

# Fast Fourier Transform (FFT) and Short-Time Fourier Transform (STFT)

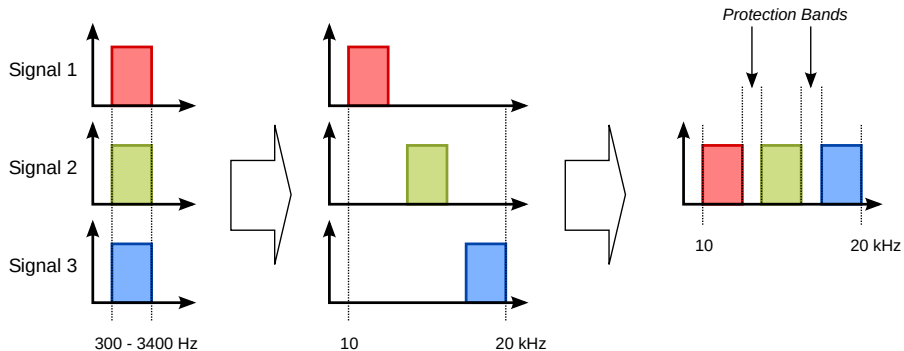
Digital Signal Processing

February 21, 2023



# Application of FFT: Communications

## Frequency Division Multiplexing (FDM)

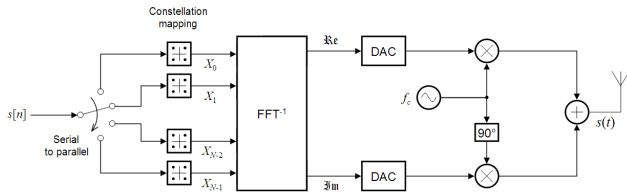


<https://commons.wikimedia.org/w/index.php?curid=18327256>

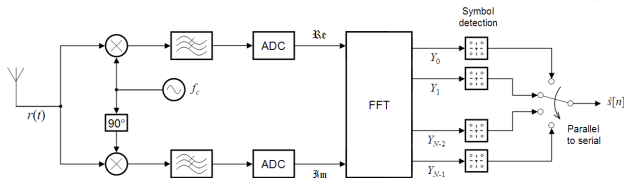
# Application of FFT: Communications

## Orthogonal Frequency Division Multiplexing (OFDM)

Transmitter:



Receiver:



Technology behind most digital wireless communication!  
(WiFi, 4G, 5G, HD Radio, HDTV)

# The Fast Fourier Transform (FFT)

# Naïve DFT Algorithm

Recall the DFT equation:

$$X[k] = \frac{1}{\sqrt{L}} \sum_{n=0}^{L-1} e^{-i\omega_0 n k} x[n]$$

## Naïve DFT Algorithm

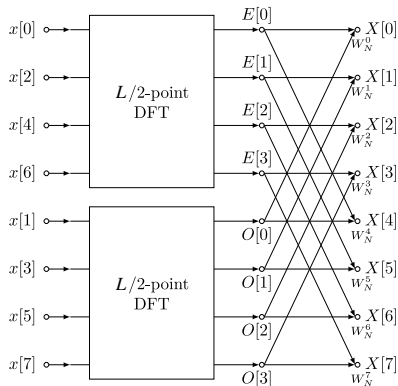
Loop over  $k = 0, \dots, L - 1$

    Compute  $X[k]$  by sum (loop) over  $n = 0, \dots, L - 1$

Complexity is  $O(L^2)$

# Fast Fourier Transform

## Cooley-Tukey Algorithm



## Divide-and-Conquer

### Radix-2 version:

- 1 compute “even” DFT
- 2 compute “odd” DFT
- 3 combine and reuse results

Recursively apply to each  $L/2$  block

Complexity is  $O(L \log L)$

# Fast Fourier Transform

- More general radix- $p$  FFT breaks the DFT into  $p$  blocks, where  $p$  is a prime factor of the signal length  $L$
- Recursively applied to each  $L/p$  block
- Recursion stops when the remaining block lengths are prime numbers (can't be factored any further)
- **Bottom line:** The FFT is most efficient when the input signal length has small prime factors, preferably  $L$  is a power of 2.
- Sometimes it is more efficient to pad a signal with zeros to get a good prime factorization.

# First Half of FFT

Compute  $X[k]$  for  $k = 0, 1, \dots, \frac{L}{2} - 1$ ,

$$\begin{aligned} X[k] &= \frac{1}{\sqrt{L}} \sum_{n=0}^{L-1} e^{-i\omega_0 n k} x[n] \\ &= \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2mk} x[2m] + \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 (2m+1)k} x[2m+1] \\ &= \underbrace{\frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2mk} x[2m]}_{E[k]=\text{sum of even terms}} + e^{-i\omega_0 k} \underbrace{\frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2mk} x[2m+1]}_{O[k]=\text{sum of odd terms}} \\ &= E[k] + e^{-i\omega_0 k} O[k] \end{aligned}$$



# Second Half of FFT

Compute  $X[k + \frac{L}{2}]$  for  $k = 0, 1, \dots, \frac{L}{2} - 1$ ,

$$\begin{aligned} X\left[k + \frac{L}{2}\right] &= \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2m(k + \frac{L}{2})} x[2m] + \\ &\quad + e^{-i\omega_0(k + \frac{L}{2})} \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2m(k + \frac{L}{2})} x[2m + 1] \\ &= E[k] - e^{-i\omega_0 k} O[k] \end{aligned}$$

Using  $e^{-i\omega_0 2m(k + \frac{L}{2})} = e^{-i\omega_0 2mk}$  and  $e^{-i\omega_0(k + \frac{L}{2})} = -e^{-i\omega_0 k}$

# Second Half of FFT

Compute  $X[k + \frac{L}{2}]$  for  $k = 0, 1, \dots, \frac{L}{2} - 1$ ,

$$\begin{aligned} X\left[k + \frac{L}{2}\right] &= \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2m(k + \frac{L}{2})} x[2m] + \\ &\quad + e^{-i\omega_0(k + \frac{L}{2})} \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2m(k + \frac{L}{2})} x[2m + 1] \\ &= E[k] - e^{-i\omega_0 k} O[k] \quad \text{Reused!} \end{aligned}$$

Using  $e^{-i\omega_0 2m(k + \frac{L}{2})} = e^{-i\omega_0 2mk}$  and  $e^{-i\omega_0(k + \frac{L}{2})} = -e^{-i\omega_0 k}$

# FFT Historical Trivia

- FFT actually invented by Gauss in 1805! (but lost)
- Re-invented by Cooley and Tukey in 1965
- Tukey coined the term “bit” (for “binary digit”) and was first to use the term “software” in writing



Carl Friedrich Gauss



John Tukey

# The Short-Time Fourier Transform (STFT)

# STFT Definition

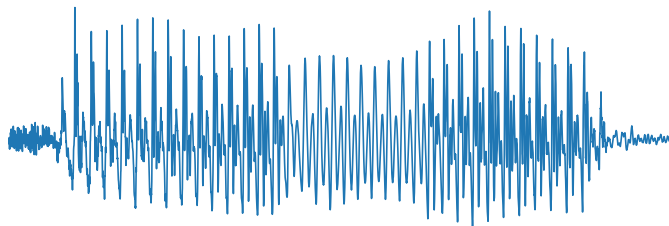
The STFT of a signal,  $x[n]$ , is a function of frequency,  $k$ , and time,  $m$ , given by:

$$X[k, m] = \frac{1}{\sqrt{W}} \sum_{n=0}^{W-1} x[n + mh] w[n] e^{-\frac{i2\pi kn}{W}},$$

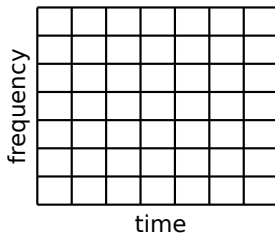
# STFT Procedure

Multiply signal  $x[n]$  by a sliding window  $w[n]$  and take FFT.

$x[n]$



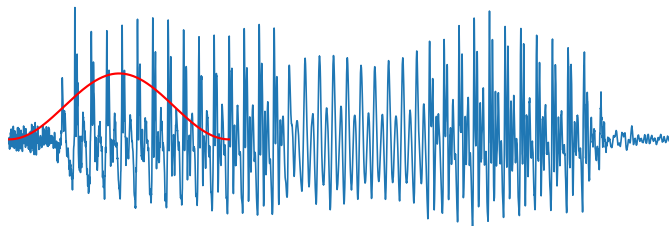
$X[k, m]$



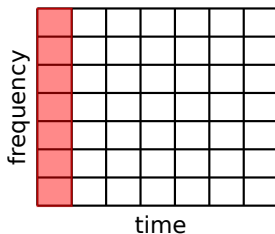
# STFT Procedure

Multiply signal  $x[n]$  by a sliding window  $w[n]$  and take FFT.

$x[n]$



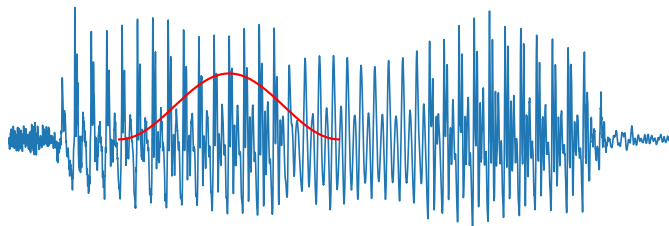
$X[k, m]$



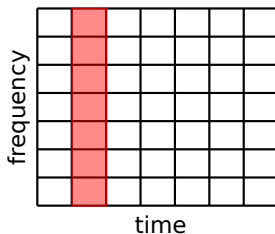
# STFT Procedure

Multiply signal  $x[n]$  by a sliding window  $w[n]$  and take FFT.

$x[n]$



$X[k, m]$

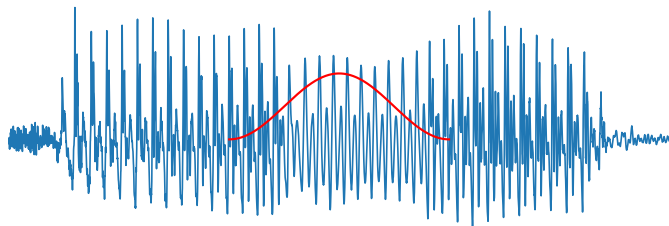




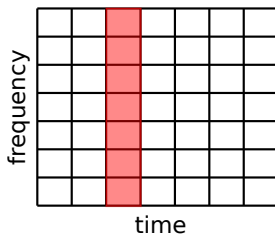
# STFT Procedure

Multiply signal  $x[n]$  by a sliding window  $w[n]$  and take FFT.

$x[n]$



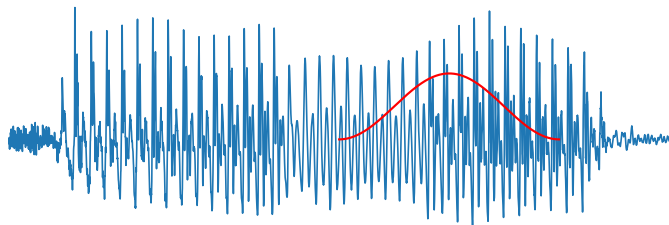
$X[k, m]$



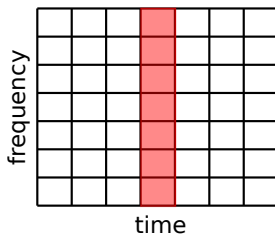
# STFT Procedure

Multiply signal  $x[n]$  by a sliding window  $w[n]$  and take FFT.

$x[n]$



$X[k, m]$



# STFT Pseudocode

```
x = input signal
w = window function
h = hop length
H = number of hops
X = output STFT

for m = 0 .. H
    x_clip = x[m*h : m*h + len(w)]
    X[:, m] = FFT(x_clip * w)
```

# Overlap-Add (OLA)

We can reconstruct a signal  $x[n]$  from its STFT,  $X[k, m]$ , using a method called **Overlap-Add (OLA)**:

- 1 Compute the inverse fast Fourier transform on each column of  $X[k, m]$  to get

$$s[n, m] = \mathcal{DFT}^{-1}(X[k, m])$$

- 2 Scale  $s$  by a window,  $w[n]$ , and sum over  $m$ :

$$\tilde{x}[n] = \sum_m s[n - mh, m]w[n - mh],$$

where  $h$  is the hop length used to compute  $X[k, m]$ , and the range of the summation is  $\lceil \frac{1-W+n}{h} \rceil \leq m \leq \lfloor \frac{n}{h} \rfloor$ .

# Pseudocode for OLA

```
X = STFT
w = window function
x = output signal

initialize x[n] = 0 for all n
for m = 0 .. H
    s = IFFT(X[:, m])
    x[m*h : m*h + len(w)] += s * w
```

# Perfect Reconstruction Conditions

The  $\tilde{x}[n]$  resulting from OLA is a reconstruction of  $x[n]$ , but are they equal?

Yes, if window satisfies the constant overlap-add (COLA) condition:

$$\sum_m w[n - mh]^2 = 1$$

Note: We can also apply two different windows  $w_f[n]$  and  $w_b[n]$  during forward STFT and backward OLA, respectively. Then the condition is that  $\sum_m w_f[n - mh]w_b[n - mh] = 1$ .

# Perfect Reconstruction Conditions

First, we have

$$\begin{aligned} s[n, m] &= \mathcal{DFT}^{-1}\{X[k, m]\} \\ &= \mathcal{DFT}^{-1}\{\mathcal{DFT}\{x[n + mh]w[n]\}\}, \\ &= x[n + mh]w[n] \end{aligned}$$

# Perfect Reconstruction Conditions

So, assuming  $w[n]$  is COLA:

$$\begin{aligned}\tilde{x}[n] &= \sum_m s[n - mh, m] w[n - mh] \\ &= \sum_m (x[n] w[n - mh]) w[n - mh] \\ &= x[n] \underbrace{\sum_m w[n - mh]^2}_{=1 \text{ if COLA}} \\ &= x[n]\end{aligned}$$