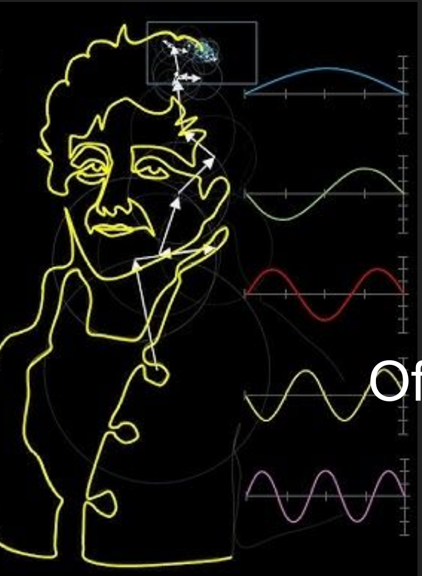
Fourier
PHYS4840

Analysis, Transform, Series…



github.com/nialljmiller/PHYS4840_Fourier

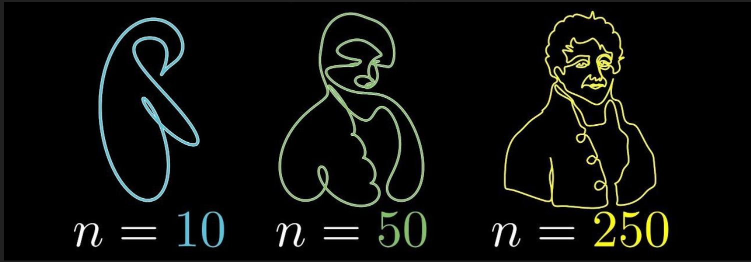
(This is currently the pinned repo on my page)

Dr Niall Miller

AG 404

Office hours - 2 hours before class or by request

Fourier



- Fourier series
  Using lots of trig functions to make any other function

- Fourier transform
  Using lots of trig to represent the frequency space of a complex signal

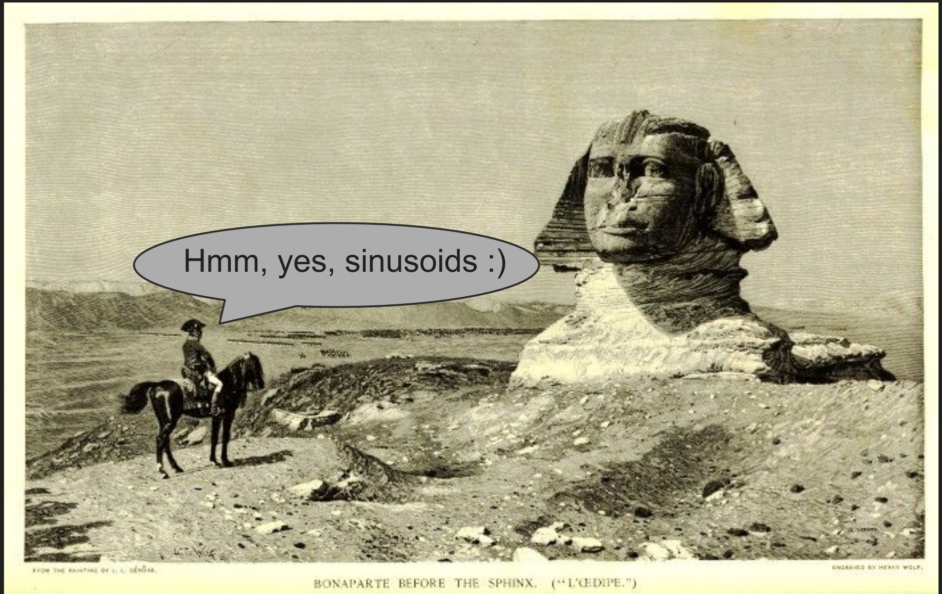
- Discrete Sine and Cosine transforms (jpegs)

- Fast Fourier Transforms
  Using lots of trig to represent the frequency space of a complex signal
      but on a computer and fast :)

And other things like reverse DFT, Gibbs, window functions…

Fourier

- What is Fourier analysis? (Signal analysis)
- Why is Fourier? (19th century scholars wanted to represent functions as sums of trig)
- Who is Fourier? (A French chap who was quite good with maths and also ran parts of Egypt for a bit and also Napoleon's m8)
- How is Fourier? (Dead)
- When is Fourier? (~ 1800s)



Fourier Series

- Any periodic function can be represented as a sum of sines and cosines
- So we can model complex signals with *lots* of simple components

F(x) = a0/2 + a1*cos(x) + b1*sin(x) + a2*cos(2x) + b2*sin(2x) + a3*cos(3x) + b3*sin(3x) + …

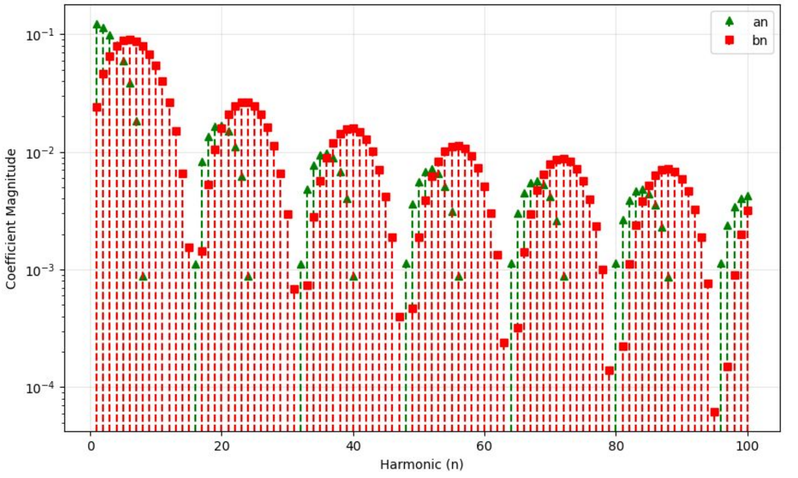
an and bn = Coefficients derived from :

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x), dx$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos(nx), dx$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx), dx$$

Values for an and bn fitting a pulse wave



Fourier Series maths

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} [a_n \cos(nx) + b_n \sin(nx)]$$

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x), dx$$

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx), dx$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx), dx$$

Fourier Series code

- Any periodic function can be represented as a sum of sines and cosines
- So we can model complex signals with *lots* of simple components

```
# Compute Fourier series coefficients
a0, an, bn = fs.compute_coefficients(wave, TERMS)

# Calculate the Fourier approximation
y_approx = fs.fourier_series_approximation(x, a0, an, bn)
```

Fourier Series code

```
# Compute Fourier series coefficients
a0, an, bn = fs.compute_coefficients(wave, TERMS)
```

```
a0 = compute_a0(func, period, num_points)
an = np.zeros(n_terms)
bn = np.zeros(n_terms)

for n in range(1, n_terms + 1):
    an[n-1] = compute_an(func, n, period, num_points)
    bn[n-1] = compute_bn(func, n, period, num_points)

return a0, an, bn
```

Fourier Series code

```python
# Compute Fourier series coefficients
a0, an, bn = fs.compute_coefficients(wave, TERMS)
```

$$a_0 = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x), dx$$

```python
def compute_a0(func, period=2*np.pi, num_points=1000):
    x = np.linspace(0, period, num_points)
    y = func(x)

    result = np.trapz(y, x)
    return (1 / period) * result
```

Fourier Series code

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx), dx$$

```python
def compute_bn(func, n, period=2*np.pi, num_points=1000):
    x = np.linspace(0, period, num_points)
    y = func(x)

    integrand = y * np.sin(2 * np.pi * n * x / period)
    result = np.trapz(integrand, x)
    return (2/period) * result
```

Fourier Series code

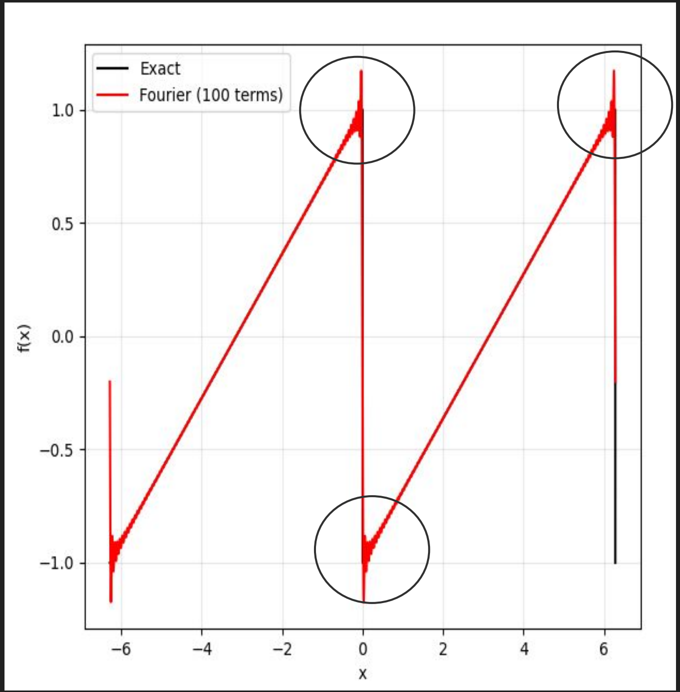
```python
# Calculate the Fourier approximation
y_approx = fs.fourier_series_approximation(x, a0, an, bn)
```

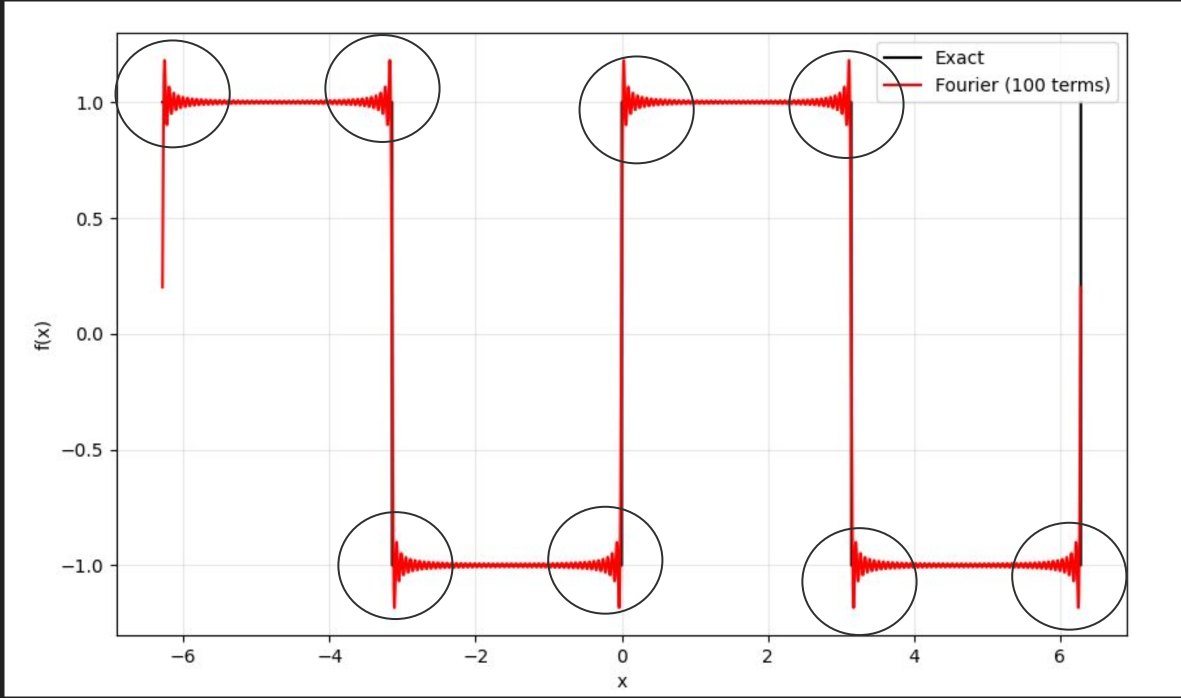
$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left[a_n \cos(nx) + b_n \sin(nx)\right]$$

```python
def fourier_series_approximation(x, a0, an, bn, period=2*np.pi):
    result = np.ones_like(x) * a0

    for n in range(1, len(an) + 1):
        result += an[n-1] * np.cos(2 * np.pi * n * x / period)
        result += bn[n-1] * np.sin(2 * np.pi * n * x / period)

    return result
```

Gibbs

The Gibbs phenomenon is the overshoot (or undershoot) that happens when you approximate a function with a discontinuity (like a step) using a finite number of terms in its Fourier series.



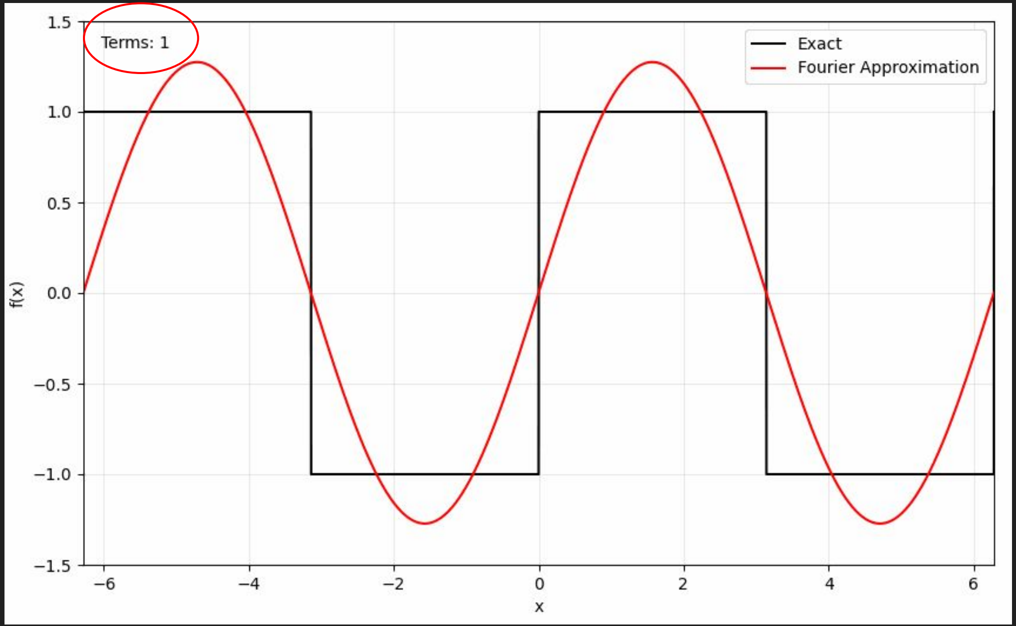


Gibbs

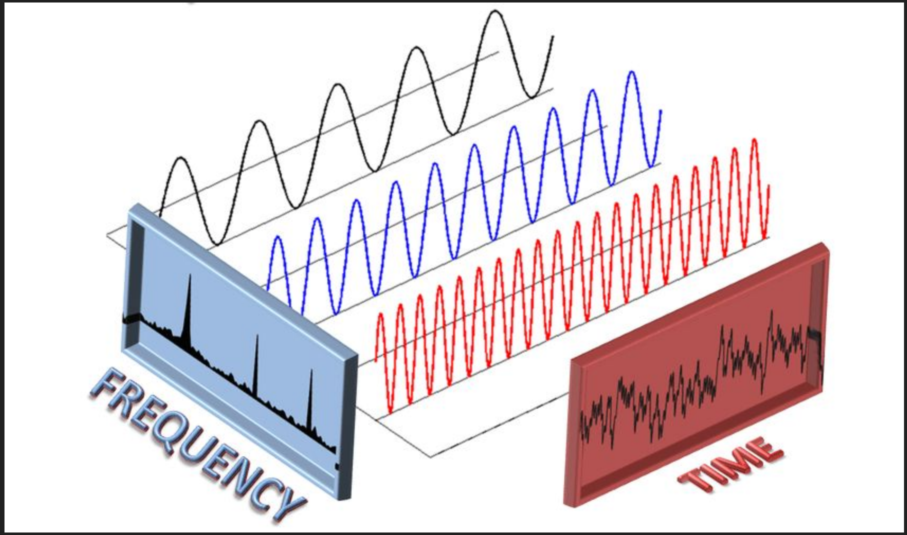
The oscillations don't go away as you add more terms — they just get narrower and more localized near the discontinuity.

The max overshoot settles around 9% of the jump, no matter how many terms you add.

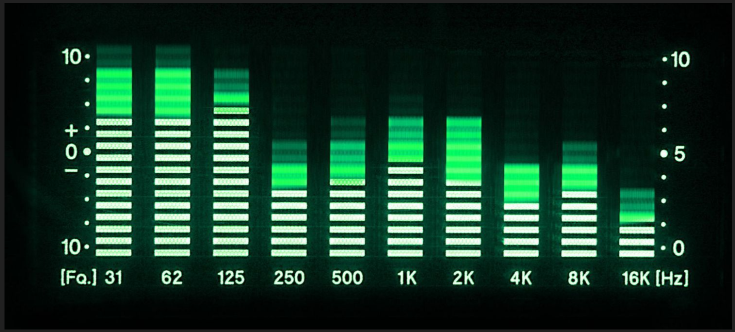
Even though the full Fourier series converges in the mean (L^2 norm), pointwise convergence fails at the jump



Fourier transform



- This is the one you know!
- All the audio visualisers work like this
- Has many applications
- <u>Is reversible</u>
- High compute complexity O(N^2)
- Sum of Fourier transforms = fourier transform of sum
- It is the frequency space representation of a signal



Fourier Series is nice if we know the periodicity

While the Fourier series applies to continuous, periodic functions
The Discrete Fourier Transform (DFT) is designed for sampled, finite-length signals.

The DFT transforms a sequence of N complex numbers into another sequence of complex numbers representing frequency components.

For a sequence $x_0, x_1, \ldots, x_{N-1}$, the DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi}{N} kn} \quad \text{for } k = 0, 1, \ldots, N-1$$

For a sequence $x_0, x_1, \ldots, x_{N-1}$, the DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi}{N} kn} \quad \text{for } k = 0, 1, \ldots, N-1$$

In terms of sines and cosines:

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi kn}{N}\right) - i \sum_{n=0}^{N-1} x_n \sin\left(\frac{2\pi kn}{N}\right)$$

Intuition Behind the Fourier Transform

- Think of it as measuring how much of each frequency component exists in the signal
- For each frequency ω, we multiply the signal by e^(-iωt) and integrate
- This is like computing a "correlation" between the signal and each frequency
- The result F(ω) gives amplitude and phase information at frequency ω

```
for k in range(N):
    for n in range(N):
        X[k] += x[n] * np.exp(-2j * np.pi * k * n / N)
```

Reversing the Fourier Transform

- Recall that Fourier transform is linear!
- We should be able to construct a signal from any FT spectra as long as our function is nicely behaved and we didn't undersample

For a sequence $x_0, x_1, \ldots, x_{N-1}$, the DFT is defined as:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i \frac{2\pi}{N} kn} \quad \text{for } k = 0, 1, \ldots, N-1$$

Reversing the Fourier Transform

- Recall that Fourier transform is linear!
- We should be able to construct a signal from any FT spectra as long as our function is nicely behaved and we didn't undersample

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{i \frac{2\pi}{N} kn} \quad \text{for } n = 0, 1, \ldots, N-1$$

Lets code!

github.com/nialljmiller/PHYS4840_Fourier

fs_demo.py