

# Fourier Analysis

- One of the most useful tools in theoretical and computational physics.
- Wide range of physics and math applications:
  - Spectral analysis
  - Signal processing
  - Solving differential equations (ODE and PDE)
  - Interpolation, and more.
- Widely used in modern technology (e.g. sound, image and video compression).

# Fourier Analysis: Motivation

- Taylor series expansion of a function is a very useful and powerful tool, as we have seen throughout this course (and many others).
- However, it has certain important limitations:
  - Not suitable for periodic functions (unique point).
  - Cannot handle discontinuities (all derivatives should exist).
- Both issues addressed by the Fourier series representation of a function.

# Fourier series

- The origin of the Fourier analysis lies in Fourier series expansion.
- Any periodic function with a finite number of discontinuities on a finite interval  $0 < x < L$  can be represented in Fourier series:

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\pi kx}{L}\right) + \sum_{k=1}^{\infty} \beta_k \sin\left(\frac{2\pi kx}{L}\right) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i\frac{2\pi kx}{L}\right)$$

where we set:  
and used that:

$$\gamma_k = \begin{cases} \frac{1}{2}(\alpha_{-k} + i\beta_{-k}), & k < 0 \\ \alpha_0, & k = 0 \\ \frac{1}{2}(\alpha_k - i\beta_k), & k > 0 \end{cases}$$

$$e^{i\theta} = \cos \theta + i \sin \theta \iff \cos \theta = \frac{1}{2}(e^{-i\theta} + e^{i\theta}), \sin \theta = \frac{i}{2}(e^{-i\theta} - e^{i\theta})$$

# Fourier series

- The original function  $f(t)$  must satisfy the following conditions:
  - Any periodic function  $f(t)$  can be represented by a Fourier series if it satisfies the following conditions:
    - **Dirichlet's theorem:** If  $f(t)$  is periodic of period  $2\pi$ , if for  $-\pi < t < \pi$  the function  $f(t)$  has a finite number of maximum and minimum values and a finite number of discontinuities, and if  $\int_{-\pi}^{\pi} f(t) dt$  is finite, then the Fourier series converges to  $f(t)$  at all points where  $f(t)$  is continuous, and at jump-points it converges to the arithmetic mean of the right-hand and left-hand limits of the function.

$$f(x) = \sum_{k=0}^{\infty} \alpha_k \cos\left(\frac{2\pi kx}{L}\right) + \sum_{k=1}^{\infty} \beta_k \sin\left(\frac{2\pi kx}{L}\right) = \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i\frac{2\pi kx}{L}\right)$$

where we set:

$$\gamma_k = \begin{cases} \frac{1}{2}(\alpha_{-k} + i\beta_{-k}), & k < 0 \\ \alpha_0, & k = 0 \\ \frac{1}{2}(\alpha_k - i\beta_k), & k > 0 \end{cases}$$

and used that:

$$e^{i\theta} = \cos \theta + i \sin \theta \iff \cos \theta = \frac{1}{2}(e^{-i\theta} + e^{i\theta}), \sin \theta = \frac{i}{2}(e^{-i\theta} - e^{i\theta})$$

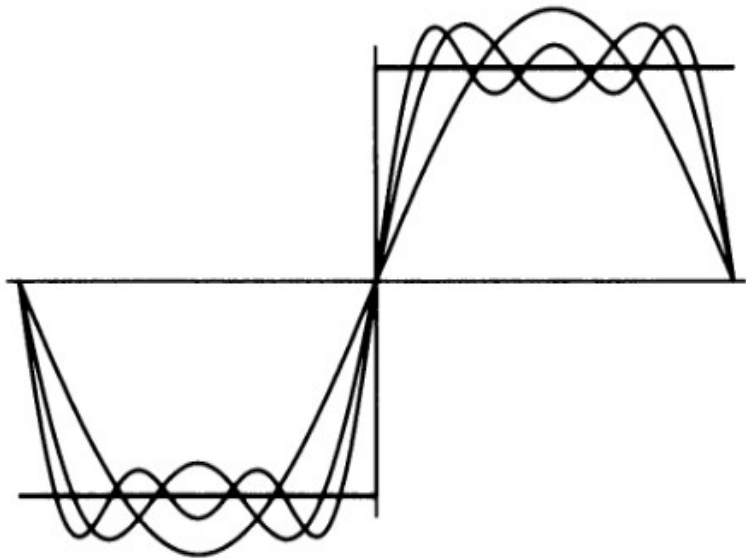
# Fourier series (cont.)

- How do we calculate  $\gamma_k$ ? (Check the math!)

$$\gamma_k = \frac{1}{L} \int_0^L f(x) \exp \left( -i \frac{2\pi k x}{L} \right) dx$$

- Notes:
  - Fourier series can also represent discontinuous functions (unlike e.g. Taylor series)
  - What about non-periodic functions? → Make them periodic!  
→ Some examples next.

# Example: step function



This series of successive approximations provides the best possible fit to the original function, improving with every successive term.

$$f(t) = \begin{cases} -1, & t < 0 \\ +1, & t > 0 \end{cases}$$

$$\begin{aligned} b_n &= \frac{1}{\pi} \int_{-\pi}^0 (-1) \sin nt \, dt + \frac{1}{\pi} \int_0^{\pi} (+1) \sin nt \, dt \\ &= \frac{2}{\pi} \int_0^{\pi} \sin nt \, dt \\ &= \frac{2}{n\pi} [1 - \cos n\pi] \\ &= \begin{cases} 0, & n = 2, 4, 6, \dots \\ 4/n\pi, & n = 1, 3, 5, \dots \end{cases} \end{aligned}$$

# How do we make a function periodic?

- We want to construct series for a function over  $0 < x < L$ .
- Simply repeat it outside that interval  $\rightarrow$  becomes periodic (remember jumps are OK!)
- Data outside  $0 < x < L$  is discarded (we do not try to expand it in series there, so it is not relevant).

# Fourier transform

- The continuous version of the Fourier series is the Fourier transform:

$$\mathcal{F}[f(t)] = g(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \quad \text{Forward}$$

$$\mathcal{F}^{-1}[g(\omega)] = f(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} g(\omega) e^{i\omega t} d\omega \quad \text{Inverse}$$



# Properties of the Fourier transform

$$\mathcal{F}[f_1 + f_2] = \mathcal{F}[f_1] + \mathcal{F}[f_2]$$

→ linear

$$\mathcal{F}[f(\alpha t)] = \frac{1}{|\alpha|} g\left(\frac{\omega}{\alpha}\right)$$

$$\mathcal{F}^{-1}[g(\beta\omega)] = \frac{1}{|\beta|} f\left(\frac{t}{\beta}\right)$$

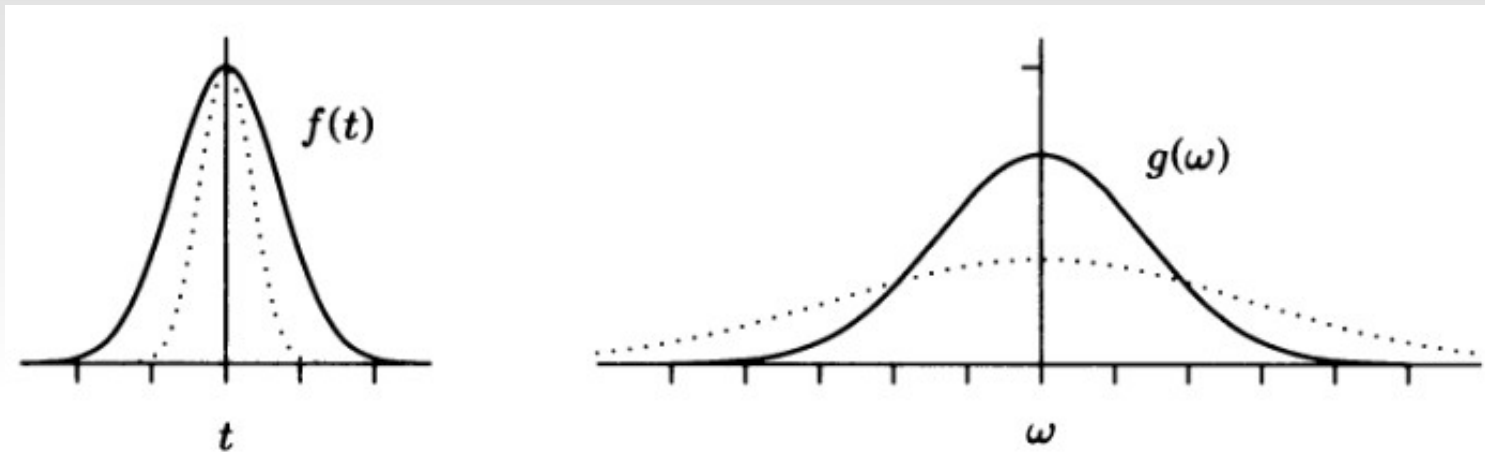
→ scaling relations

$$\mathcal{F}[f(t - t_0)] = e^{-i\omega t_0} g(\omega)$$

$$\mathcal{F}^{-1}[g(\omega - \omega_0)] = e^{i\omega_0 t} f(t)$$

→ phase shifts

# Fourier transform: properties



- The broader  $f(t)$  is, the narrower  $g(\omega)$  and vice-versa
- Related to the Heisenberg's Uncertainty Principle in Quantum Mechanics.

$$\Delta x \Delta p > \hbar/2\pi$$

# Discrete Fourier Transform

$$\gamma_k = \frac{1}{L} \int_0^L f(x) \exp \left( -i \frac{2\pi kx}{L} \right) dx$$

Q: How do we calculate  $\gamma_k$  in practice?

- For some  $f(x)$  one can do this exactly, but typically that is not possible.
- Use numerical integration, instead.

# Discrete Fourier Transform (cont.)

- Applying e.g. the trapezoidal rule, we have:

$$\gamma_k = \frac{1}{L} \frac{L}{N} \left[ \frac{1}{2} f(0) + \frac{1}{2} f(L) + \sum_{n=1}^{N-1} f(x_n) \exp \left( -i \frac{2\pi k x_n}{L} \right) \right]$$

where  $x_n = nL/N$ .

- Taking into account that  $f(0)=f(L)$  (periodic), we get:

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} f(x_n) \exp \left( -i \frac{2\pi k x_n}{L} \right)$$

# Discrete Fourier Transform (cont.)

- The typical situation is to have sampled signal  $x_n \rightarrow y_n$  (e.g. in a lab experiment measuring something at regular time intervals), giving:

$$\gamma_k = \frac{1}{N} \sum_{n=0}^{N-1} y_n \exp \left( -i \frac{2\pi k x_n}{L} \right) \quad \leftarrow c_k : \text{Fourier coefficients}$$

- This is called **Discrete Fourier Transform** (DFT) of the samples  $y_n$

# Discrete Fourier Transform (DFT)

- Even though the trapezoidal rule we used to derive DFT is an approximation to the integrals, the DFT is in certain sense **exact**.
- To show this, recall the geometric series:

$$\sum_{k=0}^{N-1} a^k = \frac{1 - a^N}{1 - a}, \text{ and set } a = e^{i2\pi m/N}$$

which gives (why?):

$$\sum_{k=0}^{N-1} e^{i2\pi km/N} = \frac{1 - e^{i2\pi m}}{1 - e^{i2\pi m/N}} = \begin{cases} 0, m \neq 0 \\ N, m = 0 \end{cases}$$

# Discrete Fourier Transform (DFT)

- Coming back to DFT, consider the sum:

$$\begin{aligned}\sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right) &= \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} y_m \exp\left(-i\frac{2\pi km}{N}\right) \exp\left(i\frac{2\pi kn}{N}\right) \\ &= \sum_{m=0}^{N-1} y_m \sum_{k=0}^{N-1} \exp\left(i\frac{2\pi k(n-m)}{N}\right)\end{aligned}$$

but the last sum is the same we just calculated above, (if we replace  $m \rightarrow n-m$ ), i.e. is equal to  $N$ , so:

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \exp\left(i\frac{2\pi kn}{N}\right)$$

# Discrete Fourier Transform (DFT)

$$y_n = \frac{1}{N} \sum_{k=0}^{N-1} c_k \exp \left( i \frac{2\pi kn}{N} \right)$$

- This is the inverse DFT, which means that given the Fourier coefficients  $c_k$  we can recover the original data points  $y_n$  **exactly** (with arbitrary precision), i.e.  $c_k$  and  $y_n$  are equivalent datasets.



# Discrete Fourier Transform (DFT)

- DFT is similar to the Fourier series, but not the same:
  - Sum is not infinite
  - Most importantly, function is given **only at the sample points  $y_n$**  and nowhere else – in-between the samples function could be doing anything and DFT will be **the same**.

Therefore, for proper representation we need good enough sampling and a reasonably smooth function.

# Discrete Fourier Transform (DFT)

- This is also related to so-called **aliasing**, as follows:
  - The highest frequency that DFT can sample is  $(N-1)\Delta\omega$ ,  $\Delta\omega=2\pi/T$
  - In fact, the highest one is really  $(N/2)\Delta\omega$  since the frequencies  $(N/2)\Delta\omega$  to  $(N-1)\Delta\omega$  are the same as  $-(N/2)\Delta\omega$  to  $-\Delta\omega$ .
  - If  $f(t)$  is periodic in  $t$ , its Fourier transform is periodic in  $\omega$ .
- This highest frequency that can be represented is called Nyquist frequency:  $\omega_{\text{Nyquist}} = (N/2)\Delta\omega$

# Discrete Fourier Transform (DFT)

- The aliasing and the Nyquist frequency have a simple physical meaning:

e.g. if we have a periodic function with period  $T=1\text{s}$  and we sample it every  $2\text{s}$  it will appear to be constant.

Q: What happens if we sample it every  $1.5\text{s}$ ?

- Therefore, we need **sufficiently frequent sampling** for faithful representation.

# DFT: properties

All results up to now apply equally whether  $f(x)$  is real or complex function. However, esp. in physics,  $f(x)$  is typically real, e.g. a signal we measure. This yields some simplifications:

Let  $y_n$  be real numbers and consider  $c_k$  for some  $N/2 < k < N$ . Then,  $k = N - r$ , where  $1 \leq r < N/2$  and we have:

$$\begin{aligned} c_{N-r} &= \sum_{n=0}^{N-1} y_n \exp \left( -i \frac{2\pi(N-r)n}{N} \right) \\ &= \sum_{n=0}^{N-1} y_n \exp(-i2\pi n) \exp \left( i \frac{2\pi r n}{N} \right) \\ &= \sum_{n=0}^{N-1} y_n \exp \left( i \frac{2\pi r n}{N} \right) = c_r^* \end{aligned}$$

(using  $e^{-i2\pi n} = 1$  and  $y_n$  being real)

→  $c_{N-r}$  and  $c_r$  are complex conjugates  
→ we really need only calculate  $c_k$  for  $0 \leq k \leq N/2$

If  $f(x)$  is complex we still need to calculate all  $c_k$ 's.

# Sample DFT Python code and usage

```
from numpy import zeros,loadtxt
from pylab import plot,xlim,show
from cmath import exp,pi
```

← cmath – same as math, but can handle complex numbers

```
def dft(y):
```

← DFT code

```
    N = len(y)
```

```
    c = zeros(N//2+1,complex)
```

```
    for k in range(N//2+1):
```

```
        for n in range(N):
```

```
            c[k] += y[n]*exp(-2j*pi*k*n/N)
```

```
    return c
```

usage: read in a text file

containing a signal to be analysed

```
y = loadtxt("pitch.txt",float)
```

```
c = dft(y)
```

← calculate and plot the Fourier coefficients

```
plot(abs(c))
```

```
xlim(0,500)
```

```
show()
```

Note: this code is simple, but **computationally expensive** and thus slow (why?)

# Fourier transforms in 2D and 3D

- How do we do higher-dimensional (2D, 3D) Fourier transforms?
- Useful in e.g. image processing, solving equations in 2D, 3D domains.
  - just do transforms in each direction in turn.
- Let's consider  $M \times N$  grid of samples  $y_{mn}$ . We first transform each of the  $M$  rows:
$$c'_{ml} = \sum_{n=0}^{N-1} y_{mn} \exp \left( -i \frac{2\pi l n}{N} \right)$$
  - $N$  coefficients for each row  $m$ , one for each  $l$ .

# Fourier transforms in 2D and 3D

- Next, we take  $l^{\text{th}}$  coefficient in each of the  $M$  rows and transform these:

$$c_{kl} = \sum_{m=0}^{M-1} c'_{mn} \exp \left( -i \frac{2\pi km}{N} \right)$$

- The two steps could be combined to give:

$$c_{kl} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} y_{mn} \exp \left[ -i2\pi \left( \frac{km}{M} + \frac{ln}{N} \right) \right]$$

with corresponding inverse transform:

$$y_{mn} = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} c'_{kl} \exp \left[ -i2\pi \left( \frac{km}{M} + \frac{ln}{N} \right) \right]$$

# Physical meaning of the Fourier transform

- The Fourier transform breaks each function into a sum of simple sinusoidal waves (real or complex) – these could be spatial ones or temporal (or both) called **modes**.
- The magnitude (absolute value) of the coefficients of those waves give how important each particular frequency is → spectral analysis of the signal/image, etc.
- These typically include main frequency/frequencies, harmonics and noise.



# Fast Fourier Transform (FFT)

- As seen, Fourier transforms are quite useful in a wide range of physical and technical applications.
- However, their calculation is quite computationally expensive and thus slow, even in the case of DFT (itself much cheaper than the full FT).

$$c_k = \sum_{n=0}^{N-1} y_n \exp \left( -i \frac{2\pi kn}{N} \right)$$

→ calculating each Fourier coefficient (single point in Fourier/frequency space!) takes N operations, or  $N \times N = N^2$  for all N points.

# Fast Fourier Transform (FFT)

- In 2D or 3D this is still more expensive – number of operations scales as  $N^4$  and  $N^6$ , respectively.
- How large problems are practical to do? Depends on your computer, but as a rule of thumb let's say  $10^9$  operations are reasonable. So we can do about 30,000 points – i.e. not too many, about 1 second worth of music.
- In 2D/3D things are still much worse, we can do just 178/32 samples per dimension.

What can we do to improve this?

# Fast Fourier Transform (FFT)

- Need a better algorithm! Fortunately it exists – FFT
- History note: first derived by Gauss in 1805, reinvented and improved several times since then.
- Method is simplest for  $N=2^m$  points, so let's consider that:
  - Divide the points into odd and even (always possible)
  - The even terms give:

$$E_k = \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp \left( -i \frac{2\pi k(2r)}{N} \right) = \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp \left( -i \frac{2\pi k r}{N/2} \right)$$

→ again a DFT, but with just  $N/2$  points.

# Fast Fourier Transform (FFT)

- FFT algorithm (cont.)

$$E_k = \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp \left( -i \frac{2\pi k(2r)}{N} \right) = \sum_{r=0}^{\frac{1}{2}N-1} y_{2r} \exp \left( -i \frac{2\pi kr}{N/2} \right) \rightarrow \text{even}$$

- The odd terms give:

$$\begin{aligned} & \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp \left( -i \frac{2\pi k(2r+1)}{N} \right) = \\ & e^{-i2\pi k/N} \sum_{r=0}^{\frac{1}{2}N-1} y_{2r+1} \exp \left( -i \frac{2\pi kr}{N/2} \right) = e^{-i2\pi k/N} O_k \end{aligned}$$

→ again a DFT with  $N/2$  points, times a phase factor.

- We now can do the DFT in  $2(N/2)^2$  operations!

# Fast Fourier Transform (FFT)

- So, we halved the number of operations to do!

$$c_k = E_k + e^{-i2\pi k/N} O_k$$

- But why stop here? Do this again for each of the DFTs, and again, ... until we are left with just a single point (whose transform is the point itself). Can do this because there are  $2^m$  points in total.
- How many operations we need to do now?
  - $N \log_2 N$  – way better than the original  $N^2$ !
  - e.g. for  $N \log_2 N = 10^9$  we have  $N \sim 40$  million.
- Even 2D and 3D FTs become feasible now. FFTs can be done also for non- $2^m$  sizes (but algebra is quite tedious).

# DFTs and FFTs in Python

- Provided by module `numpy.fft`
- Includes a variety of functions (check help),

e.g. `rfft` → 1D real DFT (using FFT)

`irfft` → inverse of `rfft`

`fftpack.ffn` → N-dimensional DFT (using FFT)

- Sample usage:

```
from numpy import array
```

```
y=array([0.,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9],float) ← input data
```

```
c=rfft(y) ← forward FFT
```

```
print(c)
```

```
[ 4.5+0.j      -0.5+1.53884177j  -0.5+0.68819096j  -0.5+0.36327126j  
 -0.5+0.16245985j  -0.5+0.j      ] ← Fourier coefficients (why only 6?)
```

```
y1=irfft(c) ← inverse FFT
```

```
print(y1)
```

```
[ 0.  0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9]
```