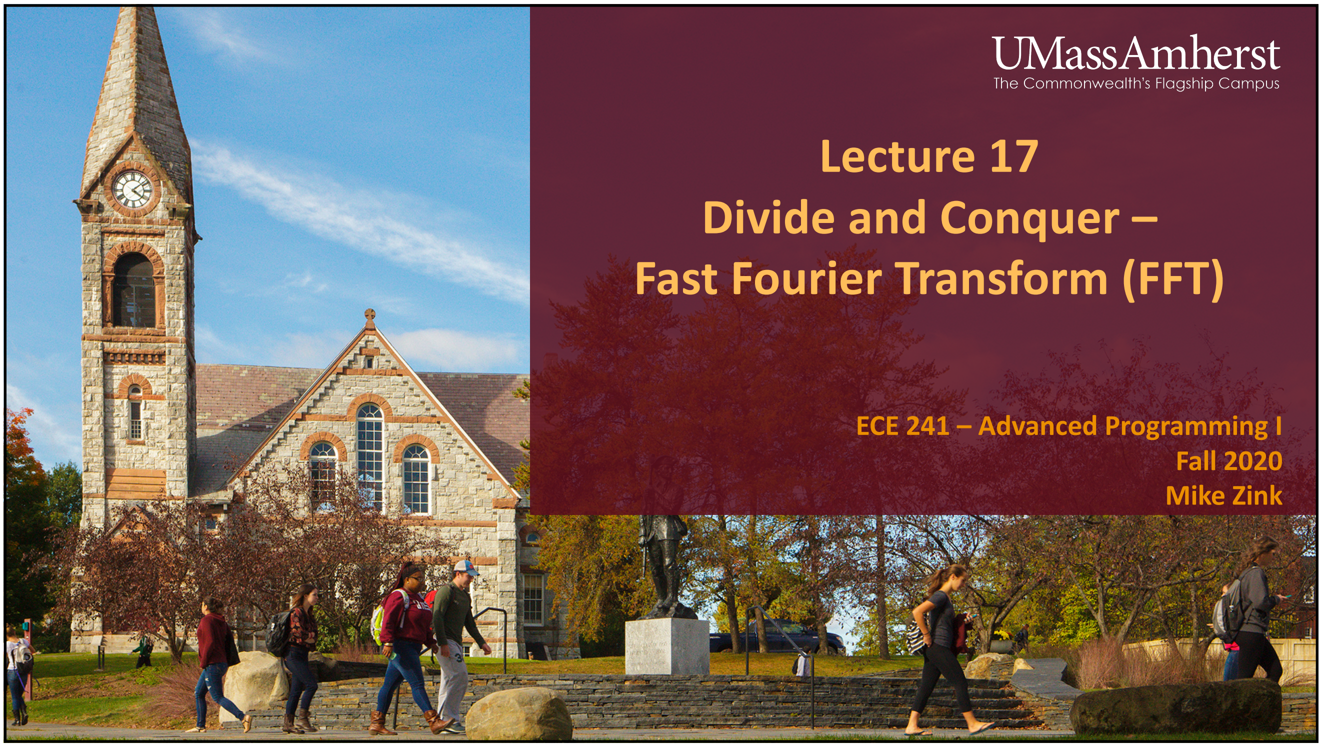


0

UMassAmherst

Introduction

- In several cases, it is desirable to evaluate a signal in the **frequency domain** as it gives a more insightful information about it.
- A few use cases of FFT:
  - audio processing to clear noise
  - image processing to smooth images
  - OFDM (used in cellular communication)
  - speech recognition
  - audio fingerprinting (apps like Shazam and SoundHound)

1

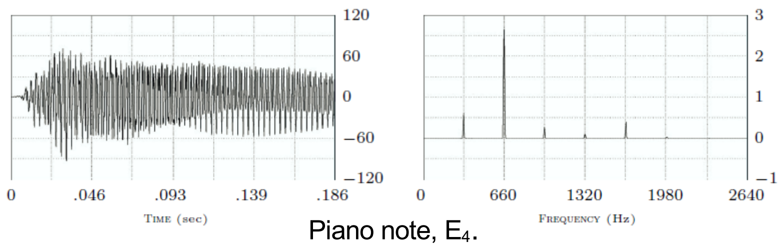
UMassAmherst

Fourier Transform

- Given the original signal, $f(t)$, the Fourier transform is denoted by

$$F(j\omega) = \int_{-\infty}^{\infty} f(t)e^{-2j\omega t}dt$$

- It decomposes the signal in the time domain into the frequency domain. For example:



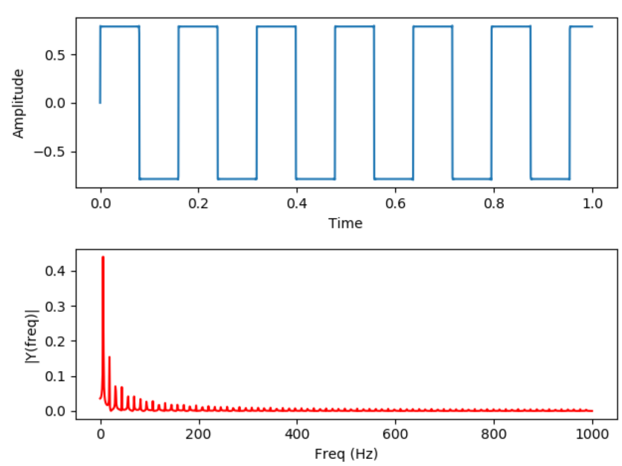
Piano note, E_4.
(Source: Time-Frequency Analysis of Musical Instruments)

2

2

UMassAmherst

Fourier Transform



- The square wave on the top left is composed of a sum of multiple sine waves.
- Fourier Transform allows us to visualize a signal in the **frequency** domain, showing all its components, called **harmonics**.
- The Fourier Transform is also useful to find distortions in a signal (among other applications).

3

3

UMassAmherst

Discrete Fourier Transform (DFT)

- The DFT is a discrete representation of the continuous Fourier transform, which can be fed into a computer.
- Let N samples be denoted by $r = 0, 1, \dots, N-1$

$$A_r = \sum_{k=0}^{N-1} X_k e^{-2j\omega kT}$$

A_r is the r^{th} coefficient of the DFT.

X_k is the k^{th} sample of the time series.

- Using conventional methods, the DFT algorithm takes $\boldsymbol{O}(\boldsymbol{N}^2)$ operations.

Reference: What Is the Fast Fourier Transform?, by WT Cochran et al. - 1967

4

4

UMassAmherst

Fast Fourier Transform (FFT)

- It is a numerically efficient way to calculate the DFT
- It was originally developed by Gauss around 1805, but rediscovered by Cooley and Tukey in 1965
- The FFT algorithm exploits the symmetries of $e^{-j\frac{2\pi}{N}kn}$

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

1. Complex conjugate symmetry $W_N^{k(N-n)} = W_N^{-kn} = \left(W_N^{kn}\right)^*$
2. Periodicity in n,k $W_N^{kn} = W_N^{k(N+n)} = W_N^{(k+N)n}$

5

5

Fast Fourier Transform (FFT)

UMassAmherst

- Uses divide and conquer algorithm to simplify the number of operations (break big FFT into smaller FFT, easier to solve)

1. **Divide** into even and odd summations of size $(N/2)$. This is called decimation in time:
   Y_k: even-numbered points $(X_0, X_2, X_4, \dots)$
   Z_k: odd-numbered points $(X_1, X_3, X_5, \dots)$

$$A_r = \sum_{k=0}^{\frac{N}{2}-1} Y_k e^{-\frac{4\pi jrk}{N}} + e^{\frac{-2\pi jr}{N}} \sum_{k=0}^{\frac{N}{2}-1} Z_k\, e^{-\frac{4\pi jrk}{N}}$$

$$r = 0, 1, \dots, \frac{N}{2} - 1$$

6

6

Fast Fourier Transform (FFT)

UMassAmherst

2. **Conquer**: recursively compute $Y_k \; and \; Z_k$
   $Y_k \; and \; Z_k$ can each be divided by 2 (yielding $N/4$ samples).
   If $N = 2^n$, we can make n such reductions.
3. **Combine**

$$A_r = Y_k(X^2) + x.Z_k(X^2)$$

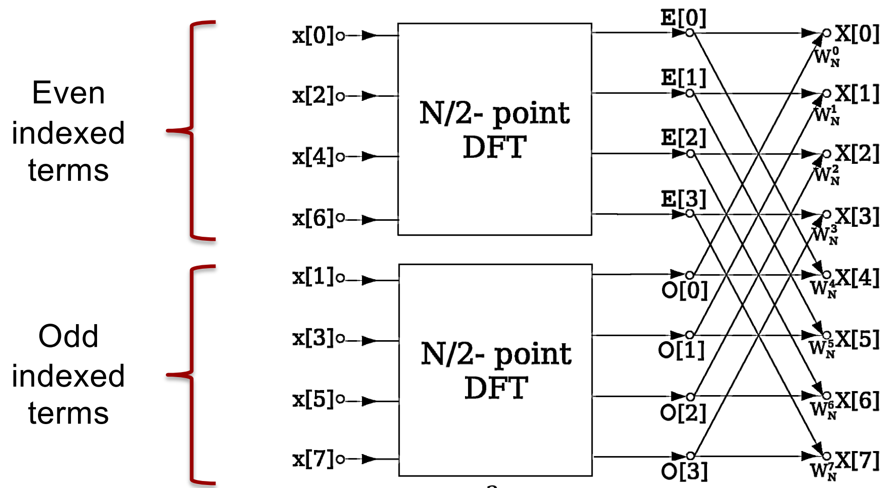
- The FFT algorithm takes $\boldsymbol{O(N\, log_2\, N)}$ operations.

7

7

UMassAmherst

Example for N=8



$\left(\frac{N}{2}\right)^2 * 2$ $+ N$ $\approx \left(\frac{N}{2}\right)^2 + \mathrm{N}$

UMassAmherst

Example for N=8

- Keep splitting the terms, i.e., each $\frac{N}{2} = 2 * \frac{N}{4}$ DFTs
- We can split $\log_2 N$ times
- As N gets large

$$\approx O(N \log_2 N)$$

UMassAmherst

DFT algorithm implementation in Python

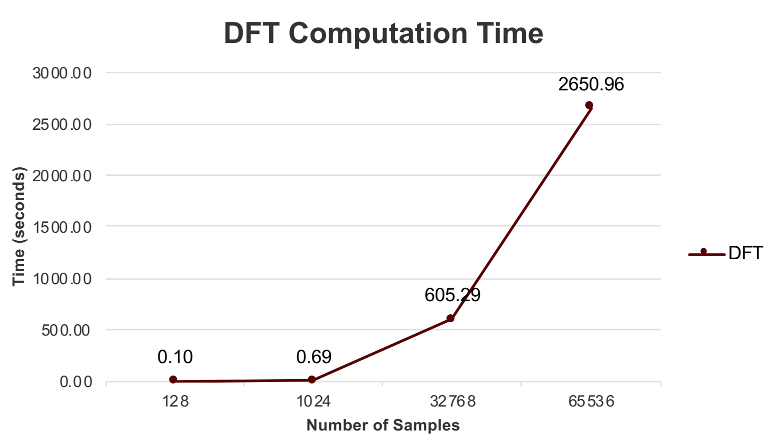
```
import numpy as np
from timeit import Timer

pi2 = np.pi * 2


def DFT(x):
    N = len(x)
    FmList = []
    for m in range(N):
        Fm = 0.0
        for n in range(N):
            Fm += x[n] * np.exp(- 1j * pi2 * m * n / N)
        FmList.append(Fm / N)
    return FmList


N = 1000
x = np.arange(N)
t = Timer(lambda: DFT(x))
print('Elapsed time: {} s'.format(str(t.timeit(number=1))))
```

10

10

UMassAmherst

DFT Performance



All this and following experiments were run on a virtual machine running Ubuntu 18.04 LTS with one processor (Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz) and 3GB of memory.

11

11

UMassAmherst

FFT algorithm implementation in Python

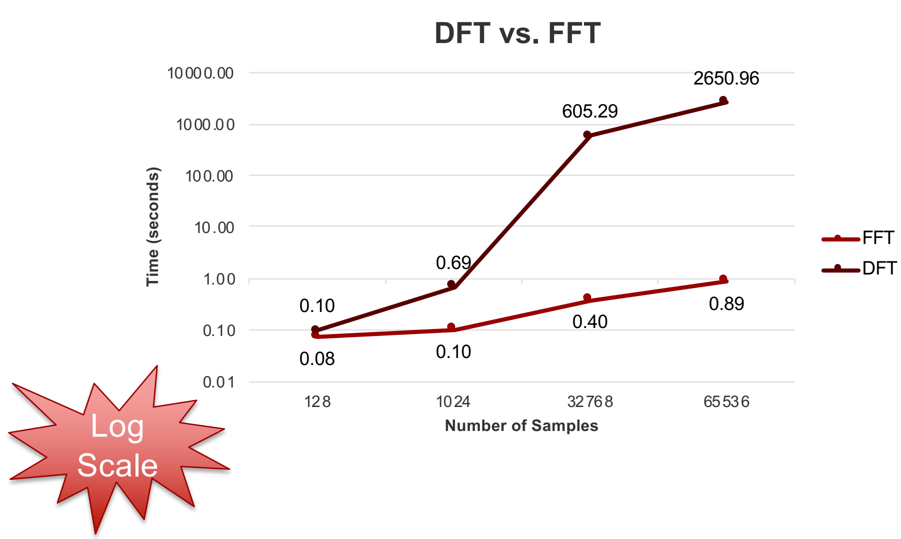
```
# Recursive FFT function

import numpy as np


def FFT(x):
    N = len(x)
    if N <= 1: return x
    even = FFT(x[0::2])
    odd = FFT(x[1::2])
    T = [np.exp(-2j * np.pi * k / N) * odd[k] for k in range(N // 2)]
    return [even[k] + T[k] for k in range(N // 2)] + \
           [even[k] - T[k] for k in range(N // 2)]


N = 1024
x = np.random.random(N)
t = Timer(lambda: FFT(x))
print('Elapsed time: {} s'.format(str(t.timeit(number=1))))
```

12

12

UMassAmherst

FFT Performance



13

13

UMassAmherst

Numpy implementations

```
# FFT example using the Numpy fftpack

import numpy as np
from timeit import Timer

N = 10000
x = np.arange(N)
t = Timer(lambda: np.fft.fft(x))
print('Elapsed time: {} s'.format(str(t.timeit(number=1))))
```

14

14

UMassAmherst

Scipy implementations

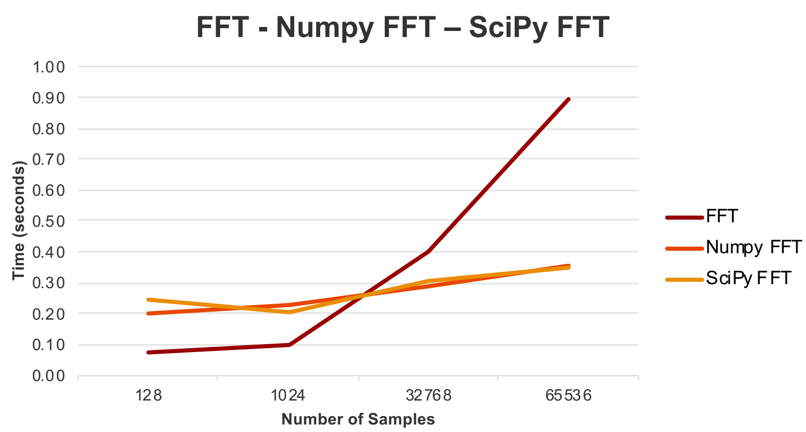
```
# FFT example using the SciPy fftpack

import scipy
from scipy.fftpack import fft
from timeit import Timer

N = 10000
x = scipy.arange(N)
t = Timer(lambda: fft(x))
print('Elapsed time: {} s'.format(str(t.timeit(number=1))))
```

15

15

UMassAmherst

To put things into perspective



16

16

UMassAmherst

Application – Audio Fingerprinting

- Audio fingerprinting is a signature that summarizes an audio recording
- Also known as Content-Based audio Identification (CBID)
- The best known application are apps like Shazam and SoundHound, that link unlabeled audio recordings to a corresponding metadata (song name and artist, for instance)

Source: http://willdrevo.com/fingerprinting-and-audio-recognition-with-python/ for all following slides, unless otherwise stated

17

17

UMassAmherst

Background on Digital Audio

- **Sampling:** the standard sampling rate in digital music, such as HIFI, is 44,100 samples per second (from Nyquist theorem – 2 x 20 kHz)
- **Quantization:** the standard quantization uses 16 bits, or 65,536 levels
- **PCM or Pulse Code Modulation:** is the representation of the analog signal into zeros and ones
- This means that each second of music will have 44,100 samples per channel (one channel – Mono; two channels – Stereo)
E.g.: 3 minutes of stereo song will have 15,876,000 samples

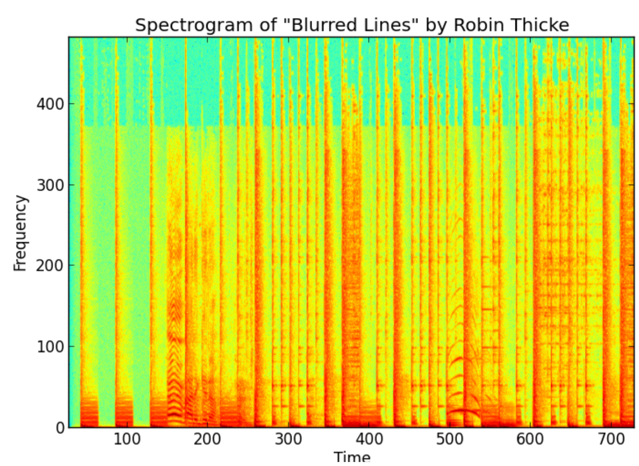
18

18

UMassAmherst

How to fingerprint an Audio

- We use the FFT to analyze the audio signal in the frequency domain
- Then we create a **spectrogram** of the song, a visual representation of the frequencies as they vary in time
- Amplitude:
Red color – higher value,
Green color – lower value

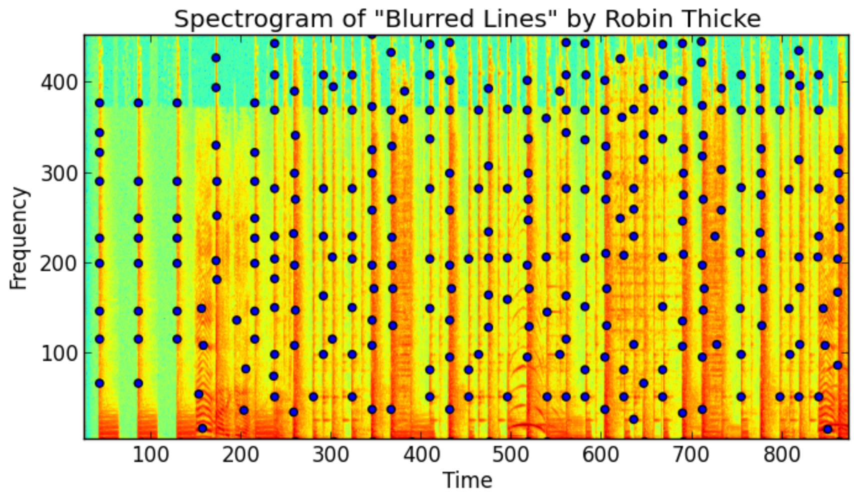


19

19

UMassAmherst

Finding Peaks

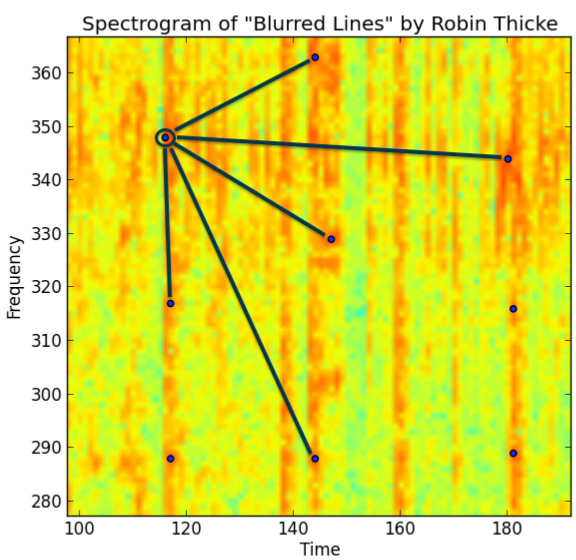


20

20

UMassAmherst

Fingerprint Hashing



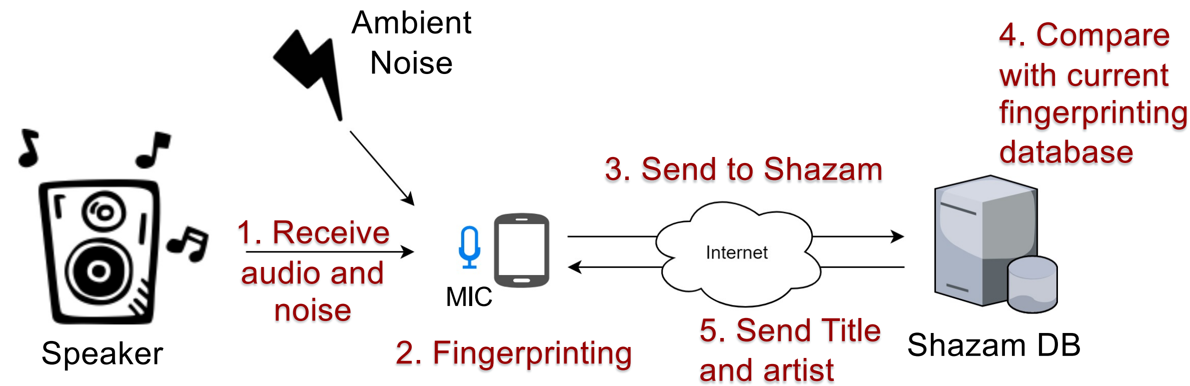
- We hash the frequency of peaks and the time difference between them
- The result is a unique fingerprint for the song
- Each app has its own hashing function to uniquely identify a song

21

21

UMassAmherst

How Shazam Works in a Nutshell



Source: An Industrial-Strength Audio Search Algorithm, by Avery Li-Chun Wang (Shazam Whitepaper) 22

22

UMassAmherst
The Commonwealth's Flagship Campus

23