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)
• Given the original signal, \( f(t) \), the Fourier transform is denoted by

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

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

![Piano note, E₄.](Source: Time-Frequency Analysis of Musical Instruments)

- 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).
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 $r = 0, 1, \ldots, N - 1$

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

$A_r$ 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 $O(N^2)$ operations.


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 $n,k$ $W_N^{kn} = W_N^{k(N+n)} = W_N^{(k+N)n}$
Fast Fourier Transform (FFT)

• 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:
   \[ Y_k : \text{even-numbered points } (X_0, X_2, X_4, ...) \]
   \[ Z_k : \text{odd-numbered points } (X_1, X_3, X_5, ...) \]

   \[
   A_r = \sum_{k=0}^{N-1} Y_k e^{-\frac{4\pi j r k}{N}} + e^{-\frac{2\pi j r}{N}} \sum_{k=0}^{N-1} Z_k e^{-\frac{4\pi j r k}{N}}
   \]

   \[ r = 0, 1, ..., \frac{N}{2} - 1 \]

   \[
   r = 0, 1, ..., \frac{N}{2} - 1
   \]

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 \( O(N \log_2 N) \) operations.
Example for $N=8$

- Keep splitting the terms, i.e., each $\frac{N}{2} = 2 \times \frac{N}{4}$ DFTs
- We can split $\log_2 N$ times
- As $N$ gets large
  \[ \approx O(N \log_2 N) \]
DFT algorithm implementation in Python

```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))))
```

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.
FFT algorithm implementation in Python

```python
#!/usr/bin/env python

from cmath import exp, pi

def FFT(x):
    N = len(x)
    if N % 2 > 0:
        raise ValueError("size of x must be a power of 2")
    # If problem is small enough, use our previous DFT function
    elif N <= 32:
        return DFT(x)
    else:
        X_even = FFT(x[::2])
        X_odd = FFT(x[1::2])
        factor = np.exp(-2j * pi * np.arange(N) / N)
        T = [np.exp(-2j) * np.pi * k / N for k in range(N // 2)]
        return np.concatenate([X_even + factor[:N // 2] * X_odd,
                                X_even + factor[N // 2:] * X_odd])

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

**FFT Performance**

![Log Scale]

DFT vs. FFT

<table>
<thead>
<tr>
<th>Time (sec nanos)</th>
<th>Number of Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.08</td>
<td>128</td>
</tr>
<tr>
<td>0.10</td>
<td>1024</td>
</tr>
<tr>
<td>0.69</td>
<td>32768</td>
</tr>
<tr>
<td>0.40</td>
<td>65536</td>
</tr>
<tr>
<td>0.89</td>
<td>2650.96</td>
</tr>
</tbody>
</table>

**Numpy implementations**

```python
# 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))))
```
Scipy implementations

# FFT example using the SciPy fftpack

```python
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))))
```

To put things into perspective

![FFT - Numpy FFT – SciPy FFT graph](image)
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

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
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

Finding Peaks
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

How Shazam Works in a Nutshell

1. Receive audio and noise
2. Fingerprinting
3. Send to Shazam
4. Compare with current fingerprinting database
5. Send Title and artist

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