ELEC 221 Lecture 13
The fast Fourier transform

Thursday 20 October 2022

Announcements

- Assignment 4 due next Wednesday
- Quizzes resume on Tuesday
- Tuesday's lecture will be a hands-on (instructions for what to bring will be posted on Piazza)

Last time

We introduced the **discrete-time Fourier transform** (DTFT):

Inverse DTFT (synthesis equation):

DTFT (analysis equation):

Last time

There is a convolution property just like in CT:

Same consequences for the impulse response in DT:

Last time

We looked at some of its key properties.

It is periodic:

There are some convergence criteria:

Practice problem

Oppenheim 5.9.

Suppose $x[n]$ is a real signal with spectrum $X(e^{j\omega})$ and we know the following about it:

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\text{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

What is $x[n]$?

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\text{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

Given a real signal $x[n]$, last class we saw that

We can find the odd part of $x[n]$ by taking the inverse DTFT.

Practice problem

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\text{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

$$\text{Odd}(x[n]) = \frac{1}{2}(\delta[n+1] - \delta[n-1] - \delta[n+2] + \delta[n-2])$$

Know that

Allows us to determine that

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\mathrm{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

So,

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\mathrm{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

We have

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\mathrm{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

Leverage Parseval's relation:

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\mathrm{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

$$x[n] \quad = \quad \begin{cases} 0 & n > 0 \\ ?? > 0 & n = 0 \\ \delta[n+1] - \delta[n+2] & n < 0 \end{cases}$$

Structure of $x[n]$ so far means that

Practice problem

1. $x[n] = 0, \quad n > 0$
2. $x[0] > 0$
3. $\mathrm{Im}(X(e^{j\omega})) = \sin\omega - \sin 2\omega$
4. $\frac{1}{2\pi}\int_{2\pi} |X(e^{j\omega})|^2 d\omega = 3$

$$x[n] = \begin{cases} 0 & n > 0 \\ 1 & n = 0 \\ \delta[n+1] - \delta[n+2] & n < 0 \end{cases}$$

At the end of the day, we can simply write

Today

Learning outcomes:

- Distinguish between the discrete-time Fourier transform and the discrete Fourier transform
- Describe the fast Fourier transform (FFT) algorithm
- Implement a basic FFT algorithm and state its algorithmic scaling

The discrete Fourier transform

Take another look at the DTFT:

$X(e^{j\omega})$ is a **continuous function** of ω.

The discrete Fourier transform

Consider what we have been doing in Python so far:

```
>>> import numpy as np
>>> x = np.array([1, 2, 3, 4, 5])
>>> y = np.fft.fft(x)
>>> y
array([15. +0.j,
       -2.5+3.4409548j,
       -2.5+0.81229924j,
       -2.5-0.81229924j,
       -2.5-3.4409548j])
```

We are computing a Fourier spectrum here. The signal is discrete and finite, and *the spectrum is also discrete and finite*. What is this thing?

The discrete Fourier transform

Recall how we derived the DTFT. We had some signal, and considered a periodic extension of it.

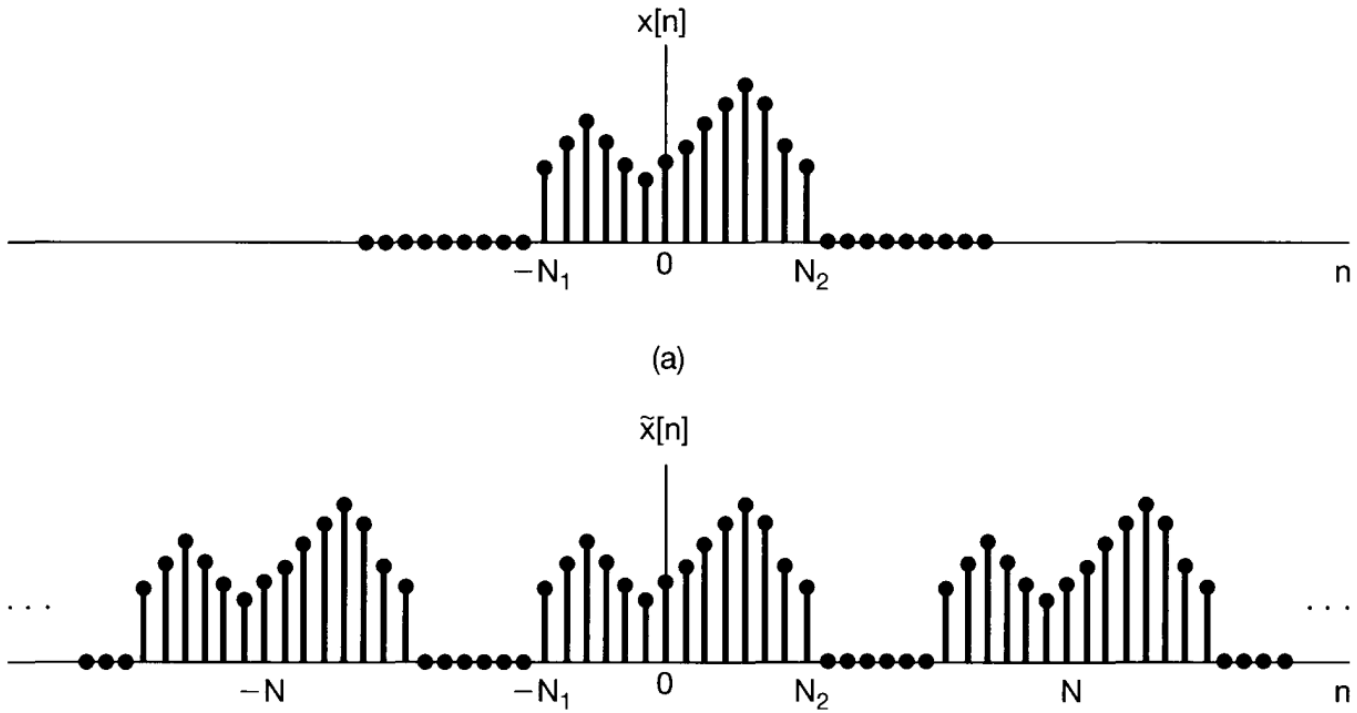


Image credit: Oppenheim chapter 5.1

The discrete Fourier transform

Without loss of generality, suppose $x[n]$ is defined from 0 to some $N_1 \leq N$.

We can define $\tilde{x}[n]$ as a periodic function with period N. Then, it has Fourier series coefficients

These $N-1$ coeffcients are known as the **discrete Fourier transform** of $x[n]$.

The discrete Fourier transform

Generally we write

Why are these interesting?

First, we can recover $x[n]$ from them:

The discrete Fourier transform

More importantly, these are related to the discrete-time Fourier transform:

If the signal is only defined from 0 to $N - 1$,

The discrete Fourier transform

What if we sample this signal at particular values of $k\omega = k2\pi/N$?

The discrete Fourier transform is a set of equally spaced samples of the full discrete-time Fourier transform.

Key point 1: Any signal $x[n]$ can be uniquely specified by a finite set of samples from its DTFT (i.e., its DFT).

The fast Fourier transform

How do we actually compute the DFT?

The fast Fourier transform

This is just a big matrix multiplication!

The fast Fourier transform

We typically define $\zeta = e^{-2\pi j/N}$ (Nth root of unity):

$$\begin{bmatrix} \tilde{X}[0] \\ \tilde{X}[1] \\ \cdots \\ \tilde{X}[N-1] \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \zeta & \zeta^2 & \cdots & \zeta^{N-1} \\ 1 & \zeta^2 & \zeta^4 & \cdots & \zeta^{2(N-1)} \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & \zeta^{N-1} & \zeta^{2(N-1)} & \cdots & \zeta^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} x[0] \\ x[1] \\ \cdots \\ x[N-1] \end{bmatrix}$$

The fast Fourier transform

This matrix is very beautiful:

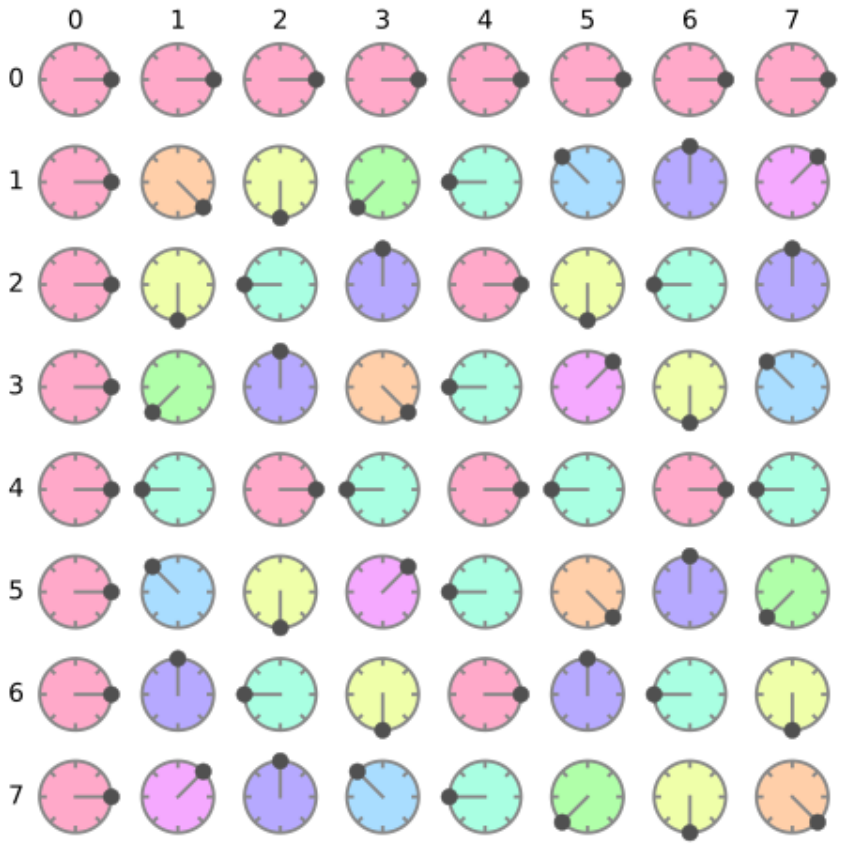


Image credit: Darnling - Own work. CCBY-SA4.0 https://creativecommons.org/licenses/by-sa/4.0

The fast Fourier transform

Key point 2: There is an efficient algorithm for the DFT: the **fast Fourier transform**.

These correspond (almost) to:

```
>>> X_k = np.fft.fft(xn)
>>> xn = np.fft.ifft(X_k)
```

Note that in NumPy, the normalization convention is "backwards":

Implementation details

There are many ways to define the DFT, varying in the sign of the exponent, normalization, etc. In this implementation, the DFT is defined as

$$A_k = \sum_{m=0}^{n-1} a_m \exp\left\{-2\pi i \frac{mk}{n}\right\} \qquad k = 0, \ldots, n-1.$$

To match the convention we use in class, you need to specify:

```
>>> y = np.fft.fft(x, norm='forward')
```

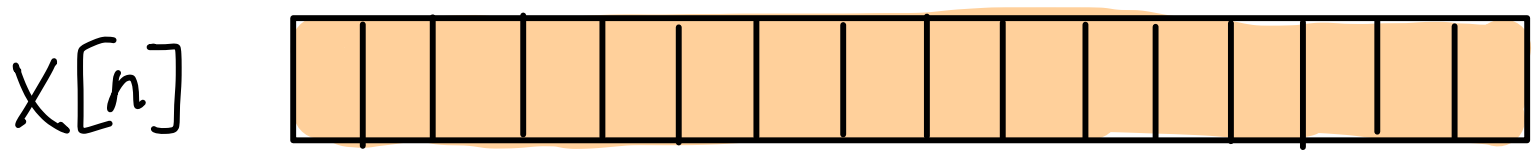
Screenshot: `https://numpy.org/doc/stable/reference/routines.fft.html#background-information`

The fast Fourier transform

The most famous FFT algorithm is the **Cooley-Tukey algorithm**: it computes the DFT using a divide and conquer method.

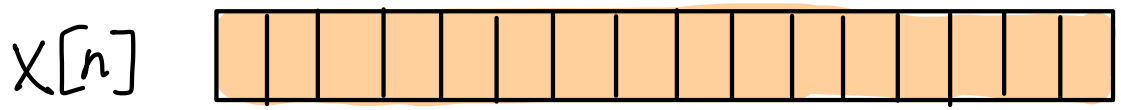
Nicest case to analyze is when N is a power of 2.

Suppose we would like to compute the kth element of the DFT.

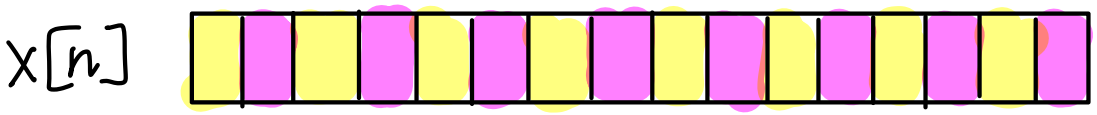


$$\tilde{X}[k] = \sum_{n=0}^{N-1} x[n] e^{-\frac{j 2\pi k n}{N}}$$

The fast Fourier transform



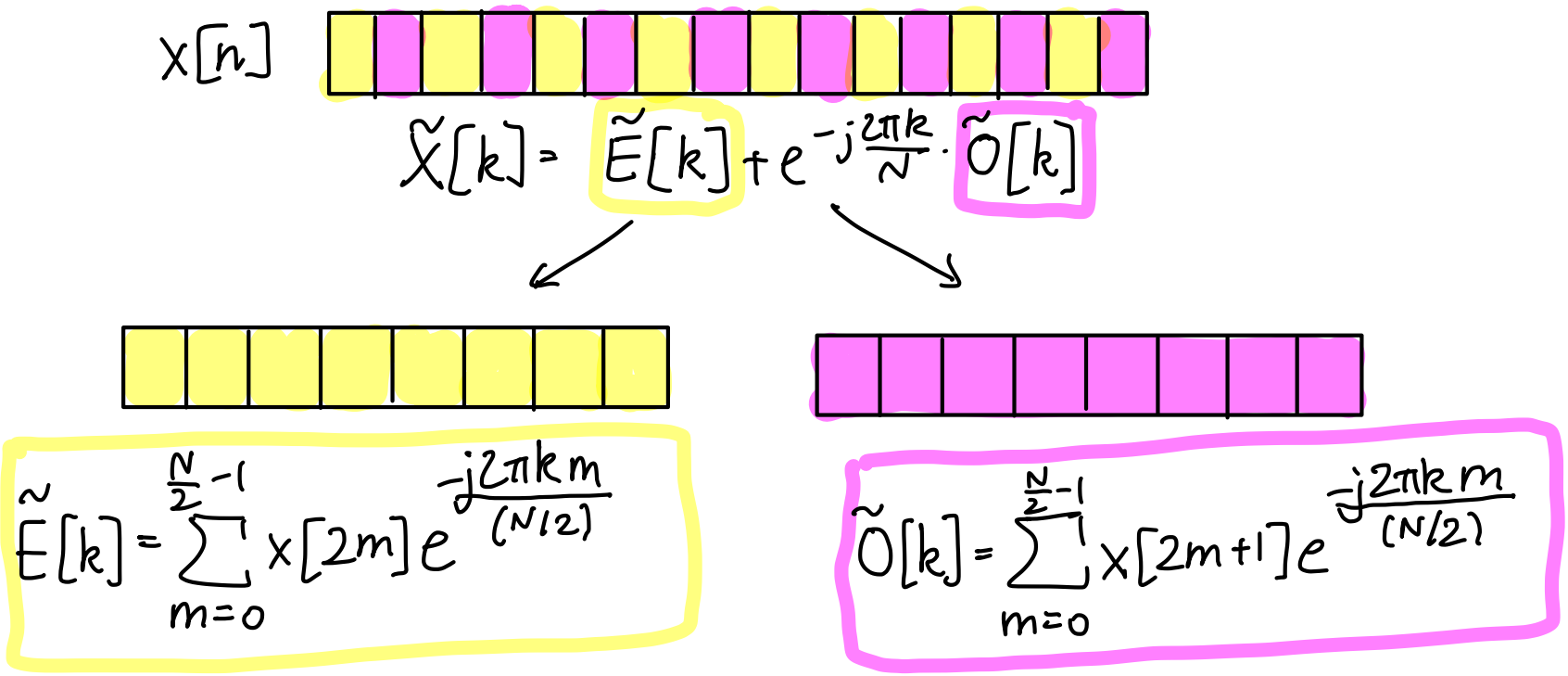
$$\tilde{X}[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}$$



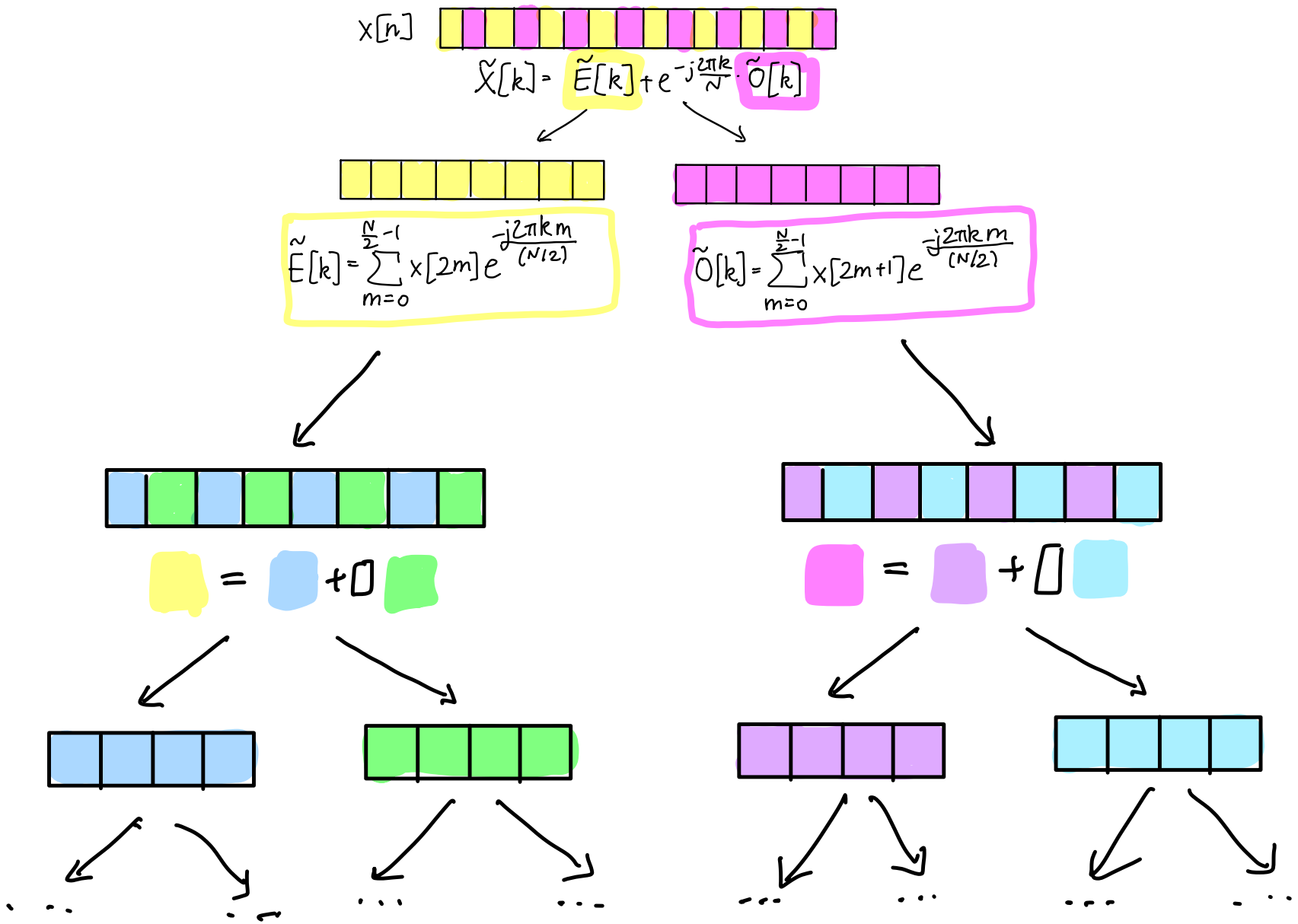
$$\tilde{X}[k] = \sum_{m=0}^{\frac{N}{2}-1} x[2m] e^{\frac{-j2\pi k(2m)}{N}} + \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] e^{\frac{-j2\pi k(2m+1)}{N}}$$

$$= \sum_{m=0}^{\frac{N}{2}-1} x[2m] e^{\frac{-j2\pi k(2m)}{N}} + e^{\frac{-j2\pi k}{N}} \sum_{m=0}^{\frac{N}{2}-1} x[2m+1] e^{\frac{-j2\pi k(2m)}{N}}$$

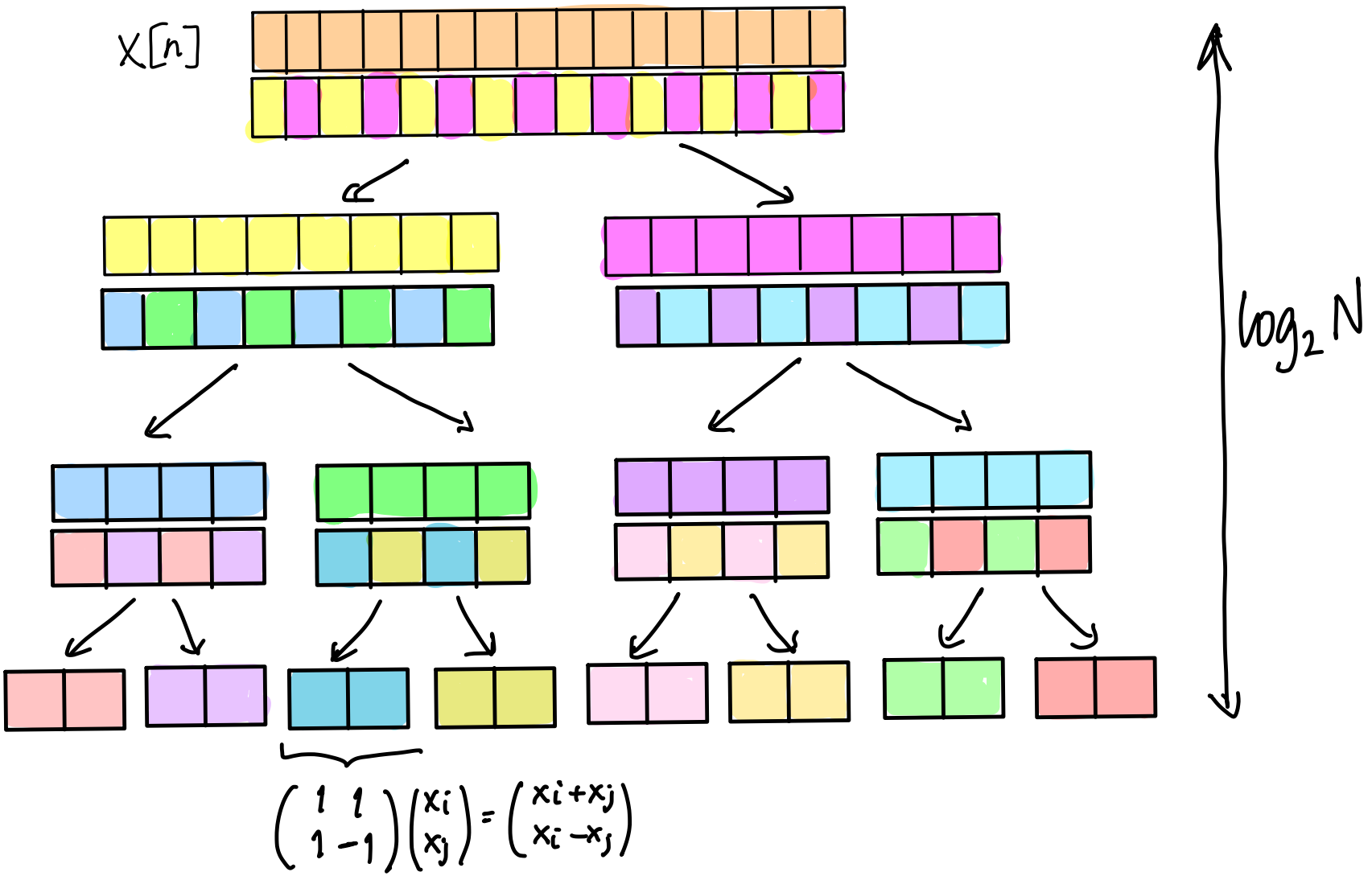
The fast Fourier transform



The fast Fourier transform



The fast Fourier transform



The fast Fourier transform

But wait!

We get something for free.

The fast Fourier transform

Recall that there is some symmetry between positive/negative values in Fourier coefficients.

The fast Fourier transform

$$\begin{aligned}\tilde{X}[k+N/2] &= \sum_{m=0}^{N/2-1} x[2m]e^{-j\frac{2\pi}{N/2}km}e^{-j2\pi m} \\ &\quad + e^{-j(k+N/2)\frac{2\pi}{N}} \sum_{m=0}^{N/2-1} x[2m+1]e^{-j\frac{2\pi}{N/2}km}e^{-j2\pi m}\end{aligned}$$

The fast Fourier transform

$$\begin{aligned}\tilde{X}[k+N/2] &= \sum_{m=0}^{N/2-1} x[2m]e^{-j\frac{2\pi}{N/2}km} \\ &\quad + e^{-jk\frac{2\pi}{N}}e^{-j\pi}\sum_{m=0}^{N/2-1} x[2m+1]e^{-j\frac{2\pi}{N/2}km}\end{aligned}$$

The fast Fourier transform

Only need to compute the first $1/2$ of values in each subdivision of the problem.

At the end of the day, the complexity of the FFT is $O(N \log_2 N)$ - way better than $O(N^2)$ matrix multiplication!

The fast Fourier transform

Let's program it. We will compare:

- The naive matrix multiplication version
- A naive implementation of the FFT
- NumPy's FFT

If you want to dig deeper:
`https://nbviewer.org/url/jakevdp.github.io/downloads/notebooks/UnderstandingTheFFT.ipynb`

Recap

Today's learning outcomes were:

- Distinguish between the discrete-time Fourier transform and the discrete Fourier transform
- Describe the steps of the fast Fourier transform (FFT) algorithm
- Implement a basic FFT algorithm and state its algorithmic scaling

What topics did you find unclear today?

For next time

Content:

- Hands-on: 2D Fourier analysis and image processing

Action items:

1. Quiz 6 on Tuesday
2. Assignment 4 due on Wednesday
3. Check Piazza for instructions on what to bring for next class

Recommended reading:

- From today's class: Oppenheim extension problems 5.53-5.54
- For next class: research the 2D DFT