the shape of the data-flow diagram in the radix-2 case, as described in the first slide of Part 2. The earliest occurrence in print of the term is thought to be in a 1969 MIT technical report.
- Most commonly, the term "butterfly" appears in the context of the Cooley–Tukey FFT algorithm, which recursively breaks down a DFT of composite size $n = r \times m$ into r smaller transforms of size m where r is the "radix" of the transform. These smaller DFTs are then combined via $size - r$ and m butterflies.
- The **Cooley-Tukey** algorithm is named after **James William Cooley** and **John Tukey**.[1][2]



Review an **example**

Consider this example:

- We have **8 points** ($N = 8$); So the DFT formula becomes like this:

$$X[k] = \sum_{n=0}^{7} \left(x[n] e^{-\frac{j2\pi nk}{8}} \right) = x[0] e^{-\frac{j2\pi k}{8} \times 0} + x[1] e^{-\frac{j2\pi k}{8} \times 1} + x[2] e^{-\frac{j2\pi k}{8} \times 2}$$

$$+ x[3] e^{-\frac{j2\pi k}{8} \times 3} + x[4] e^{-\frac{j2\pi k}{8} \times 4} + x[5] e^{-\frac{j2\pi k}{8} \times 5} + x[6] e^{-\frac{j2\pi k}{8} \times 6} + x[7] e^{-\frac{j2\pi k}{8} \times 7}$$

- Lets separate **Even** points and **Odd** points:

$$\boldsymbol{Even\ points}: \boldsymbol{G[k]} = x[0] e^{-\frac{j2\pi k}{8} \times 0} + x[2] e^{-\frac{j2\pi k}{8} \times 2} + x[4] e^{-\frac{j2\pi k}{8} \times 4} + x[6] e^{-\frac{j2\pi k}{8} \times 6}$$

$$\boldsymbol{Odd\ points}: \boldsymbol{H'[k]} = x[1] e^{-\frac{j2\pi k}{8} \times 1} + x[3] e^{-\frac{j2\pi k}{8} \times 3} + x[5] e^{-\frac{j2\pi k}{8} \times 5} + x[7] e^{-\frac{j2\pi k}{8} \times 7}$$

Review an **example**

- Simplification for **Even** points:

$$\boldsymbol{Even\ points}:\ \boldsymbol{G[k]} = x[0]e^{-\frac{j2\pi k}{8}\times 0} + x[2]e^{-\frac{j2\pi k}{8}\times 2} + x[4]e^{-\frac{j2\pi k}{8}\times 4} + x[6]e^{-\frac{j2\pi k}{8}\times 6} \rightarrow$$

$$\boldsymbol{G[k]} = x[0] + x[2]e^{-\frac{j2\pi k}{4}} + x[4]e^{-\frac{j2\pi k}{4}\times 2} + x[6]e^{-\frac{j2\pi k}{4}\times 3}$$

- Simplification for **Odd** points:

$$\boldsymbol{Odd\ points}:\boldsymbol{H'[k]} = x[1]e^{-\frac{j2\pi k}{8}\times 1} + x[3]e^{-\frac{j2\pi k}{8}\times 3} + x[5]e^{-\frac{j2\pi k}{8}\times 5} + x[7]e^{-\frac{j2\pi k}{8}\times 7} \rightarrow$$

$$\boldsymbol{H'[k]} = e^{-\frac{j2\pi k}{8}} \times \left(x[1] + x[3]e^{-\frac{j2\pi k}{8}\times 2} + x[5]e^{-\frac{j2\pi k}{8}\times 4} + x[7]e^{-\frac{j2\pi k}{8}\times 6}\right) \rightarrow$$

$$\boldsymbol{H'[k]} = e^{-\frac{j2\pi k}{8}} \times (\underbrace{x[1] + x[3]e^{-\frac{j2\pi k}{4}} + x[5]e^{-\frac{j2\pi k}{4}\times 2} + x[7]e^{-\frac{j2\pi k}{4}\times 3}}_{\boldsymbol{H[k]}})$$

Review an **example**

- Combining the **Even** and **Odd** points:

$$X[k] = G[k] + H'[k] = G[k] + e^{-\frac{j2\pi k}{8}} H[k]$$

- $G[k]$ and $H[k]$ are periodic with period $(N = 4)$; So lets calculate $G[k+4]$ and $H[k+4]$. To do this we should calculate $X[k+4]$:

$$X[k+4] = G[k+4] + e^{-\frac{j2\pi(k+4)}{8}} H[k+4]$$

$$\begin{cases} G[k] = G[k+4] \\ H[k] = H[k+4] \\ * \; e^{-\frac{j2\pi(k+4)}{8}} = -e^{-\frac{j2\pi k}{8}} \end{cases}$$

$$* \; e^{-\frac{j2\pi(k+4)}{8}} = e^{-\frac{j2\pi k}{8}} . e^{-\frac{j2\pi \times 4}{8}} = e^{-\frac{j2\pi k}{8}} . \underbrace{e^{-j\pi}}_{-1} = -e^{-\frac{j2\pi k}{8}}$$

Example to **Proof**

- So we will have:

$$X[k] = G[k] + e^{-\frac{j2\pi k}{8}} H[k]$$

$$X[k+4] = G[k] - e^{-\frac{j2\pi k}{8}} H[k]$$

- Finally we can using this conjunction, to decrease the complexity of the DFT. Generally we have this (if we have N points):

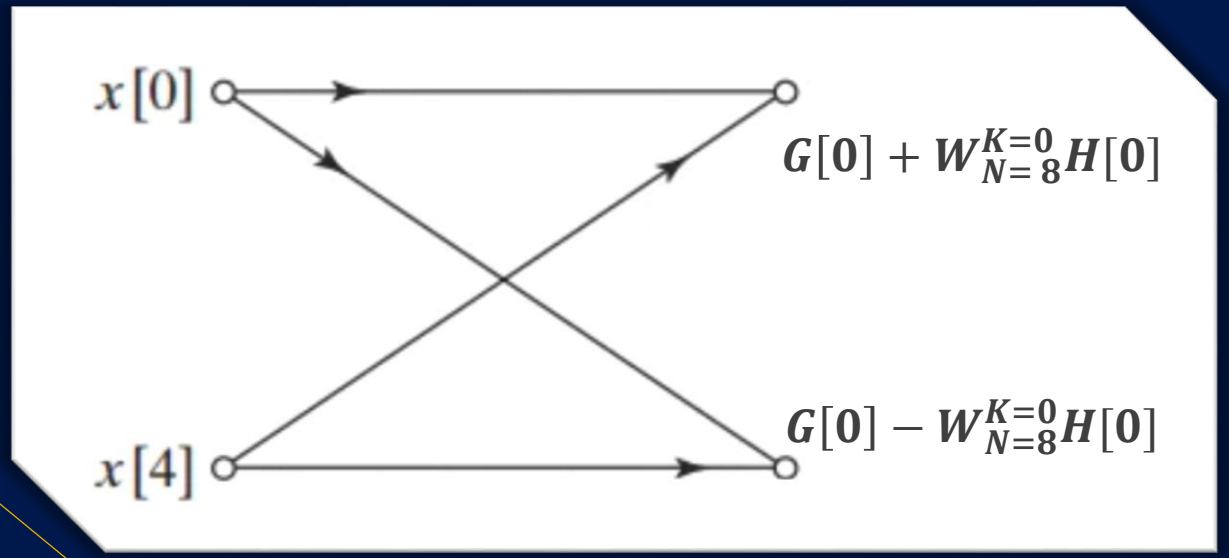
$$X[k] = G[k] + e^{-\frac{j2\pi k}{N}} H[k]$$

$$X[k + N/2] = G[k] - e^{-\frac{j2\pi k}{N}} H[k]$$

❖ **Note: N must be a power of 2; Why? Refer to Slide 22;**

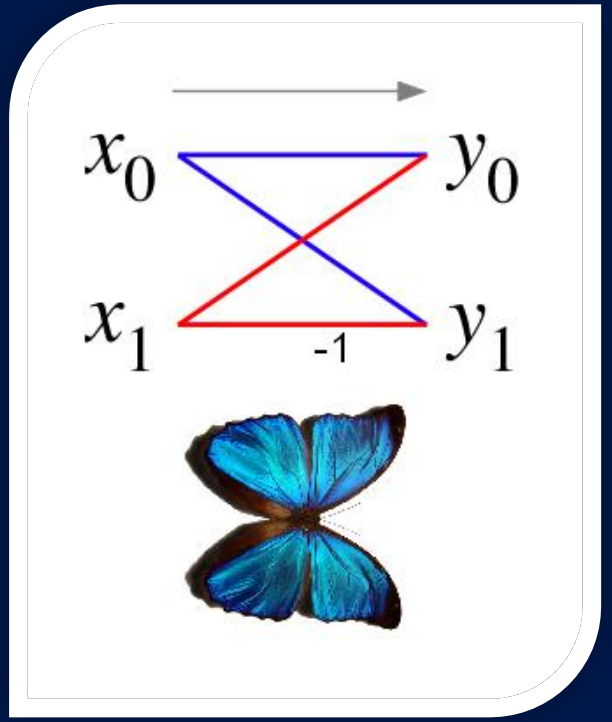
Butterfly algorithm **diagram**

- Suppose that $e^{-\frac{j2\pi k}{N}} = W_N^k$, then we have this diagram for $N = 8$:



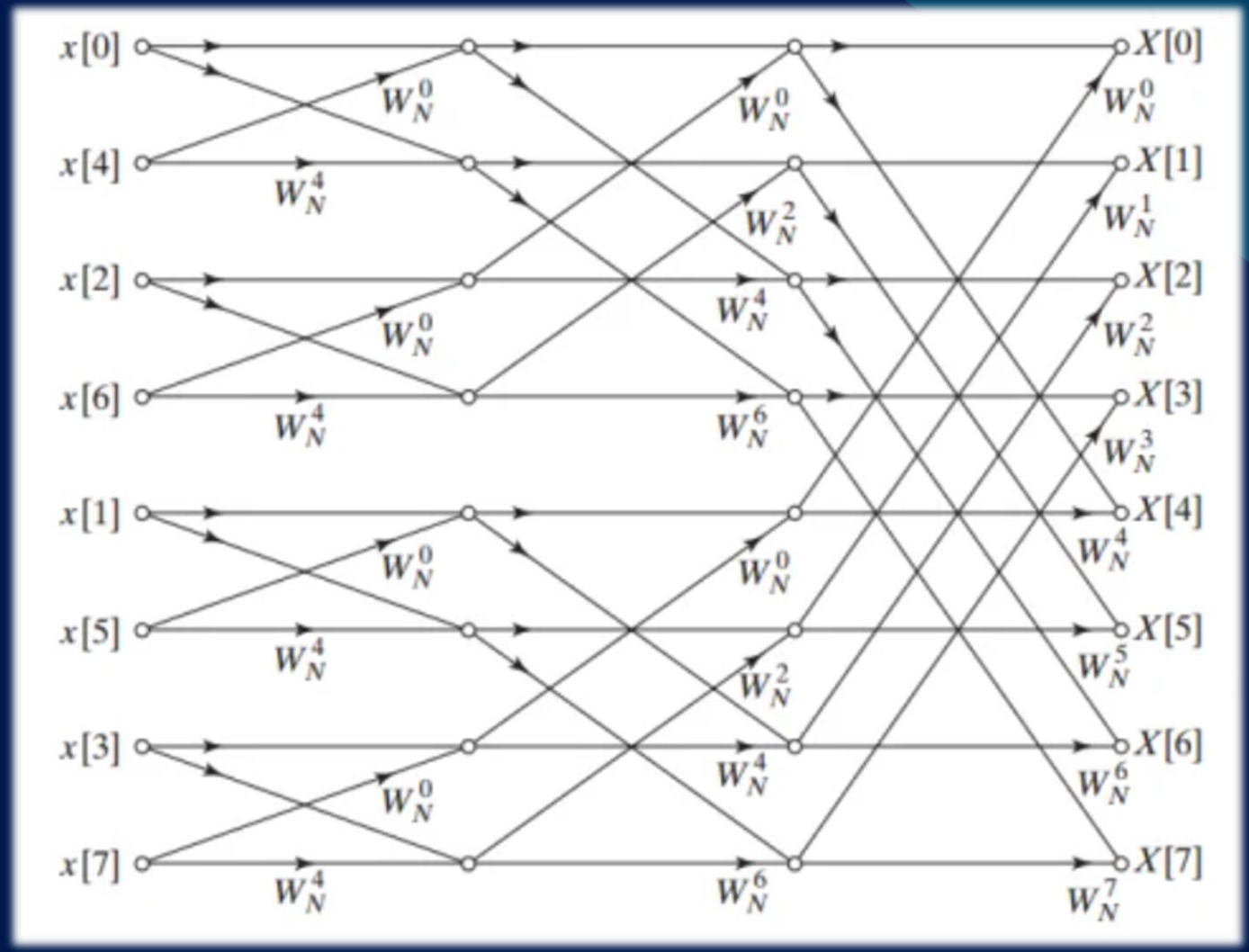
$$k = 0 \begin{cases} X[0] = G[0] + e^{-\frac{j2\pi(0)}{N}} H[0] \\ X\left[0 + (\frac{8}{2} = 4)\right] = G[0] - e^{-\frac{j2\pi(0)}{N}} H[0] \end{cases}$$

Did you see the Butterfly?!



Butterfly algorithm **diagram**

- $e^{-\frac{j2\pi k}{N}} = W_N^k$
- $N = 8$



Code review #1

DFT

Complexity is order

$$O(N^2)$$

DFT : $$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N}$$

```python
from cmath import pi, exp

def exponential(N, k):
  return exp((-2.0 * pi * 1j * k) / N)


def dft(signal):
    N = len(signal)
    ans = [0 for _ in range(N)]
    for k in range(N):
      temp = 0
      for n in range(N):
        temp += signal[n] * exponential(N, k*n)
      ans[k] = temp

    return ans
```

```python
import cmath
from dft import dft

def exponential(N, k):
  return cmath.exp((-2.0 * cmath.pi * 1j * k) / N)

def fft(signal):
  N = len(signal)

  if N == 1:
     return signal

  elif N%2:
    return ValueError(f'Number of signal points must be power of 2.  {N})')
                      (
  elif N <= 32:
    return dft(signal)

  else:
    Feven = fft([signal[i] for i in range(0, N, 2)]) # Calculating for even points
    Fodd = fft([signal[i] for i in range(1, N, 2)]) # Calculating for odd points

    # Combining
    combined = [0 for _ in range(N)]
    for k in range(N // 2):
        exp = exponential(N, k)
        combined[k] = Feven[k] +  exp* Fodd[k]
        combined[k + N // 2] = Feven[k] -  exp* Fodd[k]
    return combined
```

FFT

Code review #2

FFT

$$X[k] = G[k] + e^{-\frac{j2\pi k}{N}} H[k]$$

$$X[k + N/2] = G[k] - e^{-\frac{j2\pi k}{N}} H[k]$$

$$T(N) = 2T\left(\frac{N}{2}\right) + O(N)$$

Using master theorem

$$T(N) = \theta(N log N)$$

```python
import cmath
from dft import dft

def exponential(N, k):
  return cmath.exp((-2.0 * cmath.pi * 1j * k) / N)

def fft(signal):
  N = len(signal)

  if N == 1:
     return signal

  elif N%2:
    return ValueError(f'Number of signal points must be power of 2.  {N})')
                      (
  elif N <= 32:
    return dft(signal)

  else:
    Feven = fft([signal[i] for i in range(0, N, 2)]) # Calculating for even points
    Fodd = fft([signal[i] for i in range(1, N, 2)]) # Calculating for odd points

    # Combining
    combined = [0 for _ in range(N)]
    for k in range(N // 2):
        exp = exponential(N, k)
        combined[k] = Feven[k] +  exp* Fodd[k]
        combined[k + N // 2] = Feven[k] -  exp* Fodd[k]
    return combined
```

Review some **examples**

- Runtime of different test cases in seconds.

Size of input	$N = 2^7$	$N = 2^{10}$	$N = 2^{15}$	$N = 2^{20}$
DFT runtime	0.0294456481	1.5533757209	1550.398316621 $\approx 26\ min$	$More\ than\ 5\ hours!$ …
FFT runtime	0.0135645866	0.0546631813	1.5639145374	52.2178380489
numpy.fft() runtime	0.0001180171	$4.3869018554e$ -04	0.0008692741	0.0465128421

- For each test case, answers are similar up to 10 decimal places.

References

The most used references in this article

- Faradars – FFT[1]
- Wikipedia – FFT[1], Butterfly Alg[2]
- Youtube – Most Important Algorithm of all time[1], MIT Course about Divide and Conquer for FFT[2]
- DSP-Weimich – Fast Fourier Transform and their C realizations using the octave gnu tool[1]

Thank You.

Seyed Mahdi Mahdavi Mortazavi

Seyed Mohsen Razavi Zadegan Jahromi

Github repos:

https://github.com/theMHD-120/fft

https://github.com/MohsenRazavi/fft