Lecture 5: Fourier Transforms

Brief Review (hopefully)

Slide Type

If you need a more detailed refresher on what Fourier series are, the YouTube Channel 3Blue1Brown by Grant Sanderson has this video: https://youtu.be/r6sGWTCMz2k

In computational physics, we often compute Fourier series as a way to compute Fourier transforms. For a refresher or introduction about Fourier transforms, the same channel also has a video about it: https://youtu.be/spUNpyF58BY

Slide Type

Fourier Series

Slide Type Slide

In 1822, Joseph Fourier wanted to compute temperature distributions in objects, based on the heat flux equation $\vec{\phi} = \kappa \vec{\nabla} T$, where $\vec{\phi}$ is the flux vector, κ the heat diffusivity and T is the temperature. He did so by finding a way to express periodic functions as linear combinations of sines and cosines.

We can write any periodic function f with period L on the interval $[0, L]$ as a "Fourier series".

Slide Type Sub-Slide

$$\begin{aligned} f(x) &= \sum_{k=0}^{\infty}\left[\alpha_k \cos\left(\frac{2\pi k x}{L}\right) + \beta_k \sin\left(\frac{2\pi k x}{L}\right)\right] \\ &= \sum_{k=-\infty}^{\infty} \gamma_k \exp\left(i\frac{2\pi k x}{L}\right), \end{aligned}$$

with

$$\begin{aligned} \gamma_k &= \frac{\alpha_{-k} + i\beta_{-k}}{2} && \text{if} \quad k < 0, \\ \gamma_k &= \alpha_0 && \text{if} \quad k = 0, \\ \gamma_k &= \frac{\alpha_k - i\beta_k}{2} && \text{if} \quad k > 0, \end{aligned}$$

and

$$\forall k, \quad \gamma_k = \frac{1}{L}\int_0^L f(x)\exp\left(-i\frac{2\pi k x}{L}\right)dx \quad \text{from orthogonality of sin/cos functions.}$$

Slide Type Sub-Slide

Orthogonality of the sine functions:

$$\begin{aligned} \int_0^L \sin\left(\frac{\pi n x}{L}\right)\sin\left(\frac{\pi m x}{L}\right)dx &= \frac{L}{2}\delta_{nm}, \\ \int_0^L \cos\left(\frac{\pi n x}{L}\right)\cos\left(\frac{\pi m x}{L}\right)dx &= \frac{L}{2}\delta_{nm}, \\ \int_0^L \sin\left(\frac{\pi n x}{L}\right)\cos\left(\frac{\pi m x}{L}\right)dx &= 0 \end{aligned}$$

Slide Type Sub-Slide

For non-periodic functions, we can repeat the function over the portion of interest and discard the rest
Newman's fig. 7.1
Solid grey line = actual function, Solid black line = function bracketed to the interval $[0, L]$

Dashed lines = bracketed function replicated over other intervals to make it periodic.

Slide Type

Fourier Transforms

Slide Type Slide

What if $L \to \infty$? Then, separation between wave numbers or frequencies tend to zero:

$$\frac{2\pi(k+1)x}{L} - \frac{2\pi k x}{L} = \frac{2\pi x}{L} \to 0.$$

And the discrete sums turn into integrals:

$$f(x) = \sum_{k=-\infty}^{\infty} \gamma_k \mathrm{e}^{2\pi i \nu_k x} \to \int_{-\infty}^{\infty} \hat{f}(\nu)\mathrm{e}^{-2\pi i \nu x}\mathrm{d}\nu,$$

with $\nu_k = k/L$ (discrete) or ν (continuous) the frequency of each Fourier component, $\hat{f}$ the **Fourier transform** of f.

Just like we could retrieve the γ_k's from f with the integral formulas above, we can invert the Fourier transform:

$$\hat{f}(\nu) = \int_{-\infty}^{\infty} f(x)\mathrm{e}^{2\pi i \nu x}\mathrm{d}x.$$

Slide Type

Discrete Fourier Transform

Slide Type Slide

Discretizing the Fourier transform operation: $\mathrm{d}x \to \Delta_x$ finite, $x \to x_k = x_0 + k\Delta_x$ with $0 \leq k < N$

$$\hat{f}(\nu) = \int_{-\infty}^{\infty} f(x)\mathrm{e}^{2\pi i \nu x}\mathrm{d}x \approx \sum_{k=0}^{N-1} f(x_k)\mathrm{e}^{2i\pi \nu x_k}\Delta_x.$$

- Note how discretizing also requires bounding the interval.
- Only a Riemann sum but it illustrates the properties well enough. And it is actually how we compute FTs numerically.
- Also, the frequency is discretized: we usually use $\nu_n = n/N$.

The expression above then becomes a Fourier series: **When we compute Fourier transforms, we actually compute Fourier series**! It is the user's job to know how to interpret a Fourier series as a Fourier transform.

Careful however: the formula above is not the exact expression for how to compute Fourier transforms numerically (patience).

Choice of interval

Slide Type Sub-Slide

Mechanically, we turn any continuous function defined on the real axis as a periodic function on the interval $0 \leq x < N\Delta_x$. Choose your interval wisely if you really need the whole thing!

- Periodic function: take integer number of periods. One is enough in theory but you never know (slight aperiodicity, noise...)
- Function that decays to infinity: interval wide enough that the function has almost completely decayed at edges.
- Function that keeps doing interesting stuff for ever (e.g.\ stochastic series): choose it wide enough that you encapsulate enough statistics, and know what you're not capturing.

Want to capture high frequencies? Make Δ_x smaller.

Want to capture low frequencies? Make the interval ($N\Delta_x$) longer.

Slide Type

DFT Implementation

Slide Type Sub-Slide

Now let's think about the integrals used for obtaining the Fourier coefficients γ_k's.

- We divide $[0, L]$ up into N segments and use the trapezoidal rule and periodicity of the function:

$$\begin{aligned}\gamma_k &= \frac{1}{L}\int_0^L f(x)\exp\left(-i\frac{2\pi kx}{L}\right)dx \\ &\approx \frac{1}{L}\frac{L}{N}\left[\frac{1}{2}f(0)+\frac{1}{2}f(L)+\sum_{n=1}^{N-1} f(x_n)\exp\left(-i\frac{2\pi kx_n}{L}\right)\right] \\ &= \frac{1}{N}\left[\sum_{n=0}^{N-1} f(x_n)\exp\left(-i\frac{2\pi kn}{N}\right)\right] \quad \text{because } f(0)=f(L) \text{ and } \frac{x_n}{L}=\frac{n}{N}.\end{aligned}$$

Slide Type Sub-Slide

- Now define the Discrete Fourier Transform (DFT) as follows:

$$y_k = f(x_k); \qquad c_k = N\gamma_k;$$

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

- Would it be better if we used a more precise integration function? No, because with the expression above we use the properties of the $\exp(-2i\pi kn/N)$ series to obtain two algorithms:
  - inverse DFT
  - Fast Fourier Transform

Slide Type Sub-Slide

- Note how, for $y(x) \in \mathbb{R}$,

$$c_{N-k} = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi(N-k)n}{N}\right) = \sum_{n=0}^{N-1} y_n \underbrace{e^{-i2\pi n}}_{=1} \exp\left(+i\frac{2\pi kn}{N}\right) = c_k^*,$$

  or, in short, $c_{N-k} = c_k^*$.
  - If $y(x) \in \mathbb{R}$, then we only need $N/2+1$ (N even) or $(N+1)/2$ (N odd) points to actually know the DFT.
  - Python's `N//2+1` will give you this number.

Slide Type Fragment

```python
N = 9  # increase it
N//2 + 1
```

5

Slide Type

Discrete sine and cosine Fourier transforms

Slide Type Slide

Recall

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

- If f odd (i.e., $f(-x) = -f(x)$), then $\forall k$, $\alpha_k = 0$,
- If f even (i.e., $f(-x) = +f(x)$), then $\forall k$, $\beta_k = 0$.

If you know that function has one of these properties, computing only 1/2 coefficients saves time and memory.

Slide Type

Inverse Discrete Fourier transform

Slide Type Slide

The inverse DFT follows from the definition of the DFT and properties of exponential sums.

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

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

Slide Type Sub-Slide

We can simplify using geometric series:

$$\forall a \in \mathbb{C}, \quad \sum_{k=0}^{N-1} a^k = \frac{1-a^N}{1-a}.$$

Using $a = \exp(+i2\pi m/N)$,

$$\sum_{k=0}^{N-1} \exp\left(+i\frac{2\pi km}{N}\right) = \frac{1-\exp(i2\pi m)}{1-\exp(i2\pi m/N)}.$$

$m \in \mathbb{N} \Rightarrow 1-\exp(i2\pi m) = 0$. Two possibilities for denominator:

- If m not multiple of N, denom. $\neq 0 \;\Rightarrow \sum_{k=0}^{N-1} \cdots = 0$.
- If m is 0 or multiple of N, then $1-\exp(i2\pi m/N) = 0$ also!

  0 divided by 0, we need to step back:

  $$\sum_{k=0}^{N-1} \exp(+i2\pi kp) = \sum_{k=0}^{N-1} 1 = N.$$

Slide Type Sub-Slide

Therefore, the innermost sum of the previous double-sum is N when $p = n$, and zero otherwise:

$$\sum_{p=0}^{N-1} y_p \sum_{k=0}^{N-1} \exp\left(i\frac{2\pi k(n-p)}{N}\right) = y_n \sum_{k=0}^{N-1} \exp\left(i\frac{2\pi k(n-n)}{N}\right) = N y_n.$$

Divide the first and last expressions above, and you retrieve the iDFT expression framed above.

Note there is no approximation error here, it's an exact result (up to machine precision)! A kind of double-compensation of errors happened, but it works, thanks to the trapezoidal rule!

Fast Fourier Transforms (FFTs)

Slide Type Slide

Can we speed up the DFT? Recall:

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

The `dft` snippet below requires $\approx N^2$ "unit" operations.

Slide Type Fragment

```
for k in range(N//2+1):
    for n in range(N):
        c[k] += y[n]*np.exp(-2j*np.pi*k*n/N)
```

Slide Type Fragment

- Your computer can afford a billion operations? Your limit is $N \sim 32{,}000$: too few to be practical.
- Fast Fourier Transform (FFT) overcomes this (Cooley & Tukey 1960's, first found by Gauss 1805).
- There are alternative implementations, but we present the "historical" version.

Slide Type

Divide-and-conquer strategy

Slide Type Sub-Slide

Assume $N = 2^M$ (other prime numbers in the decomposition are possible, but they will slow down the execution).

$$\text{Split} \qquad c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) = E_k + \omega^k O_k,$$

with

$$E_k = \sum_{p=0}^{N/2-1} y_{2p} \exp\left(-i\frac{2p\pi k}{N/2}\right) \quad \text{the even indices } (n = 2p),$$

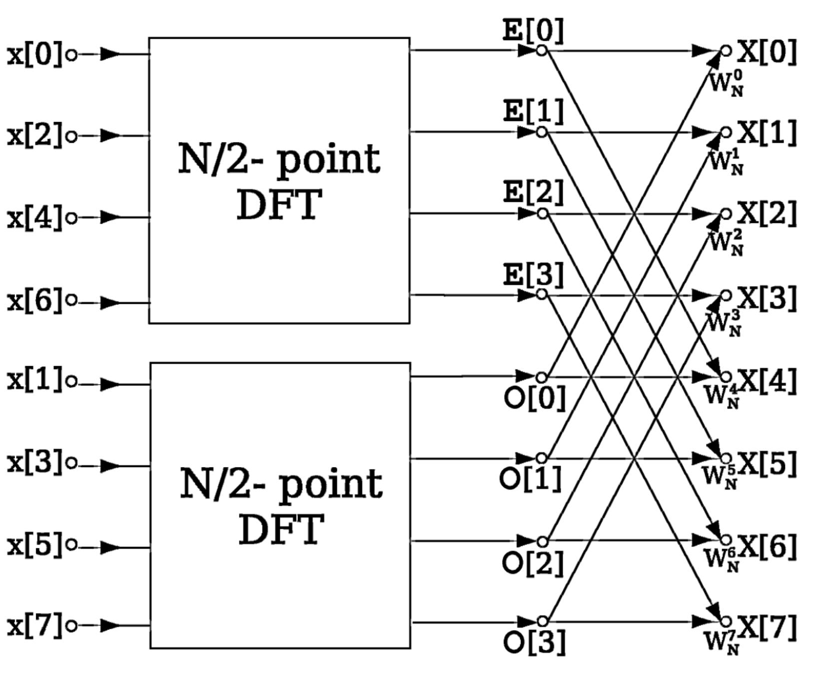
$$O_k = \sum_{p=0}^{N/2-1} y_{2p+1} \exp\left(-i\frac{2p\pi k}{N/2}\right) \quad \text{the odd indices, and}$$

$$\omega = \mathrm{e}^{-i2\pi/N} \quad \text{and} \quad \omega^k = \mathrm{e}^{-i2\pi k/N} \quad \text{the "twiddle factor".}$$

Slide Type Sub-Slide

$$\text{Split} \qquad c_k = \sum_{n=0}^{N-1} y_n \exp\left(-i\frac{2\pi kn}{N}\right) = E_k + \omega^k O_k,$$

- E_k and O_k represent DFTs over points sampled twice as far apart as the original interval.
- # of operations for each E_k and $D_k: \approx (N/2)^2$.
- If we stopped here: # of operations would be $2 \times (N/2)^2 + 2 \approx N^2/2 + 2$ (bisection + twiddle factor; OK, twiddle factor is a bit more, not enough to matter): **a lot less operations for large N!**
- keep going: E_k and O_k can be bisected (split into two) themselves.
- How many times can we do this until each E_k, O_k has one term?
  - $N = 8 = 2^3$: we can do it $3 = \log_2(8)$ times.
  - $N = 16 = 2^4$: we can do it $4 = \log_2(16)$ times.
  - ...
  - $N = 2^M$: we can do it $M = \log_2(N)$ times.



Slide Type

General formulas

Slide Type Sub-Slide

The textbook provides the derivation, but they're hard to follow without using actual numbers.

- j-th set of samples at the m^{th} stage:
$$E_k^{(m,j)} = \sum_{p=0}^{N/2^m-1} y_{2^m p+j} \exp\left(-i\frac{2\pi kp}{N/2^m}\right), \quad j \in \{0\dots 2^m-1\}$$
Note: all E_k and O_k of previous slides are now some $E_k^{(m,j)}$.
- 2^m = # of DFTs at each level (indexed by j),
- $N/2^m$ = # of samples per intermediate DFT (indexed by k),
- Recursively, working from $M = \log_2 N$:
  - First step: $E_k^{(M,j)} = y_j$ (no k dependence), **ops:** N
  - Next steps: $E_k^{(m,j)} = E_k^{(m+1,j)} + \omega^{2^m k} E_k^{(m+1,j+2^m)}$, **ops:** $N/2^m \times 2^m = N$
  - Last step: $E_k^{(0,0)} = c_k$, the desired DFT coefficients. **ops:** $N \times 1 = N$

Slide Type Sub-Slide

- We end up with N terms in each of the $\log_2(N)$ bisections, so the number of operations is $N\log_2(N)$.
- Huge speed increase for large N
- For $N = 10^6$, old DFT algorithm is $O(N^2) = 10^{12}$ ops, but FFT is $O(N\log_2(N)) \sim 2\times 10^7$ ops.
- Opens door to a wide range of calculations.
- Also more precise: less ops = less accumulation of machine precision errors.
- Note that the same reasoning applies to the inverse FT: the algorithm is called the inverse FFT (iFFT).

Slide Type

Implementation Notes

Slide Type Fragment

- The general formulas are useful if you want to code the FFT yourself: see textbook Exercise 7.7), and script `fft_ts.py` (derived from `dft_ts.py`)
- But it's better to use packages. There are good tricks for saving memory that are implemented in packages like `numpy.fft`: https://numpy.org/doc/stable/reference/routines.fft.html

Slide Type

2D DFTs

Slide Type Slide

Suppose we have a $M \times N$ sample grid, with values y_{mn}. To perform 2D DFT:

Slide Type Fragment

- Fourier transform the M rows:

$$c'_{m\ell} = \sum_{n=0}^{N-1} y_{mn} \exp\left(-i\frac{2\pi \ell n}{N}\right)$$

Slide Type Fragment

- Fourier transform the N columns:

$$c_{k\ell} = \sum_{m=0}^{M-1} c'_{m\ell} \exp\left(-i\frac{2\pi km}{M}\right) = \sum_{k=0}^{M-1}\sum_{n=0}^{N-1} y_{mn} \exp\left[-i2\pi\left(\frac{km}{M} + \frac{\ell n}{N}\right)\right].$$

Slide Type

Inverse 2D DFT

Slide Type Sub-Slide

Inverse 2D DFT:

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

Slide Type

Crucial points to remember

Slide Type Slide

Discrete Fourier Transforms

- Compute the integrals in the formulas for the Fourier coefficients with the trapezoidal rule
- Periodicity of the signal makes the trapezoidal rule even easier
- Trapezoidal rule: not the best integral, but to compute the inverse DFT yields **exactly** the original values!

Slide Type Sub-Slide

Fast Fourier Transforms

- End result is **exactly** the end result of DFT
- Much faster simply thanks to a clever rearrangement of order of operations: "divide-and-conquer".
- Made possible by symmetries in roots of unity $\exp(2i\pi n/N)$
- Bisections and multiplications: $O(N \log_2 N)$ ops.
- Much faster than $O(N^2)$ for DFT.