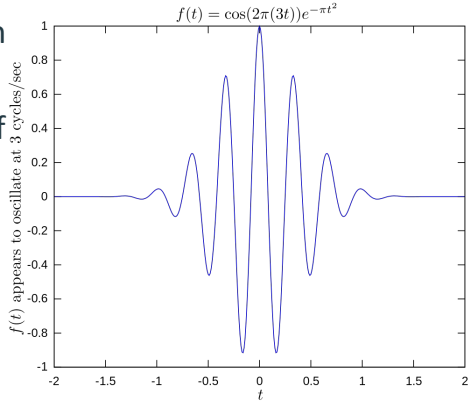
Concurrent Implementation of the Fast Fourier Transform in Kotlin

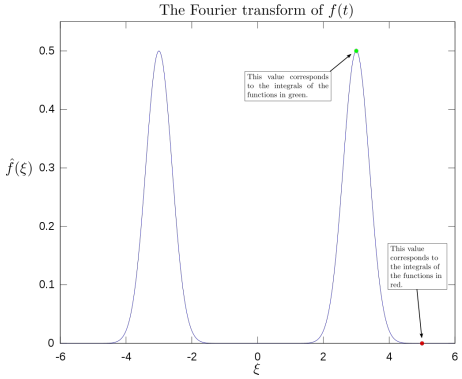
Benjamin W. Amato

The Discrete Fourier Transform

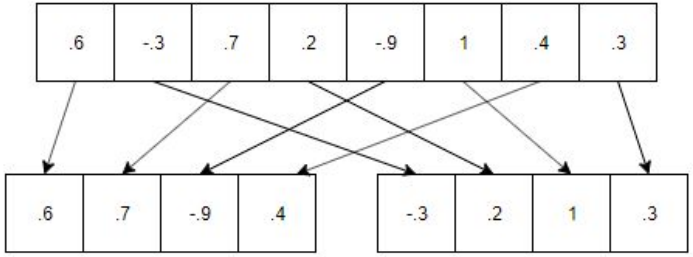
- Convert between time and frequency domain
- Operates on periodic signals
- Can decompose periodic signals into a sum of sine waves because of the sum-of-sines property
- O(N^2) Computational Complexity
- Can be expressed by the equation:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$





Fast Fourier Transform



- Algorithm uses here is the Cooley-Tukey FFT
- Separates the FFT into parts, every other element and uses divide and conquer
- Computational Complexity of O(Nlog(N))

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i2\pi kn/N}$$

- Only works on even inputs
- Relies on standard DFT for odd inputs

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i2\pi k2m/N} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i2\pi k(2m+1)/N}$$

$$X_k = \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i2\pi k2m/N} + e^{-i2\pi k/N} \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i2\pi k(2m)/N}$$

Why Kotlin?

- Wanted to use one of the languages learned in class
- Swift, miniKanren, Idris were not reasonable
- Already lots of implementations in lisp / haskell
- Concurrency seemed simpler in Kotlin than F#
    - Use of async / await
    - Data classes good for quick development
    - Infix operator overloading for Complex numbers
    - Java interoperability allowed use of Java WavFile and JFreeChart libraries

Result Analysis

- DFT is much slower for large Lists
- Similar for speed for list of 64 length
- 700 times faster for list of 32768 length

Size	DFT (ms)	FFT (ms)
64	0	1
128	2	1
256	6	3
512	24	4
1024	114	8
2048	397	17
4096	1664	36
8192	6798	72
16384	27354	145
32768	113108	295

Results Continued

- Concurrent faster at large list sizes
- Gain less than DFT vs FFT
- Small sizes have similar values because cutoff
- Run on 4 core-8 thread machine
- Actual improvement is 2-4x for larger sizes

Size	Non-Concurrent (ms)	Concurrent (ms)
1024	9	9
2048	17	29
4096	36	37
8192	75	76
16384	154	90
32768	299	152
65536	626	180
131072	1326	311
262144	2688	805

Output Analysis

- Sum of two sine wave results in two spikes
- Music looks somewhat random prior to fft
- After FFT, almost all sound is less than 10,000hz, primarily less than 5000hs

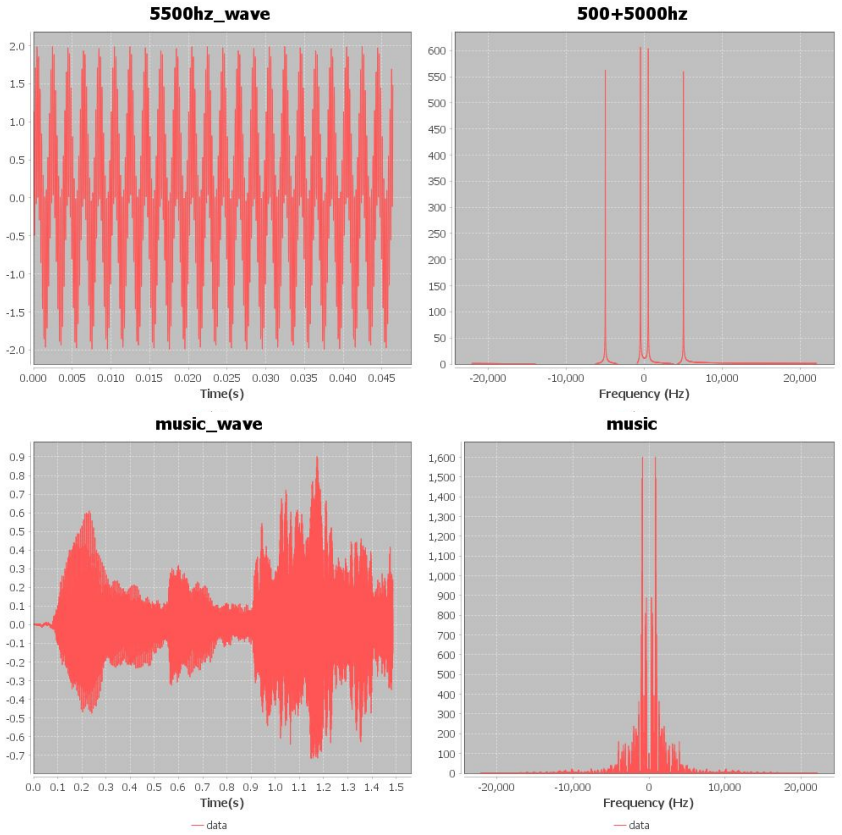


Image Sources

https://en.wikipedia.org/wiki/File:Function_ocsillating_at_3_hertz.svg

https://en.wikipedia.org/wiki/File:Fourier_transform_of_oscillating_function.svg