MPS224 Scientific Computing

Dr. Sam Dolan

School of Mathematical and Physical Sciences,
University of Sheffield

Spring 2026

9. Signal processing and Fourier transforms

G18 Hicks Building
s.dolan@sheffield.ac.uk

Today's lecture

- The **Discrete Fourier Transform** (DFT)
- **Motivations:**
  - looking for hidden signals in experimental data
  - finding the frequencies of a damped driven oscillator
- Definition of DFT
- The inverse DFT
- Filtering

Integral transforms

- An **integral transform** maps an equation from its original domain into another domain where it might be manipulated and solved much more easily than in the original domain.
- An integral transform $\widetilde{f}$ of a function $f(t)$ takes the form

$$\widetilde{f}(\omega) = \int_a^b f(t)\, K(\omega, t)\, dt$$

  where $K(\omega, t)$ is the **kernel**.
- The most well-known integral transform is the **Fourier transform**:

$$\widetilde{f}(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt$$

- It maps from the **time domain** (t) to the **frequency domain** (ω).
- For data sets, the integral is replaced by a sum to get a **discrete transform**.

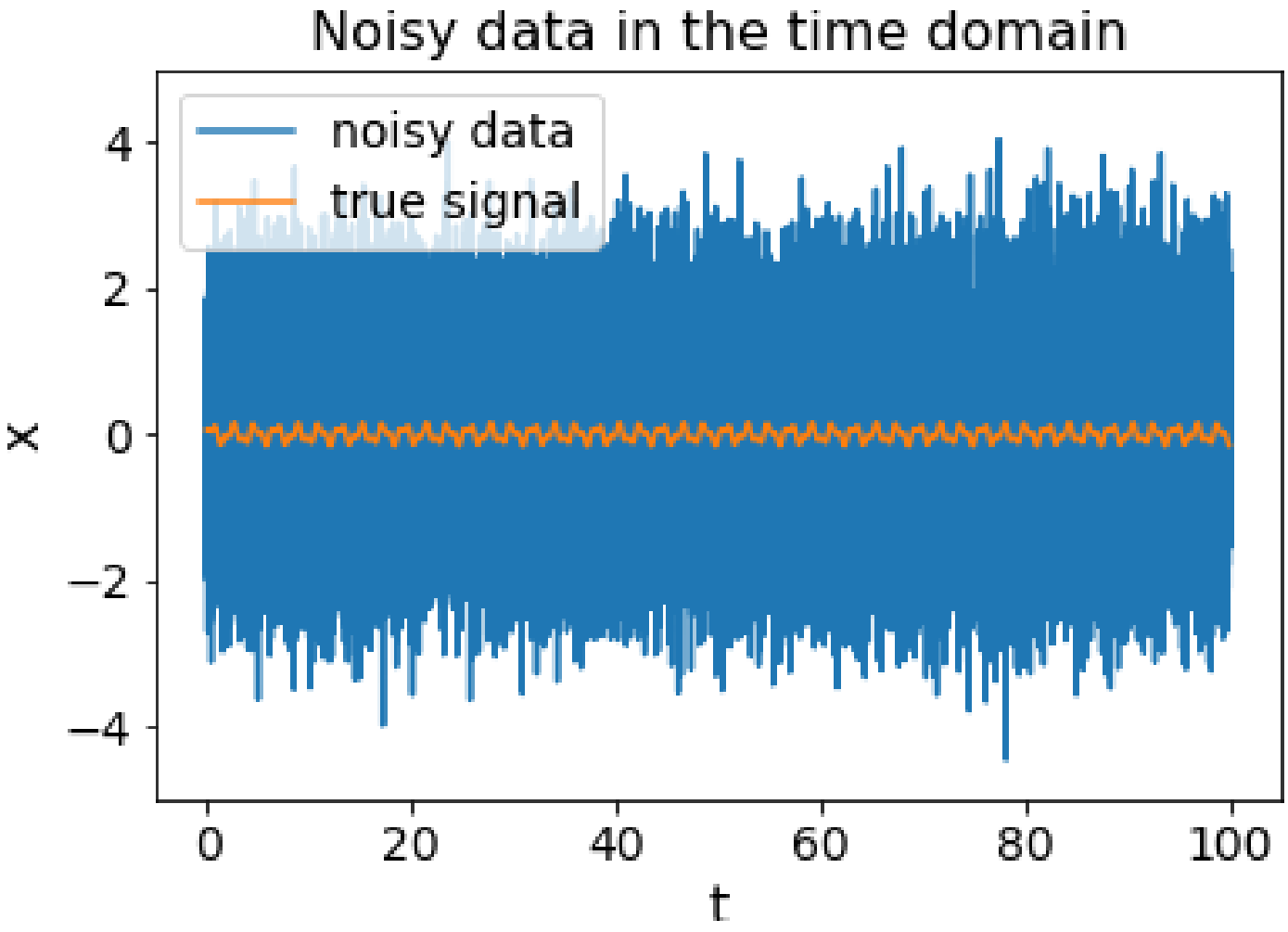
Example (1)

- Imagine a scientific instrument (e.g. a seismometer or radio telescope) measuring noisy data with a hidden signal.
- For example,

$$y(t) = 0.1\sin(3t) + 0.06\cos(7t) + n(t)$$

- Here blue = signal and red = noise.
- Suppose $n(t)$ is noise drawn from a Gaussian distribution with standard deviation $\sigma = 1$.

Example (1): Noisy data with hidden signal

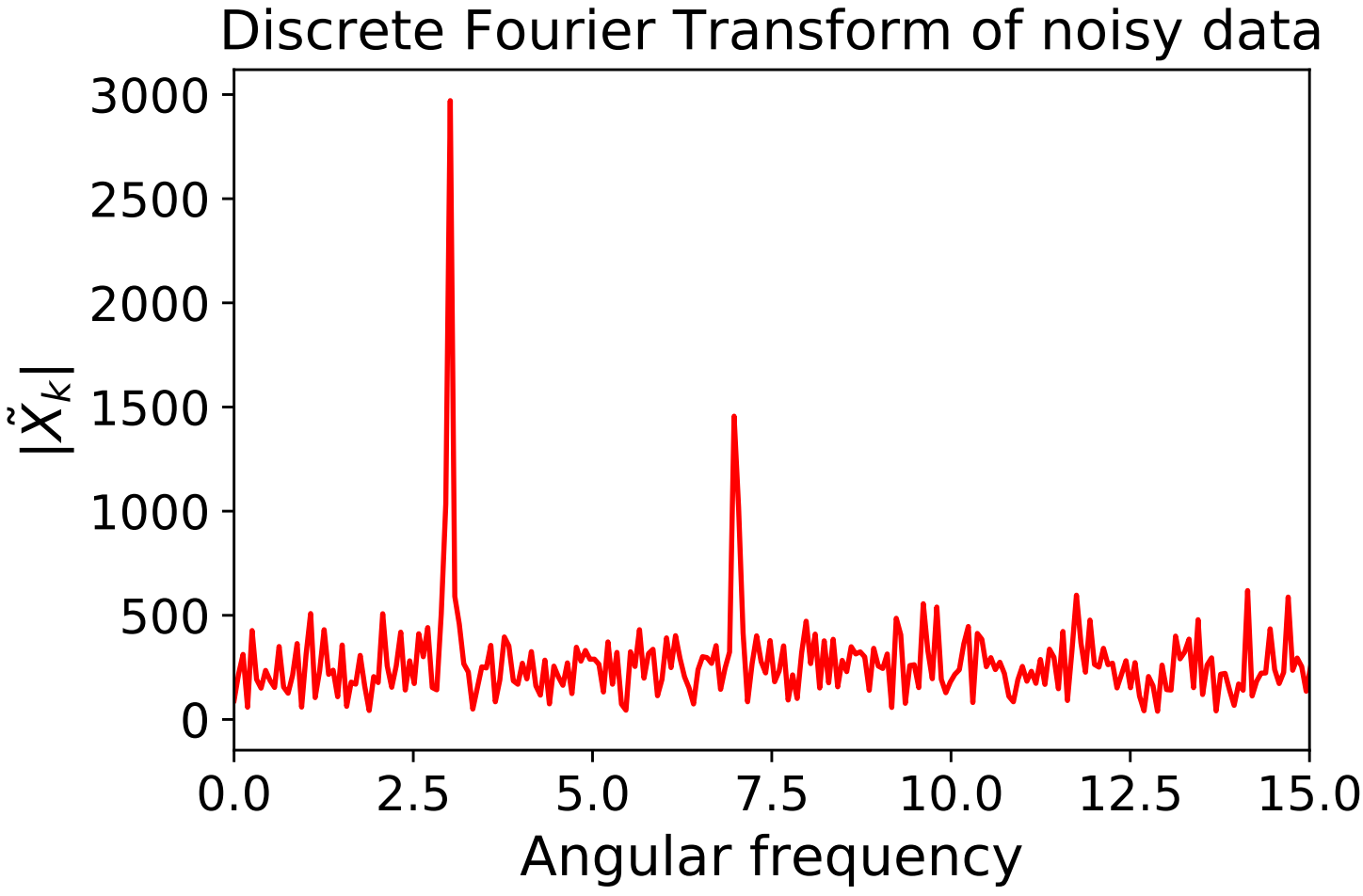


(a) time domain: nothing visible

Example (1): Noisy data with hidden signal

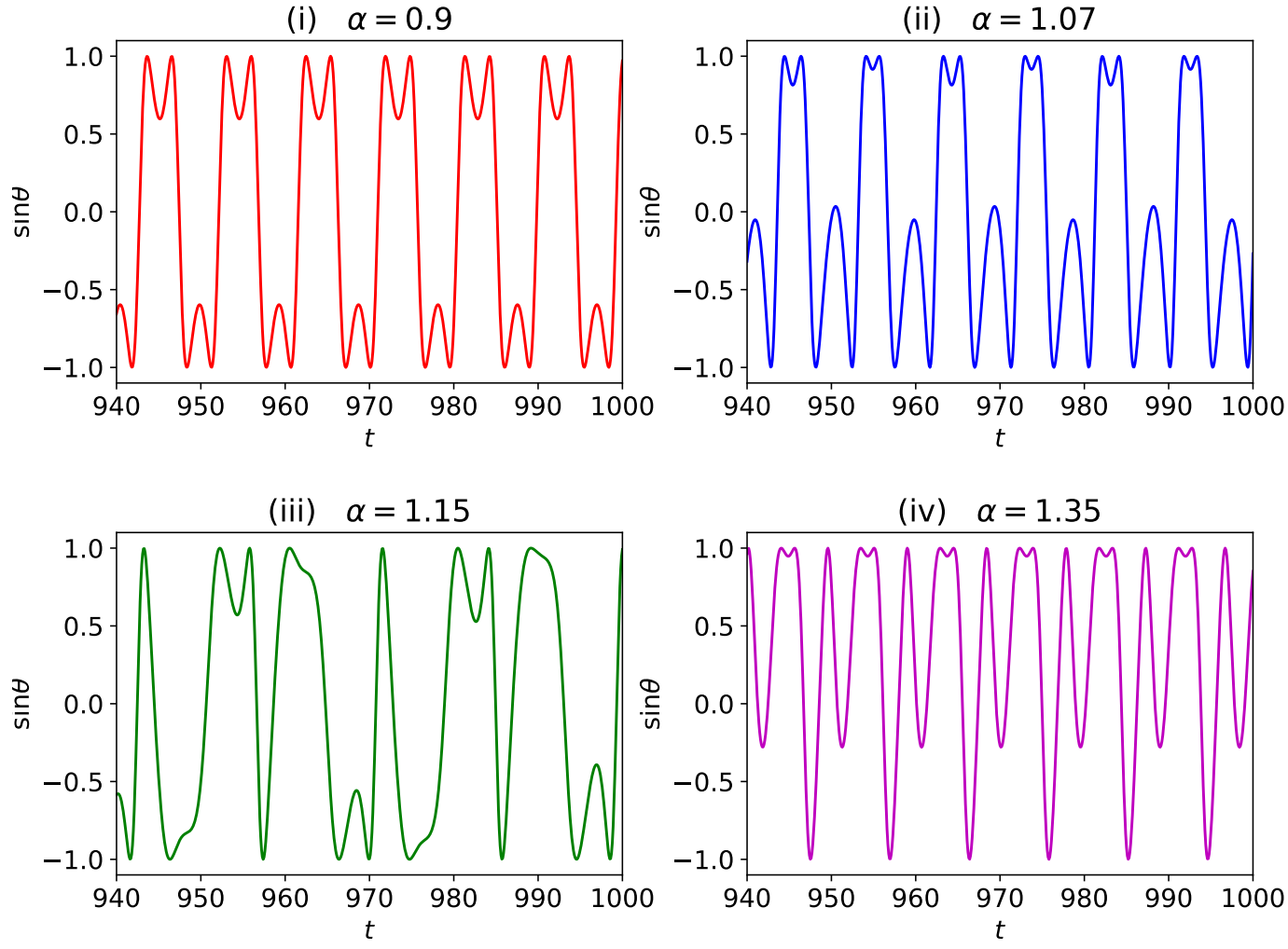
. . . now take the Discrete Fourier Transform . . .

Example (1): Noisy data with hidden signal



(b) frequency domain: two ‘spikes’.

Example (2): A damped driven oscillator

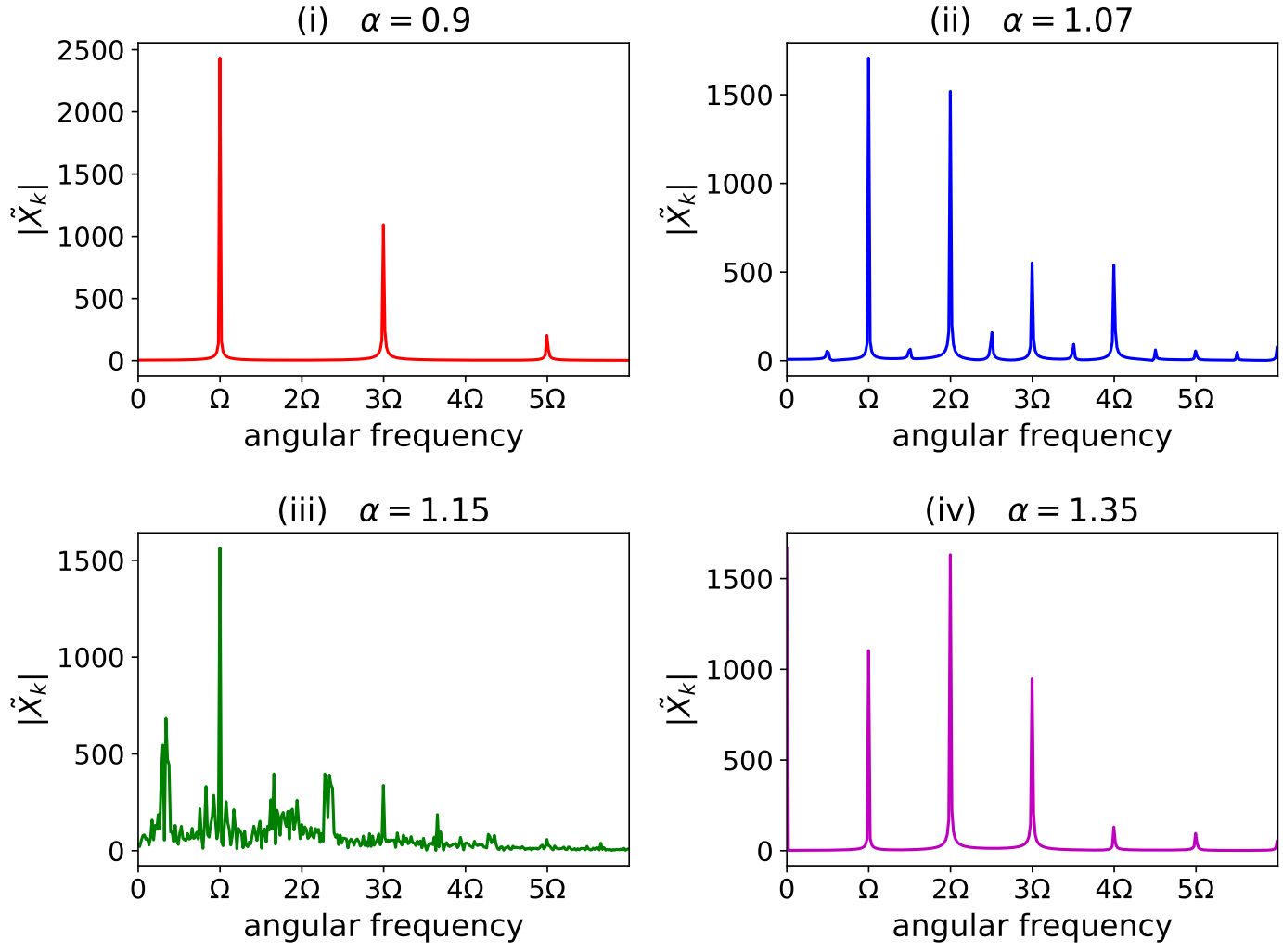


- **Q.** Is the response periodic?
  What is the fundamental period?

Example (2): A damped driven oscillator

. . . now take the Discrete Fourier Transform . . .

Example (2): A damped driven oscillator



- The DFT reveals the **harmonics** present in the signal.

The Discrete Fourier Transform (DFT)

Definition:

- Suppose we have a data set made up of a sequence of complex numbers: $x_0, x_1, \ldots x_{n-1}$
- The discrete Fourier transform (DFT) of x_j is the sequence of **complex numbers** $\widetilde{X}_k$ defined by

$$\widetilde{X}_k \equiv \sum_{j=0}^{n-1} x_j \exp\left(-i\,2\pi jk/n\right), \qquad j,k \in \mathbb{Z},$$

  where i is the unit imaginary.
- The sequence $\widetilde{X}_k$ is ***n*-periodic** (as $e^{2i\pi} = 1$):

$$\widetilde{X}_{k\pm n} = \widetilde{X}_k$$

- $\widetilde{X}_k$ are called the **Fourier coefficients** of x_j.

The Inverse Transform

Inverse Discrete Fourier Transform

- The inverse DFT **goes the other way**:

$$x_j = \frac{1}{n} \sum_{k=0}^{n-1} \widetilde{X}_k \exp\left(+i\, 2\pi jk/n\right)$$

- Compare with

$$\widetilde{X}_k \equiv \sum_{j=0}^{n-1} x_j \exp\left(-i\, 2\pi jk/n\right)$$

- For the inverse transformation, use `numpy.fft.ifft()`.

Proof of inverse formula:

- Substitute definition of $\widetilde{X}_k$ into inverse formula:

$$
\begin{aligned}
& \frac{1}{n}\sum_{k=0}^{n-1}\left\{\widetilde{X}_k\right\}\exp\left(+i\,2\pi jk/n\right) \\
= \quad & \frac{1}{n}\sum_{k=0}^{n-1}\left\{\sum_{j'=0}^{n-1}x_{j'}\exp\left(-i\,2\pi j'k/n\right)\right\}\exp\left(+i\,2\pi jk/n\right) \\
= \quad & \sum_{j'=0}^{n-1}x_{j'}\times\frac{1}{n}\sum_{k=0}^{n-1}\exp\left(+i\,2\pi(j-j')k/n\right) \\
= \quad & \sum_{j'=0}^{n-1}x_{j'}\delta_{jj'} \\
= \quad & x_j \qquad \text{as required.}
\end{aligned}
$$

- N.B. $\delta_{jj'}$ is 1 if $j'=j$, and 0 otherwise.

Implementation in Python

- Python has several modules for DFTs, including `numpy.fft` and `scipy.fftpack` and `scipy.signal`.
- FFT = **Fast** Fourier Transform (efficient algorithm).
- Here are some useful functions:
  - `numpy.fft.fft()` : Compute the one-dimensional discrete Fourier Transform.
  - `numpy.fft.ifft()` : Compute the one-dimensional **inverse** discrete Fourier Transform.
  - `numpy.fft.rfft()`: Computes the FFT of an array of **real** values.
  - `scipy.signal.periodogram()` : Estimate power spectral density using a periodogram. (The PSD is the square magnitude of the Fourier coefficients).

Interpreting the DFT: time series

- Suppose $\{x_j\}$ is a data set of length n, sampled at regular time intervals Δt:

$$t_j \equiv j\Delta t.$$

- We can think of the DFT as a map from the '**time domain**' to the '**frequency domain**':

$$(t_j, x_j) \longrightarrow (\omega_k, \widetilde{X}_k)$$

where

$$\omega_k = k\Delta\omega, \qquad \Delta\omega = \frac{2\pi}{T}, \qquad T \equiv n\Delta t$$

such that $2\pi jk/n = \omega_k t_j$.

- The frequency spacing $\Delta\omega$ is inversely proportional to the **total duration** $T = n\Delta t$.
- The maximum frequency ($\omega_{\text{max}} = 2\pi/\Delta t$) is inversely proportional to the sampling interval Δt.

Interpreting the DFT

Suppose x_j are **real** measurements at times $t_j = j\Delta t$.

- x_j are real $\Leftrightarrow$ $\widetilde{X}_k$ are **Hermitian**: $\widetilde{X}_k^* = \widetilde{X}_{-k}$
- $\Rightarrow$ $\widetilde{X}_0$ is **real**: $\widetilde{X}_0 = \sum x_j = n\bar{x}$
- Let $\widetilde{X}_k = a_k - ib_k$. Then one can show that

$$x_j = \frac{1}{2}a_0 + \frac{2}{n}\sum_{k=1}^{n-1}\left\{a_k \cos\left(\omega_k t_j\right) + b_k \sin\left(\omega_k t_j\right)\right\}$$

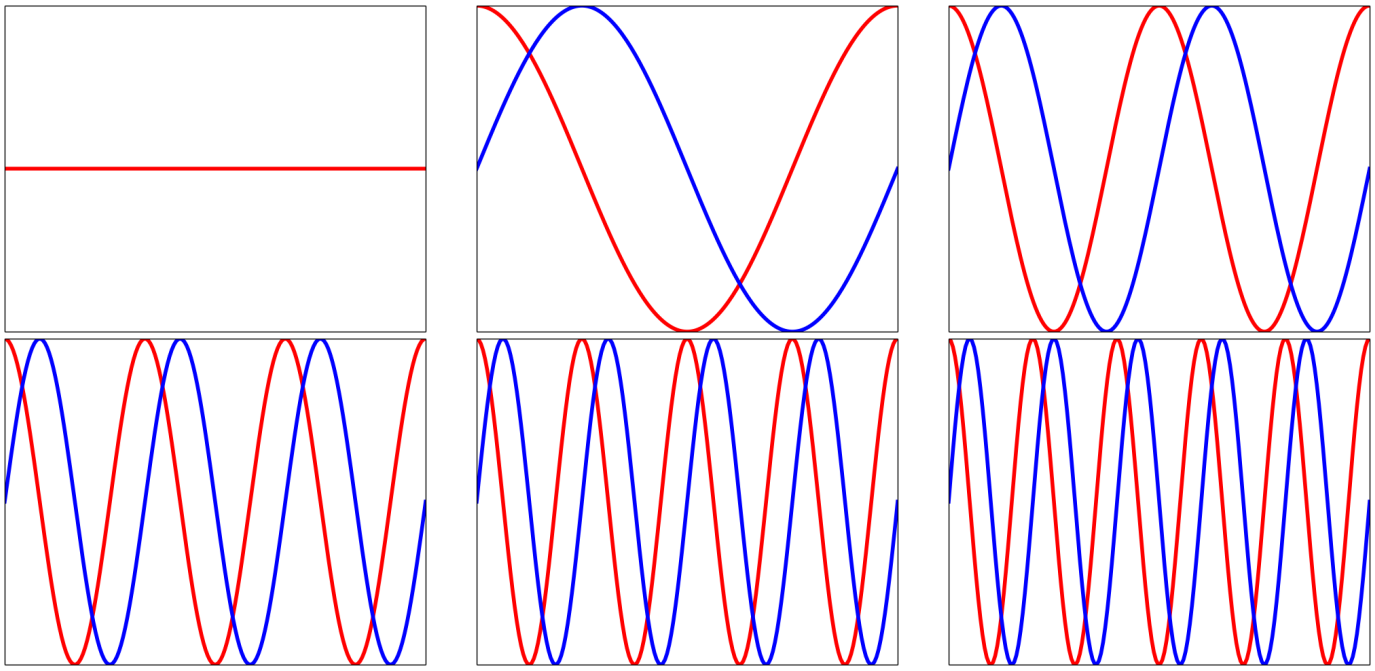
where a_k and b_k are real coefficients

$$a_k = \mathrm{Re}\sum_{j=0}^{n-1} x_j e^{-i\,2\pi jk/n} = \sum_{j=0}^{n-1} x_j \cos\left(\omega_k t_j\right)$$

$$b_k = -\mathrm{Im}\sum_{j=0}^{n-1} x_j e^{-i\,2\pi jk/n} = \sum_{j=0}^{n-1} x_j \sin\left(\omega_k t_j\right)$$

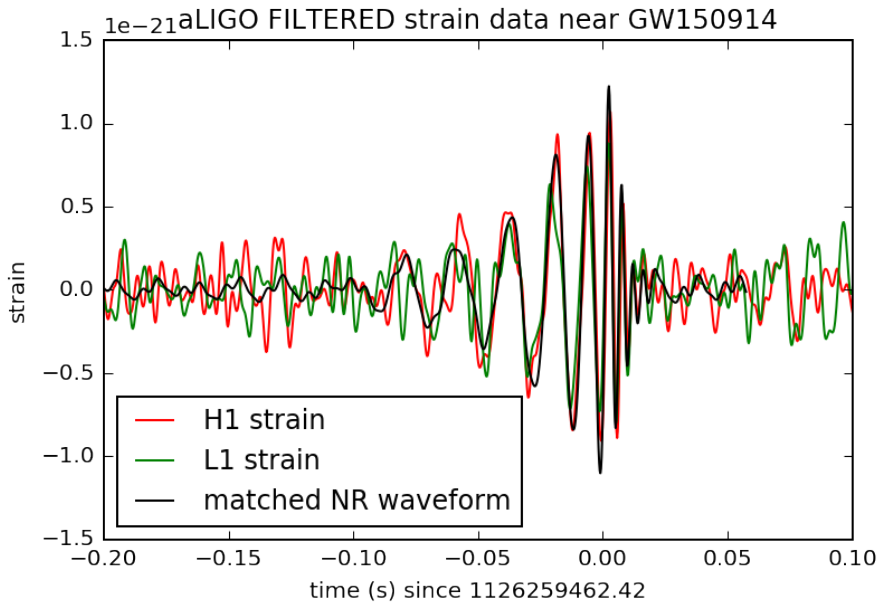
Interpreting the DFT

- x_j is made up of a sum of harmonics:
- a_k and b_k are these amplitudes of the harmonics



Filtering

- Many signal processing algorithms work as follows:
  - Apply a DFT
  - Apply a filter in the frequency domain (e.g. remove high or low frequencies)
  - Apply an inverse DFT.
- **Example:** For a tutorial in signal processing for **gravitational-wave detectors**, see `https://losc.ligo.org/s/events/GW150914/GW150914_tutorial.ipynb`



Filtering

- Suppose I wanted to use a **low-pass filter** to remove the high frequencies from a data set a_j.
- I can define a **filter** in terms of its Fourier components $\widetilde{F}_k$,
- then define a **filtered dataset** b_j via

$$b_j = IDFT(\widetilde{B}_k), \qquad \widetilde{B}_k = \widetilde{A}_k \widetilde{F}_k$$

- A naive low-pass filter would be:

$$\widetilde{F}_k = \Theta(k, k_0) \equiv \begin{cases} 0 & |k| > k_0 \\ 1 & \text{otherwise} \end{cases}, \quad -\frac{n}{2} \leq k \leq \frac{n}{2}.$$

- But this is not suitable. **Why not**? First we need to understand the **convolution theorem**.

Convolution

Definition:

- The convolution of two n-periodic sequences a_j and b_j is defined by

$$(a \otimes b)_j = \sum_{j'=0}^{n-1} a_{j'} b_{j-j'}$$

Notes:

- Convolution is like **smearing** one sequence with another.

The Convolution Theorem

- Suppose a_j and b_j are sequences with Fourier coefficients $\widetilde{A}_k$ and $\widetilde{B}_k$. Now consider the sequence c_j whose Fourier coefficients are $\widetilde{C}_k = \widetilde{A}_k \widetilde{B}_k$. What are c_j?

$$
\begin{aligned}
c_j &= \frac{1}{n} \sum_{k=0}^{n-1} \left(\widetilde{A}_k \widetilde{B}_k \right) \exp\left(i\, 2\pi jk/n \right) \\
&= \frac{1}{n} \sum_k \sum_p \sum_q a_p e^{-i2\pi pk/n} b_q e^{-i2\pi qk/n} e^{i2\pi jk/n} \\
&= \sum_p \sum_q a_p b_q \times \frac{1}{n} \sum_k \exp\left(i \frac{2\pi k}{n} (j - p - q) \right) \\
&= \sum_p a_p b_{j-p} = (a \otimes b)_j
\end{aligned}
$$

- That is, c_j is the **convolution** of a_j and b_j.

The Convolution Theorem

- This works both ways:

$$\widetilde{C}_k = \widetilde{A}_k \widetilde{B}_k \qquad \Leftrightarrow \qquad c_j = (a \otimes b)_j$$

and

$$\widetilde{C}_k = \left(\widetilde{A} \otimes \widetilde{B}\right)_k \qquad \Leftrightarrow \qquad c_j = a_j b_j.$$

Practical application

- Naive low-pass filter:

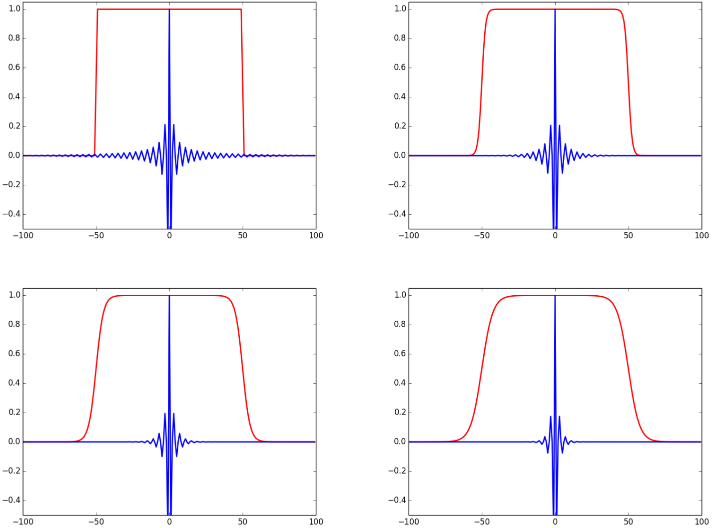
$$\widetilde{F}_k = \Theta(k, k_0) \equiv \begin{cases} 0 & |k| > k_0 \\ 1 & \text{otherwise} \end{cases} \qquad \text{for} \quad -\frac{n}{2} \leq k \leq \frac{n}{2}.$$

- Defining $c = \pi j/n$,

$$\begin{aligned} f_j &= \frac{1}{n} \sum_{k=-n/2}^{n/2-1} \Theta(k, k_0) e^{2ick} \\ &= \frac{1}{n} \sum_{k=-k_0}^{k_0} \left(e^{2ic}\right)^k \\ &= \frac{1}{n} \frac{e^{ic(2k_0+1)} - e^{-ic(2k_0+1)}}{e^{ic} - e^{-ic}} \\ &= \frac{1}{n} \frac{\sin((2k_0+1)\pi j/n)}{\sin(\pi j/n)} \end{aligned}$$

- This is **wide** and oscillatory. Not suitable.

Filters and their Inverse DFTs



Red = filter in frequency domain, Blue = time domain profile.

Filtering

- **Conclusion**: Use a smooth filter in the frequency domain, rather than a sharp cut-off, to avoid introducing oscillations called **ringing artifacts** associated with the Gibbs' phenomenon.