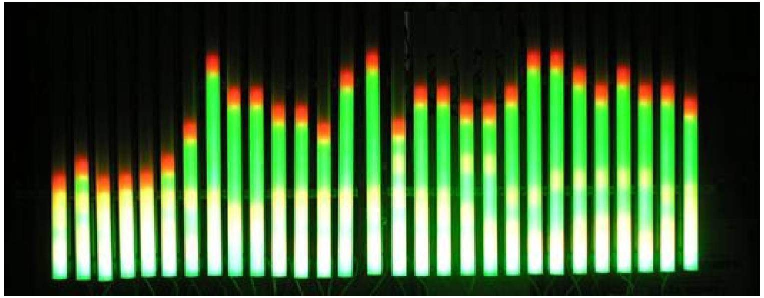
Fast Fourier Transforms

Michael Palumbo
Astro 503

Brief Review: Fourier Transform Applications

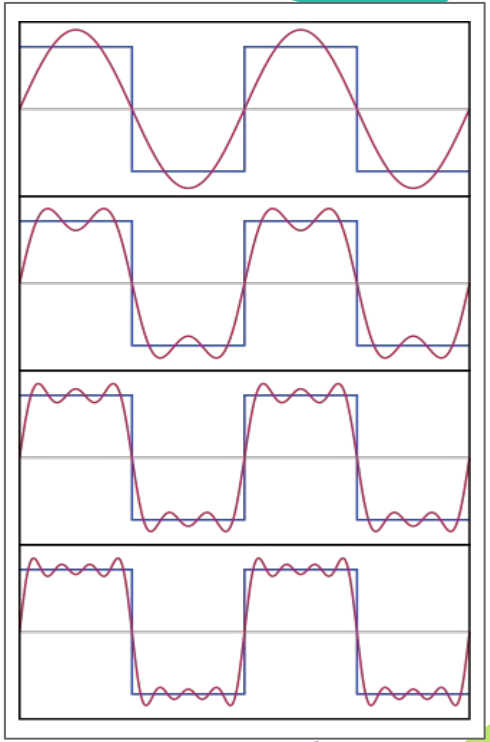
- Fourier Transforms are ubiquitous in compression (mp3, jpeg, etc)
- Signal processing: Shazam identifies music by comparing the Fourier decomposition of a recording to a library of known songs



Ask a Mathematician: What is a Fourier transform?

Brief Review: Analytical Form

- Fourier Series represent a wave-like function as a sum of sine waves
- The Fourier Transform is the generalization of the Series as $L \to \infty$
- "Transform" refers to both the operation and the resulting function



Wikipedia: Fourier Series

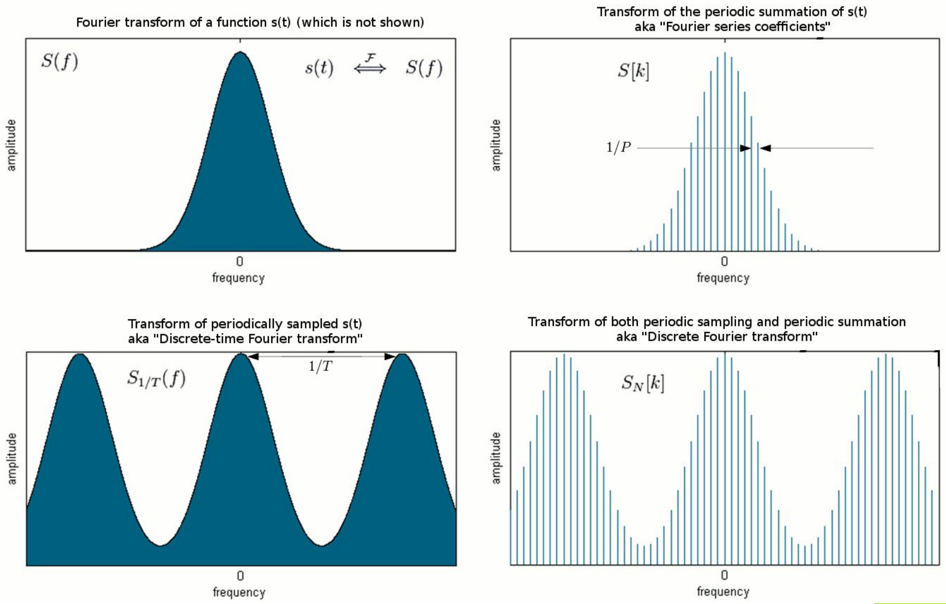
Brief Review: Analytical Form

	Series	Transform
Trigonometric	$f(x) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty} a_n \cos(nx) + \sum_{n=1}^{\infty} b_n \sin(nx)$ $a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)dx$ $a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx)dx$ $b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)dx$	n/a
Complex	$f(x) = \sum_{n=-\infty}^{\infty} A_n e^{inx}$ $A_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} f(x)e^{-inx}dx$	$F(k) = \int_{-\infty}^{\infty} f(x)e^{-2\pi ikx}dx$ $f(x) = \int_{-\infty}^{\infty} F(k)e^{2\pi ikx}dk$

Reproduced from Wolfram MathWorld

Brief Review: Discrete Fourier Transform

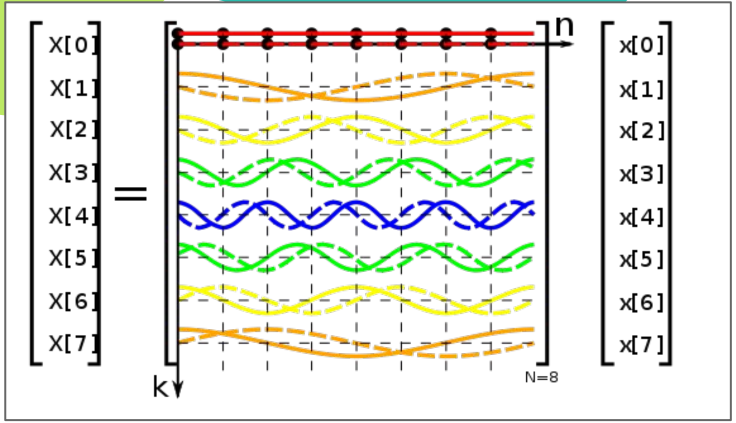
- Converts a finite sequence of function samples into the discrete Fourier Transform



Wikipedia: Discrete Fourier Transform

Brief Review: The DFT Matrix (Brute Force)

- We can express the DFT as a transformation matrix, which we can apply to the signal.
- $X = Wx$
- $\omega = e^{-2\pi i/N}$
- Sum form: $X_k = \sum_{j=0}^{n-1} \exp(-2\pi i k j/n) x_j$
- Drawback: requires $O(n^2)$ operations

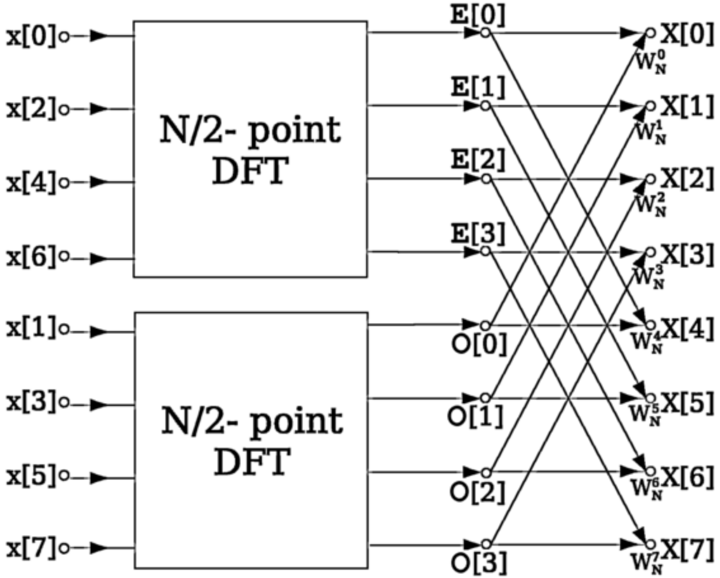


$$W = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \omega^3 & \cdots & \omega^{N-1} \\ 1 & \omega^2 & \omega^4 & \omega^6 & \cdots & \omega^{2(N-1)} \\ 1 & \omega^3 & \omega^6 & \omega^9 & \cdots & \omega^{3(N-1)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \omega^{3(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

Wikipedia: DFT Matrix

The Fast Fourier Transform (FFT)

- Underlying principal: decompose the DFT matrix into mostly sparse matrices
- Reduces operations to $O(n\log_2 n)$
- There are a number of algorithms, the most popular being the Cooley - Tukey



Wikipedia: Cooley - Tukey FFT algorithm

The Cooley - Tukey Algorithm

- Show that $X_{N+k} = X_k$
- Each subproblem requires half the calculations
- Use SFT on suitably small sub-problems

$$
\begin{aligned}
X_{N+k} &= \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,(N+k)\,n\,/\,N} \\
&= \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,n} \cdot e^{-i\,2\pi\,k\,n\,/\,N} \\
&= \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N}
\end{aligned}
$$

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i\,2\pi\,k\,(2m)\,/\,N} + \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i\,2\pi\,k\,(2m+1)\,/\,N} \\
&= \sum_{m=0}^{N/2-1} x_{2m} \cdot e^{-i\,2\pi\,k\,m\,/\,(N/2)} + e^{-i\,2\pi\,k\,/\,N} \sum_{m=0}^{N/2-1} x_{2m+1} \cdot e^{-i\,2\pi\,k\,m\,/\,(N/2)}
\end{aligned}
$$

Pythonic Perambulations: Understanding the FFT Algorithm

Implementing Cooley - Tukey

- This FFT implementation is an order of magnitude faster than DFT, but still slower than NumPy's optimized *fft* routine.

```
import numpy as np
def DFT_slow(x):
    """Compute the discrete Fourier Transform of the 1D array x"""
    x = np.asarray(x, dtype=float)
    N = x.shape[0]
    n = np.arange(N)
    k = n.reshape((N, 1))
    M = np.exp(-2j * np.pi * k * n / N)
    return np.dot(M, x)
```

```
def FFT(x):
    """A recursive implementation of the 1D Cooley-Tukey FFT"""
    x = np.asarray(x, dtype=float)
    N = x.shape[0]

    if N % 2 > 0:
        raise ValueError("size of x must be a power of 2")
    elif N <= 32:  # this cutoff should be optimized
        return DFT_slow(x)
    else:
        X_even = FFT(x[::2])
        X_odd = FFT(x[1::2])
        factor = np.exp(-2j * np.pi * np.arange(N) / N)
        return np.concatenate([X_even + factor[:N / 2] * X_odd,
                               X_even + factor[N / 2:] * X_odd])
```

The Fastest Fourier Transform in the West (FFTW)

- The fastest free software implementation of FFT
- Comes as a collection of C routines
- Computes FT for 1 or more dimensions, arbitrary input size, and of real or complex data
- Works best for arrays sizes that are powers of 2, worst for large primes
- Chooses among various CT algorithms, or others for prime array sizes

FFTW.org

FFTW

- FFTW is released under a GNU General Public License and can be obtained at fftw.org
- FFTW is written in C, but also has Fortran and Ada interfaces
- There is also a python package known as pyFFTW
- Since FFTW “plans” the fastest transform in advance, one must supply data types, array sizes, precision, etc.

FFTW cont'd

- The FFTW planner minimizes execution time, not floating point operations
- The transform is done by an *executor* consisting of *codelets*
- The combination of *codelets* is determined by a *planner*

http://www.fftw.org/fftw-paper-icassp.pdf

pyFFTW - Implementation

- “A pythonic wrapper around FFTW”
- The FFTW library and NumPy are dependencies

```
import pyfftw

a = pyfftw.empty_aligned(128, dtype='complex128')
b = pyfftw.empty_aligned(128, dtype='complex128')

fft_object = pyfftw.FFTW(a, b)

c = pyfftw.empty_aligned(128, dtype='complex128')
ifft_object = pyfftw.FFTW(b, c, direction='FFTW_BACKWARD')
import numpy

# Generate some data
ar, ai = numpy.random.randn(2, 128)
a[:] = ar + 1j*ai

fft_a = fft_object()
```

→

```
>>> fft_a is b
True
>>> fft_a = fft_object()
>>> ifft_b = ifft_object()
>>> ifft_b is c
True
>>> numpy.allclose(a, c)
True
>>> a is c
False
```

Other FFT Algorithms

- Newer algorithms can identify the most weighted frequencies, and discard the “lightweights”
- Sufficiently sparse signals can be sampled randomly
- Hassanieh et al. (2012) propose an algorithm that offers improvement over FFT even as sparsity *k* approaches the input size *n*
- Other popular algorithms include: Prime - Factor, Bruun’s, Rader’s, and Bluestein’s

arXiv:1201.2501v1

FFTW & pyFFTW Links

- http://www.FFTW.org
- FFTW documentation: http://www.fftw.org/fftw3.pdf
- Python package index: https://pypi.python.org/pypi/pyFFTW
- pyFFTW tutorial: https://hgomersall.github.io/pyFFTW/sphinx/tutorial.html

References:

- [1] Wolfram MathWorld
- [2]Various Wikipedia figures
- [3]http://news.mit.edu/2012/faster-fourier-transforms-0118
- [4]http://nbviewer.jupyter.org/url/jakevdp.github.io/downloads/notebooks/UnderstandingTheFFT.ipynb
- [5]http://www.fftw.org
- [6]http://www.fftw.org/fftw-paper-icassp.pdf
- [7]http://www.askamathematician.com/2012/09/q-what-is-a-fourier-transform-what-is-it-used-for/
- [8]http://news.mit.edu/2012/faster-fourier-transforms-0118
- [9]arXiv:1201.2501v1