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

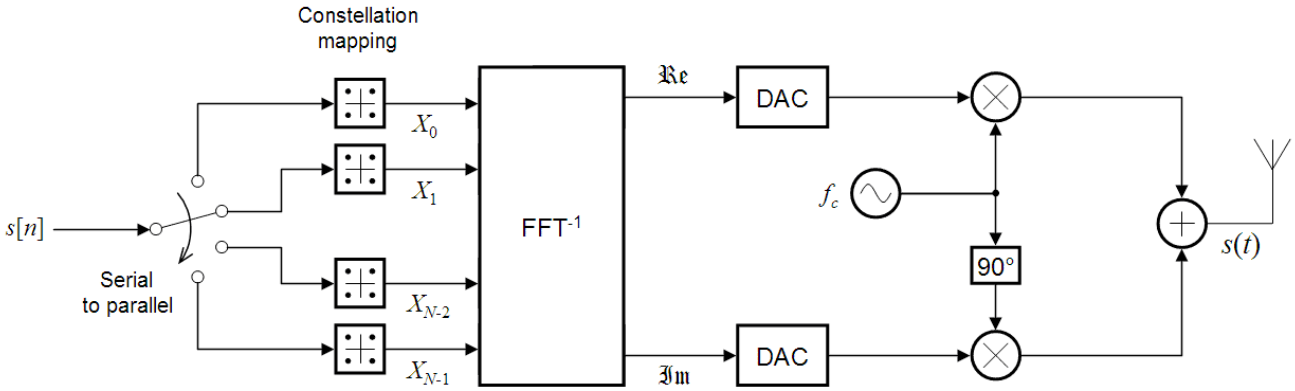
Digital Signal Processing

February 21, 2023

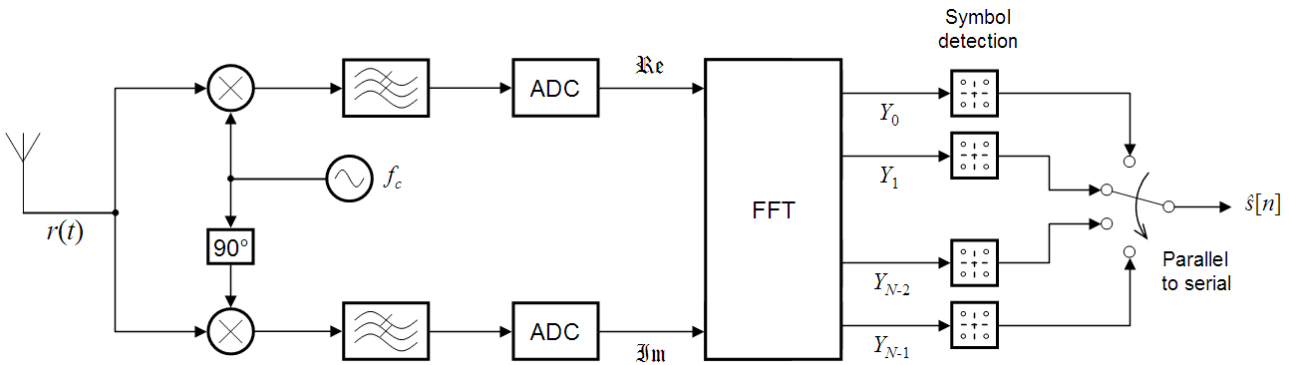
Application of FFT: Communications

Orthogonal Frequency Division Multiplexing (OFDM)

Transmitter:



Receiver:



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

The Fast Fourier Transform (FFT)

Naïve DFT Algorithm

Recall the DFT equation:

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

Naïve DFT Algorithm

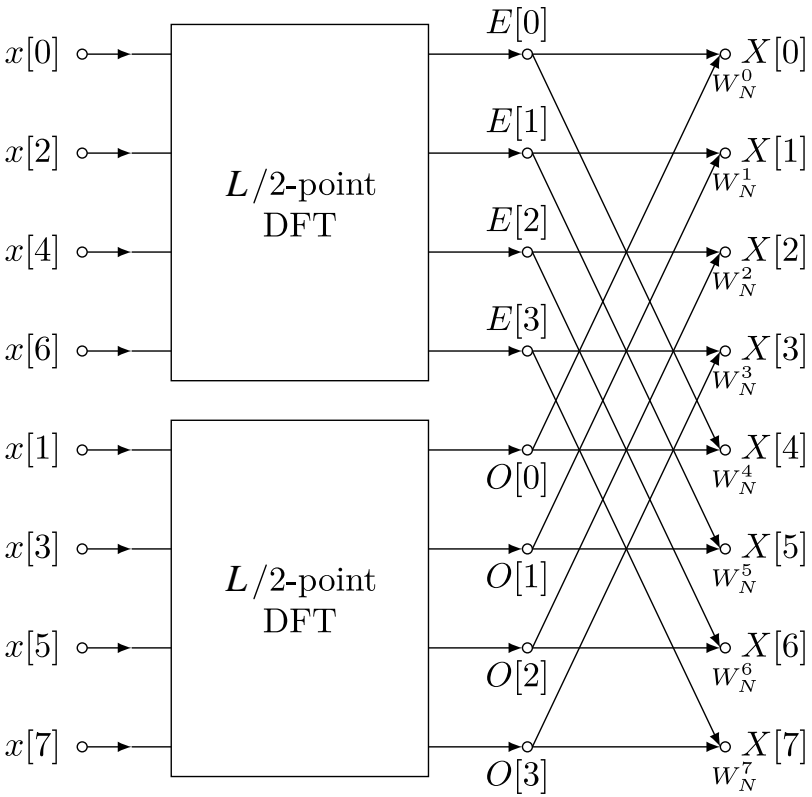
Loop over $k = 0, \ldots, L-1$

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

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, preferrably 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 $k = 0, 1, \ldots, \frac{L}{2} - 1$,

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

$$= \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2mk} x[2m] + \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 (2m+1)k} x[2m+1]$$

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

$$= E[k] + e^{-i\omega_0 k} O[k]$$

Second Half of FFT

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

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

$$= E[k] - e^{-i\omega_0 k} O[k]$$

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 $k = 0, 1, \ldots, \frac{L}{2} - 1$,

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

$$+ e^{-i\omega_0(k+\frac{L}{2})} \frac{1}{\sqrt{L}} \sum_{m=0}^{L/2-1} e^{-i\omega_0 2m(k+\frac{L}{2})} x[2m+1]$$

$$= E[k] - e^{-i\omega_0 k} O[k] \qquad \text{Reused!}$$

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

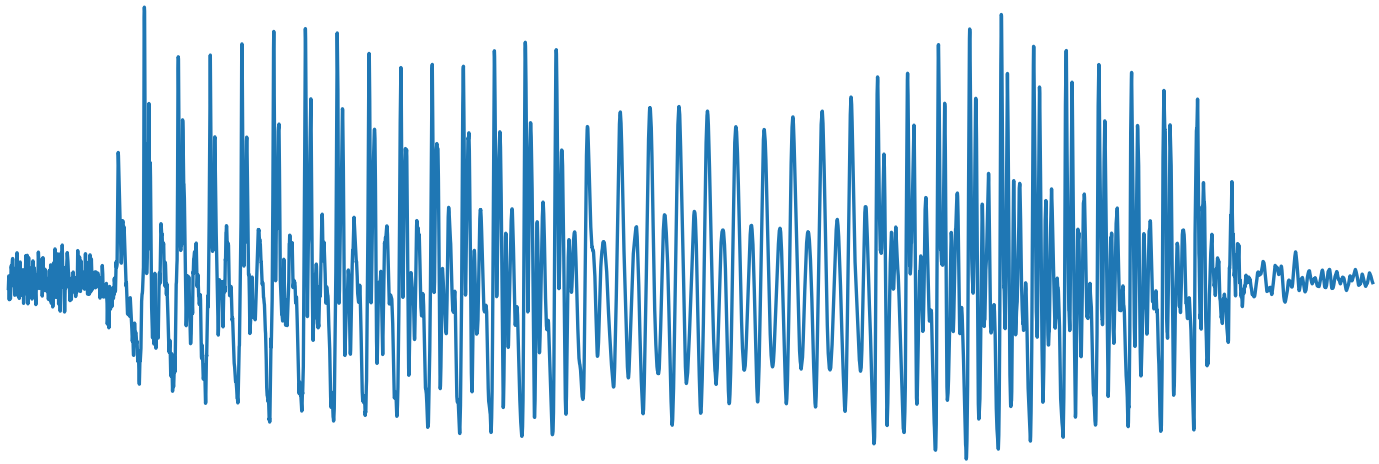
The STFT of a signal, $x[n]$, is a function of frequency, k, and time, m, given by:

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

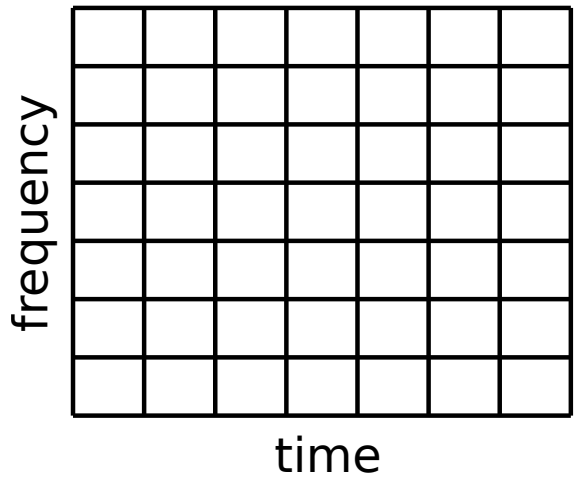
STFT Procedure

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

$x[n]$



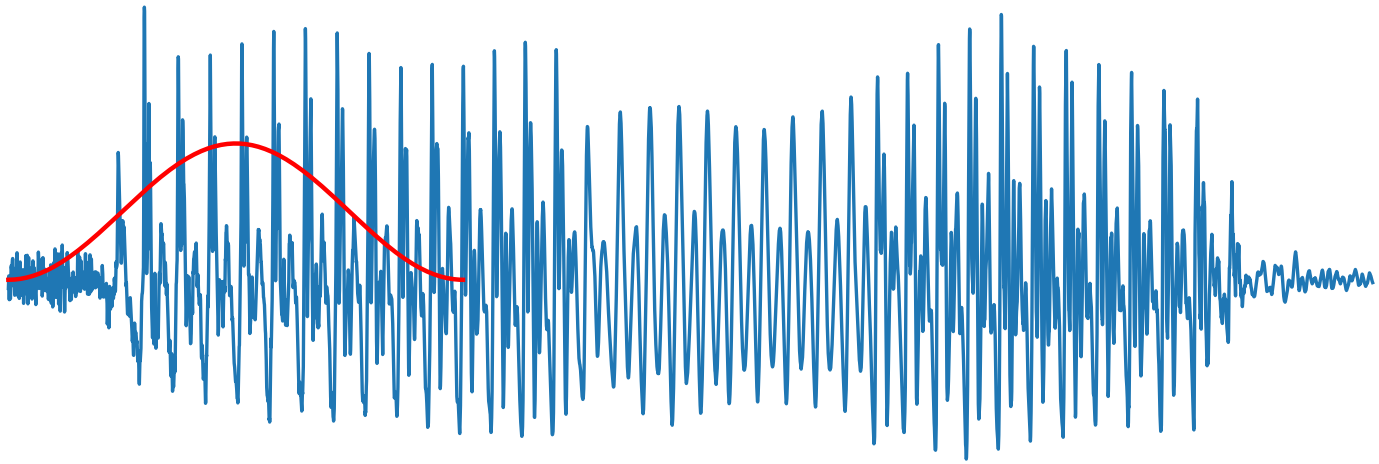
$X[k, m]$



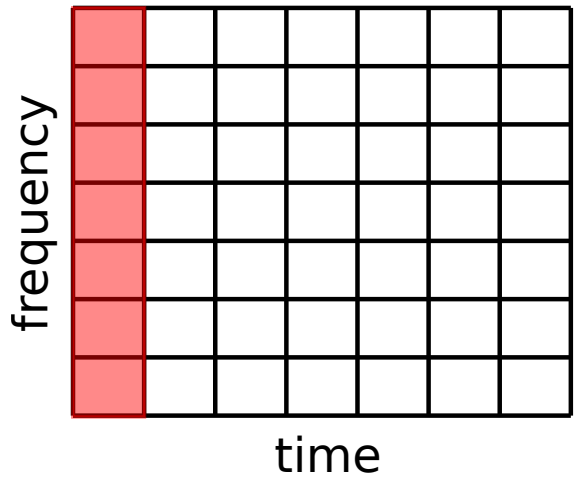
STFT Procedure

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

$x[n]$



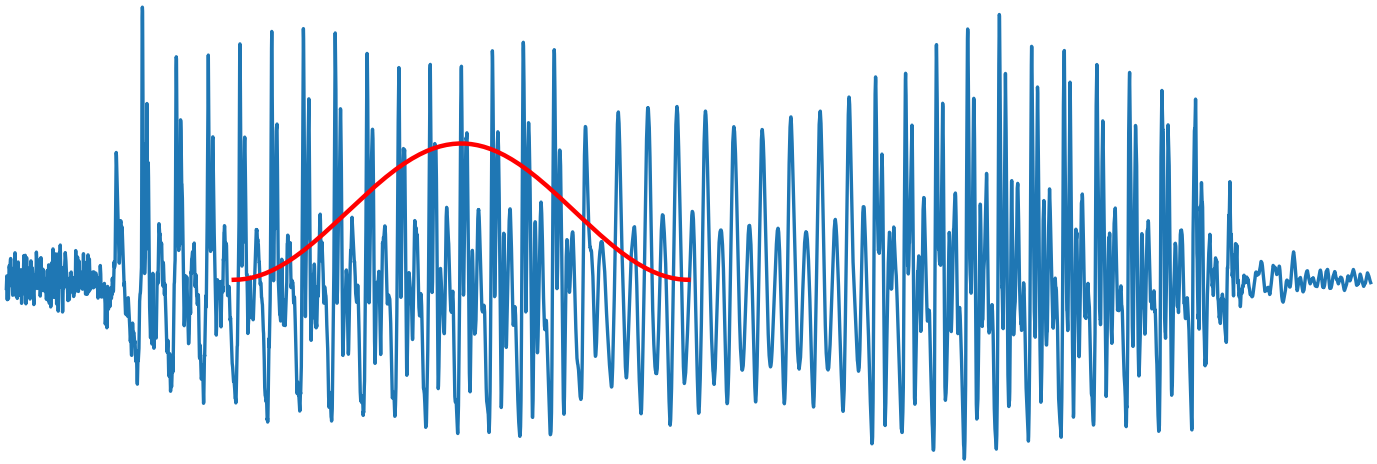
$X[k, m]$



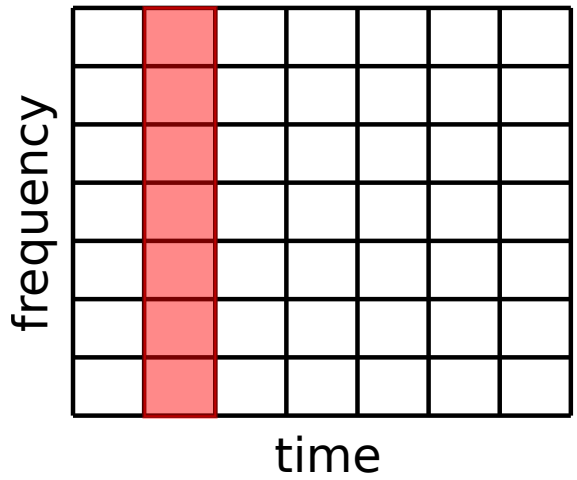
STFT Procedure

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

$x[n]$



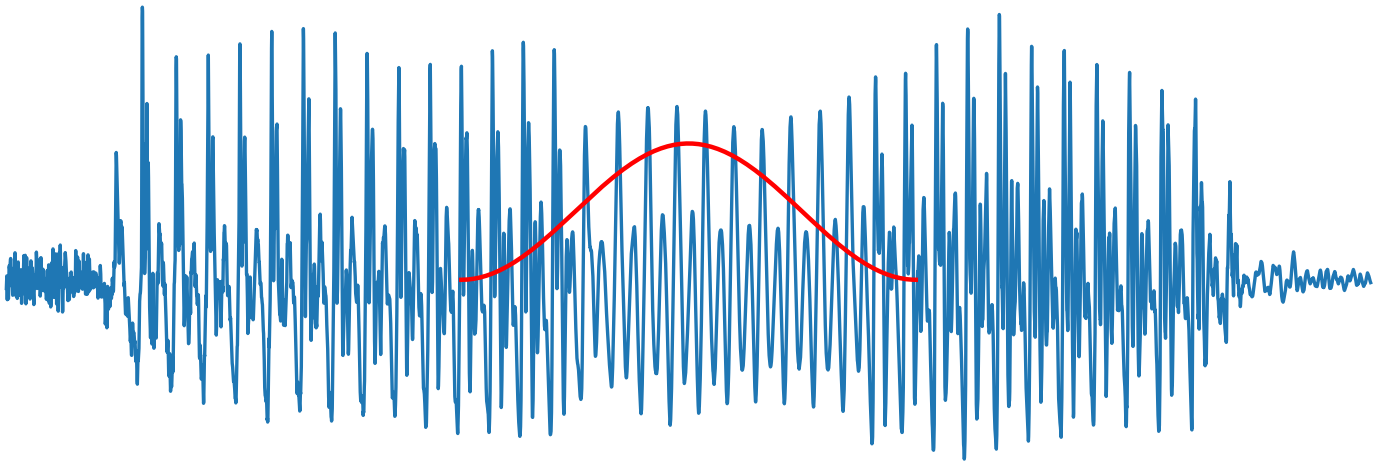
$X[k, m]$



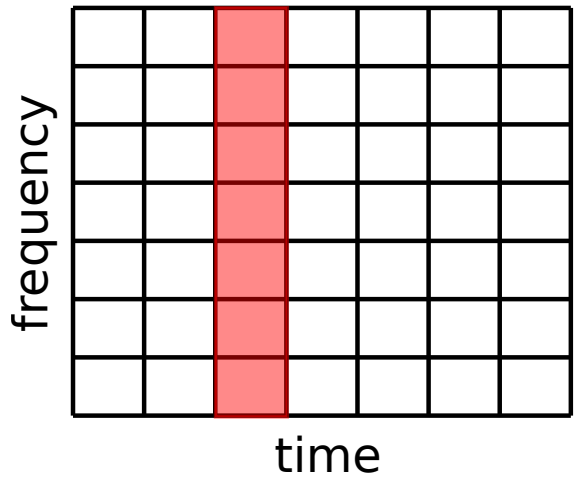
STFT Procedure

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

$x[n]$



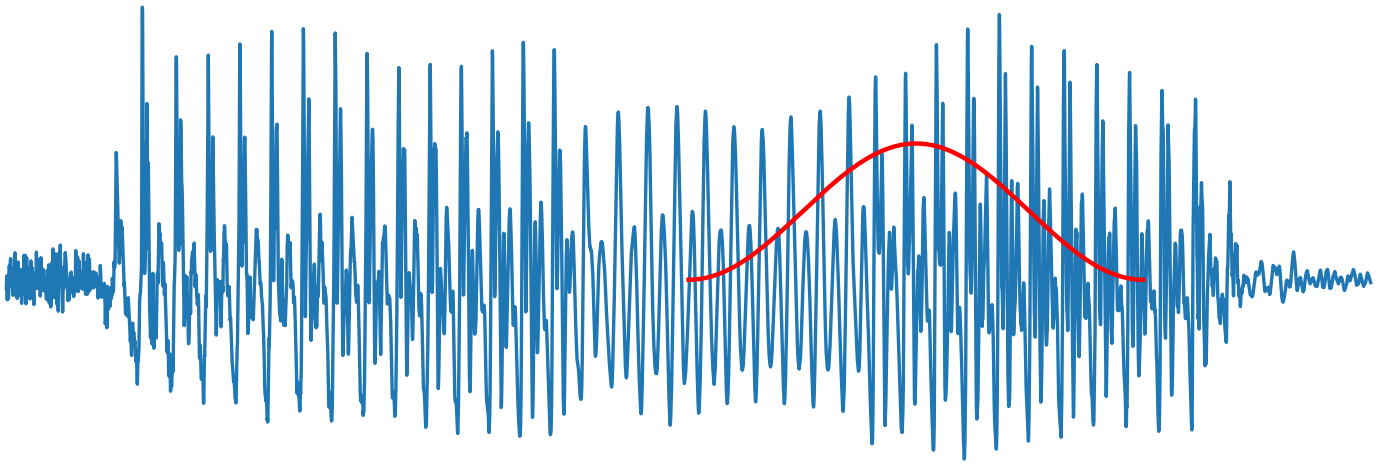
$X[k, m]$



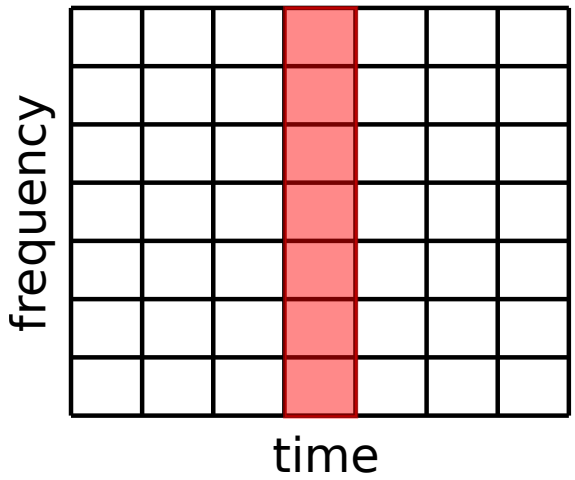
STFT Procedure

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:

$$\tilde{x}[n] = \sum_{m=0}^{W-1} s[n, m]w[n - mh],$$

where W is the length of $w[n]$, and h is the hop length used to compute $X[k, m]$.

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:

$$\sum_{n=0}^{W-1} w^2[n] = 1$$

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 $\sum_n w_f[n]w_b[n] = 1$.

Perfect Reconstricution Conditions

First, we have

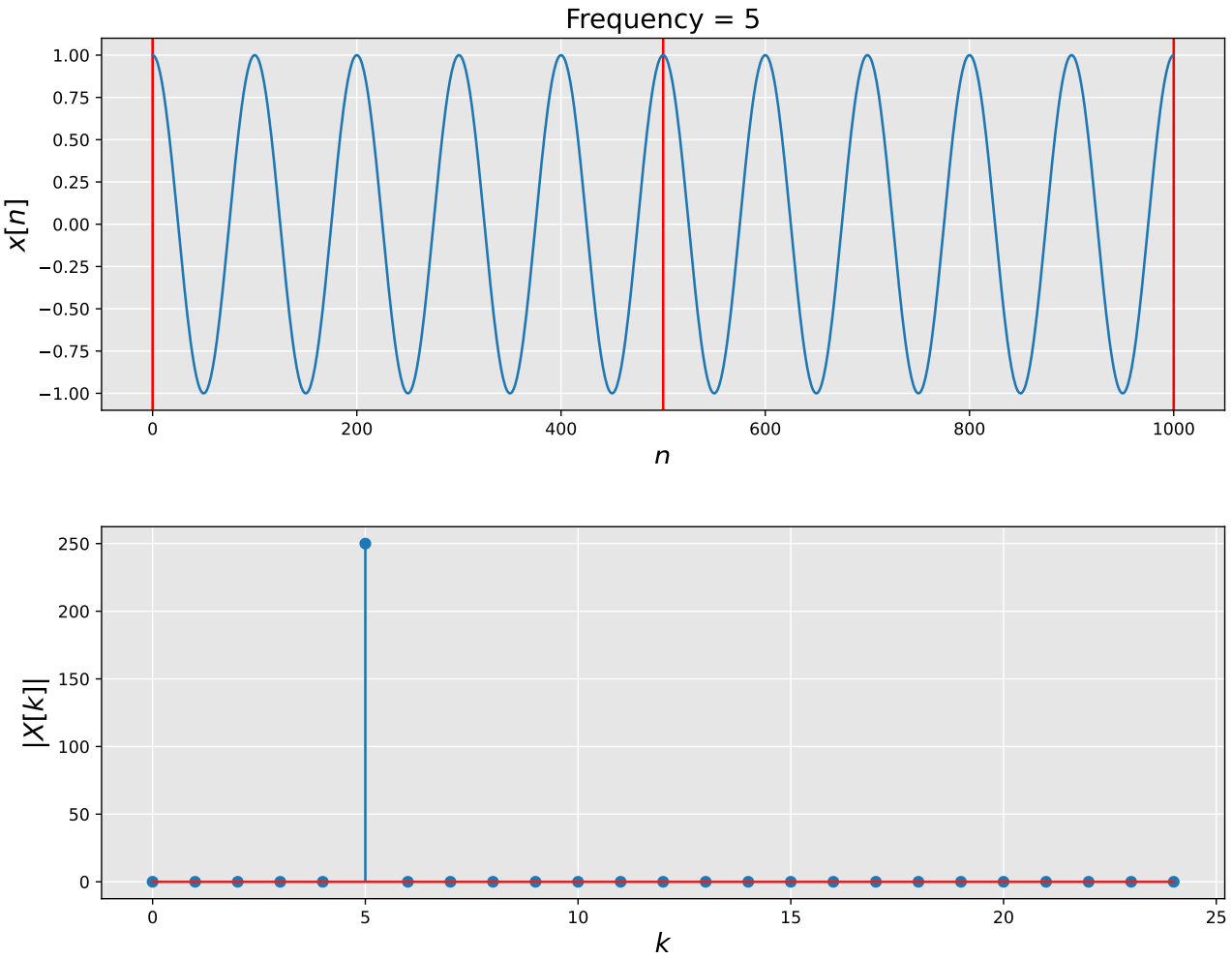
$$\begin{aligned} s[n,m] &= \mathcal{DFT}^{-1}\{X[k,m]\} \\ &= \mathcal{DFT}^{-1}\left\{\mathcal{DFT}x[n]w[n-mh]\right\}, \\ &= x[n]w[n-mh] \end{aligned}$$

Perfect Reconstricution Conditions

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

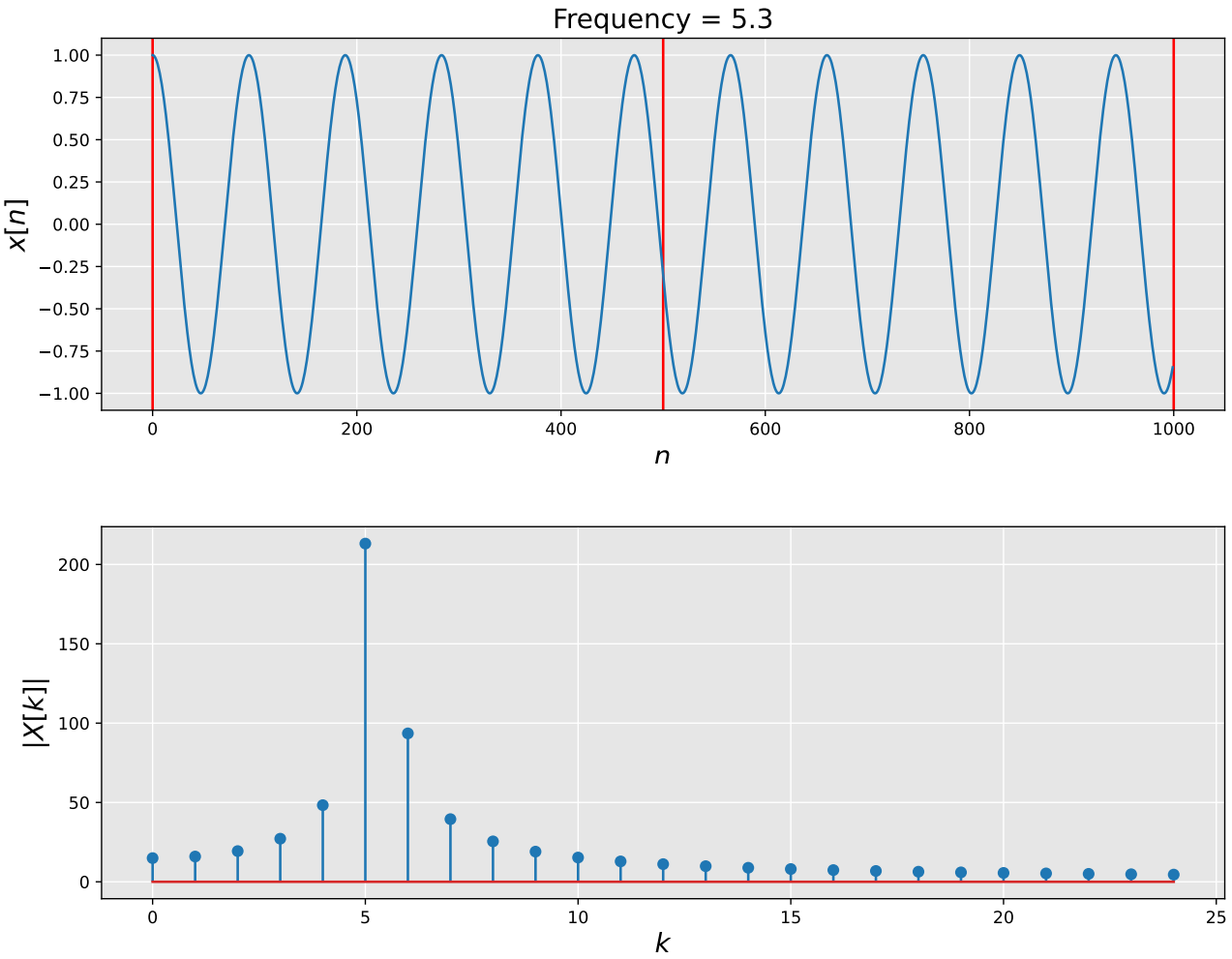
$$\begin{aligned}
\tilde{x}[n] &= \sum_{m=0}^{W-1} s[n,m]w[n-mh] \\
&= \sum_{m=0}^{W-1} (x[n]w[n-mh])w[n-mh] \\
&= x[n] \underbrace{\sum_{m=0}^{W-1} w[n-mh]^2}_{=1 \text{ if COLA}} \\
&= x[n]
\end{aligned}$$

Exact Frequency



Frequency is clear in DFT when it is an integer.

Exact Frequency



But it spreads across multiple bins when it is not an integer.

Can we Recover Non-Integer Frequency?

Continuous sinusoid:

$$x(t) = \cos(\omega_0 t + \phi)$$

Its phase angle is:

$$\theta(t) = \omega_0 t + \phi$$

Its derivative is:

$$\frac{d\theta}{dt}(t) = \omega_0$$

Which is the frequency!

Discrete Phase Derivative

Discrete sinusoid:

$$x[n] = \cos(\omega_0 n + \phi)$$

Its phase angle is:

$$\theta[n] = \omega_0 n + \phi$$

Discrete derivative (backward difference):

$$\begin{aligned}\nabla\theta[n] &= \theta[n] - \theta[n-1] \\ &= \omega_0 n + \phi - (\omega_0(n-1) + \phi) \\ &= \omega_0\end{aligned}$$

Again, frequency!

Using the STFT Phase Angle

- The DFT doesn't give us phase angle as a function of time.
- The STFT does give an estimated phase angle as a function of time!
- So, given an STFT, $X[k, m]$, we can estimate the exact frequency represented in frequency bin k at time n as:

$$\omega^*[k, n] = \text{Arg}\left(X[k, n]\right) - \text{Arg}\left(X[k, n-1]\right)$$