2) Fourier Basics

Sunday, June 18, 2017 6:01 PM

Despite our survey of basic audio-recording technology, we are not yet capable of understanding the popular .mp3 format, nor are we able to dabble in some of the most common audioprocessing techniques (which we will need to fingerprint our songs).

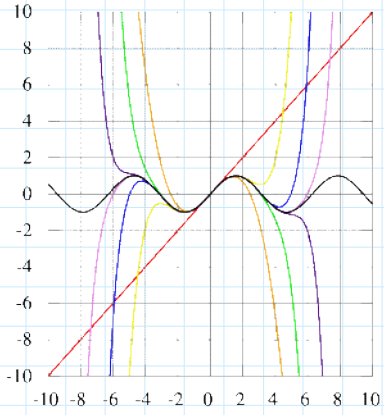
To achieve these tasks we must familiarize ourselves with Fourier Transforms

Series Representations of Functions

Taylor Series: we can represent functions as combinations of polynomials

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}$$

(This is an exact equality)



sine function and the first 1, 3, 5, 7, 9, 11, 13-term polynomials in its Taylor series.

Similarly, we can represent a function in the domain $[0, L)$ as a series of cosine and sine functions

(This assumes f is periodic in intervals of L)

$$f(x) = \sum_{k=0}^{\infty} \left[\alpha_k \cos\left(\frac{2\pi}{L} k x\right) + \beta_k \sin\left(\frac{2\pi}{L} k x\right)\right]$$

This is a Fourier series representation of f in $[0, L)$

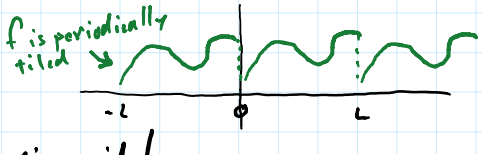
It is nice to use Euler's formula ($e^{ix} = \cos x + i \sin x$) to rewrite this:

$$f(x) = \sum_{k=-\infty}^{\infty} \gamma_k e^{i \frac{2\pi}{L} k x}, \quad \text{where } \gamma_k \in \mathbb{C}$$

γ_k are complex numbers called Fourier coefficients

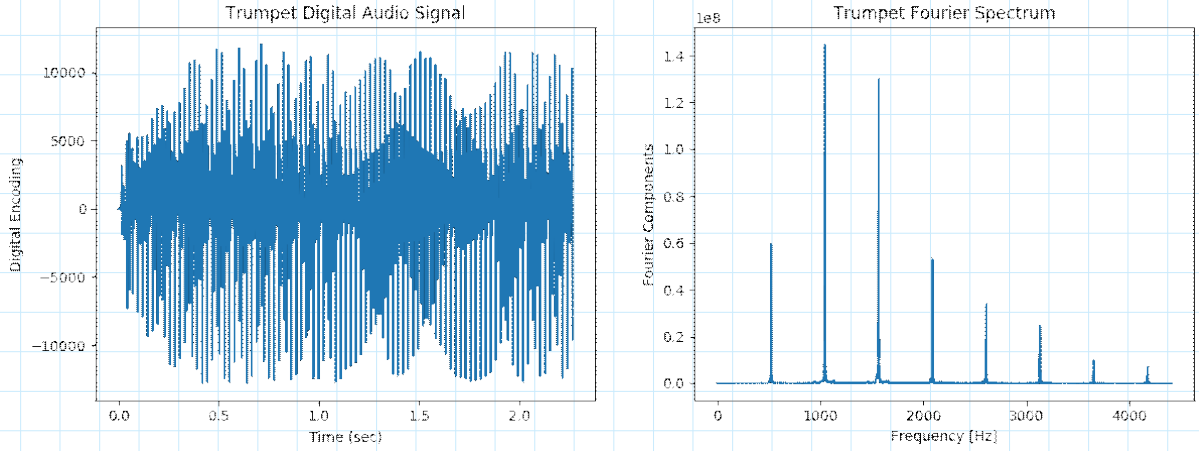
Fourier series representation of the function f in $[0, L)$. Presumes $f(x+L) = f(x)$.

Why Are We Studying Fourier Series?!



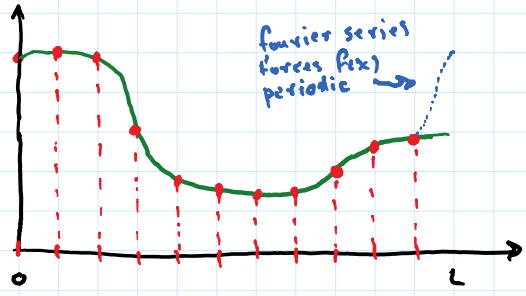
Recall that a pure note comes from a sinusoidal pressure wave (and thus a sinusoidal digital signal)

If we can represent our digital audio signal as a collection of sinusoidal waves, then we have decomposed it into pure notes, which we can analyze/manipulate!



The Discrete Fourier Transform

Rather than represent a continuous function as sine/cosine functions, we want to represent our sampled data as a Fourier Series.



We sample a function f N times at:

$$x_n = \frac{n}{N} L \qquad n = 0, 1, \ldots, N-1$$

Again the Fourier series demands that $f(x)$ is periodic:

$$f(x) = f(x+L)$$

Thus we have N samples of $f(x)$:

$$y_n = f(x_n)$$
$$x_n = \frac{n}{N} L$$
$$n = 0, 1, \ldots, N-1$$

We can represent these N samples in terms of N Fourier components (a.k.a Fourier coefficients)

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k e^{i \frac{2\pi}{L} k x_n}$$

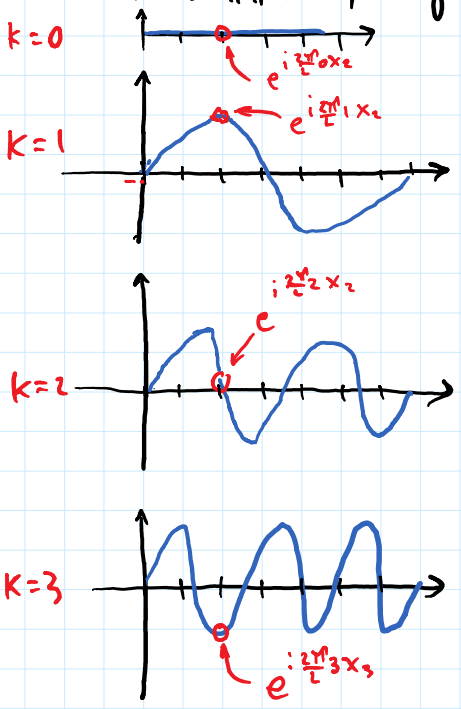
c_k is "how much" the wave of frequency $\frac{2\pi}{L}k$ contributes to the original data

Where c_k is the coefficient to the sinusoidal wave $e^{i \frac{2\pi}{L} k x}$ that oscillates w/ frequency $\frac{2\pi}{L}k$

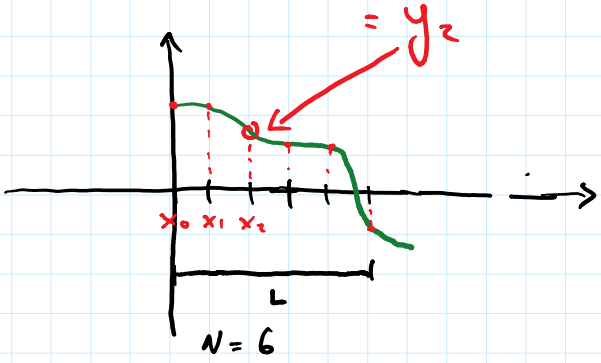
$k=0$

$$\left(e^{i\frac{2\pi}{L}0 x_2} \quad e^{i\frac{2\pi}{L}1 x_2} \quad e^{i\frac{2\pi}{L}2 x_2} \quad \ldots \quad e^{i\frac{2\pi}{L}6 x_2} \right)$$

that oscillates w/ frequency $\frac{2\pi}{L}k$



$$\frac{1}{7}\left(c_0 e^{i\frac{2\pi}{L}0x_2} + c_1 e^{i\frac{2\pi}{L}1x_2} + c_2 e^{i\frac{2\pi}{L}2x_2} + \cdots + c_6 e^{i\frac{2\pi}{L}6x_2}\right) = y_2$$



We use all N of our samples $\{y_n\}$ to compute each Fourier coefficient

$$c_k = \sum_{n=0}^{N-1} y_n e^{-i\frac{2\pi}{L}k x_n}$$

$k = 0, 1, \ldots, N-1$ $\qquad c_k \in \mathbb{C}$

c_k is a complex number

Using your N samples to compute a Fourier coefficient is called a Discrete Fourier Transform (Type I)

$$\{y_n\}_{n=0}^{N-1} \longrightarrow c_k$$

$$c_k = \sum_{n=0}^{N-1} y_n e^{-i\frac{2\pi}{L}k x_n}$$

$k \in [0, N-1]$

$x_n = \frac{n}{N}L \quad n \in [0, N-1]$

Using your N Fourier coefficients to compute a j sample of your data is called an Inverse Discrete Fourier Transform (Type I)

a U U sample of your data is called an **Inverse Discrete Fourier Transform (Type I)**

$$\{c_k\}_{k=0}^{N-1} \longrightarrow y_n$$

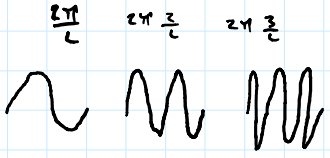
$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} c_k\, e^{i\frac{2\pi}{L}k x_n}$$

$n \in [0, N-1]$

$x_n = \frac{n}{N} L$

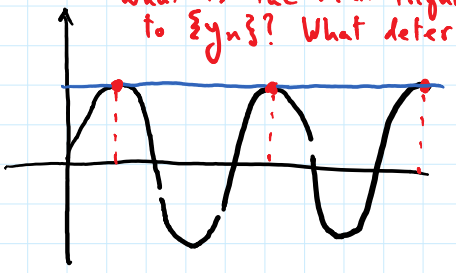
We can freely switch between $\{y_n\}$ and $\{c_k\}$ using these transforms, and we lose no information!

Question: What happens if you sample a rapidly-varying function sparsely?



Why do you think we sample audio at 44.1 kHz?

What is the max-frequency wave that can contribute to $\{y_n\}$? What determines this?



← wave and line are identical after sampling

$\frac{N-1}{N}$ is the max. frequency
Increasing # samples increases possible contributions

Simplification: $\{y_n\} \subseteq \mathbb{R}$

Complex conjugate: if $z = a + ib$ $(a, b \in \mathbb{R})$ then $z^* = a - ib$

→ and $(ab)^* = a^* b^*$

Given that $y_n^* = y_n$, prove that $c_{N-r} = c_r^*$ for $1 \le r \le \begin{cases} \frac{1}{2}N & (N \text{ even}) \\ \frac{1}{2}(N-1) & (N \text{ odd}) \end{cases}$

$$c_{N-r} = \sum_{n=0}^{N-1} y_n\, e^{-i\frac{2\pi(N-r)n}{N}}$$

$$= \sum_{n=0}^{N-1} y_n\, e^{-i2\pi n}\, e^{-i\frac{2\pi(-r)}{N}}$$

$$= \sum_{n=0}^{N-1} y_n^* \left(e^{-i\frac{2\pi r}{N}}\right)^*$$

$$= \left(\sum_{n=0}^{N-1} y_n\, e^{-i\frac{2\pi r n}{N}}\right)^*$$

$$= \left(\sum_{n=0}^{N-1} y_n e^{-i \frac{2\pi n r}{N}} \right)^*$$

$$= C_r^*$$

Thus if I compute C_1, then $C_{N-1} = C_1^*$ (if $\{y_n\} \subseteq \mathbb{R}$)

If N is even, compute

$\underbrace{C_0, C_1, \ldots, C_{\frac{N}{2}}}_{\frac{1}{2}N+1 \text{ coefficients}}$

If N is odd, compute

$\underbrace{C_0, C_1, \ldots, C_{\frac{N-1}{2}}}_{\frac{N+1}{2} \text{ coefficients}}$

cool python trick: $N//2+1 = \begin{cases} \frac{1}{2}N+1 & (N \text{ even}) \\ \frac{N+1}{2} & (N \text{ odd}) \end{cases}$

Class question: What is the complexity of this calculation?

↳ Each C_k requires $\propto N$ operations
N coeffs in total.
Thus $\mathcal{O}(N^2)$ ∴

Presently we can handle $\sim 10^9$ operations in seconds
thus using standard DFT, we can only handle
$\sim 32{,}000$ samples.

A 3min song @ 44100 Hz has $3 \times 60 \times 44100 = 7.938 \times 10^6$ samples!

It'd take us minutes to DFT 1 second of music! ↳ 10^3 operations $\sim$ 1 hour!

Fast Fourier Transform

A genius, probably alien being, named Freidrich Gauss developed a way to compute all N coeffs in an $\mathcal{O}(N \log N)$ algorithm. This algorithm is called the fast Fourier transform.

It is not an approximation - it exactly replicates the discrete Fourier transform

With $\mathcal{O}(N \log N)$ a 3 minute song can be transformed in tens of milliseconds instead of 20 minutes!

We will not delve into the FFT algorithm. We will use

numpy's fft algorithm though!

Overview

Studying a digital audio signal in its raw, digital form is unintuitive and messy.

We can represent an audio signal (and any other data, in general) as a combination of sine/cosine waves of various frequencies and amplitudes.

Taking a discrete Fourier transform takes N data points $\{y_n\}_{n=0}^{N-1}$ sampled at $x_n = \frac{n}{N} L$ $(0 \le n \le N-1)$ and produces N Fourier coefficients $\{c_k\}_{k=0}^{N-1}$

The coefficient c_k represents "how much" the wave $e^{i\frac{2\pi}{L}kx}$ contributes to the original signal.

Given $\{c_k\}_{k=0}^{N-1}$ we can exactly recover $\{y_n\}_{n=0}^{N-1}$ via an inverse discrete Fourier transform.

$$\{y_n\}_{n=0}^{N-1} \underset{\text{Inverse DFT}}{\overset{\text{DFT}}{\rightleftharpoons}} \{c_k\}_{k=0}^{N-1}$$

The DFT algorithm is $O(N^2)$. The equivalent FFT algorithm is $O(N \log N)$!

If $\{y_n\} \subseteq \mathbb{R}$ then $c_{N-r} = c_r^*$ $r = 1, 2, \ldots \{^{N/2}_{\frac{N-1}{2}}$

Thus we need only compute $N//2 + 1$ coeffs in this case

Do all 3 DFT notebooks

Independent reading: discrete cosine transforms

How do: JPEGs, MPEGs, mp3s use Fourier transforms to do compression?

How many bytes should a 16-bit digital encoding of a 3-minute song be?

$$16 \frac{\text{bits}}{\text{sample}} \times \frac{1 \text{ byte}}{8 \text{ bits}} \times \frac{44100 \text{ samples}}{1 \text{ second}} \times \frac{60 \text{ second}}{1 \text{ minute}} \times 3 \text{ minutes} = 15.9 \text{ Megabytes}$$

$\times 2$ channels

$= 31.7$ MB

High quality mp3 : 320 kbit/s → 7.2 MB