An Introduction to Signal Processing

The Fourier Domain

Made with GWpy: gwpy.github.io

Dr. Jess McIver

LSSTC DSFP

June 12, 2019

LIGO DCC G1901110

THE UNIVERSITY OF BRITISH COLUMBIA

Outline

Gravitational waves and LIGO data

Introduction to GWpy

Into the frequency domain!

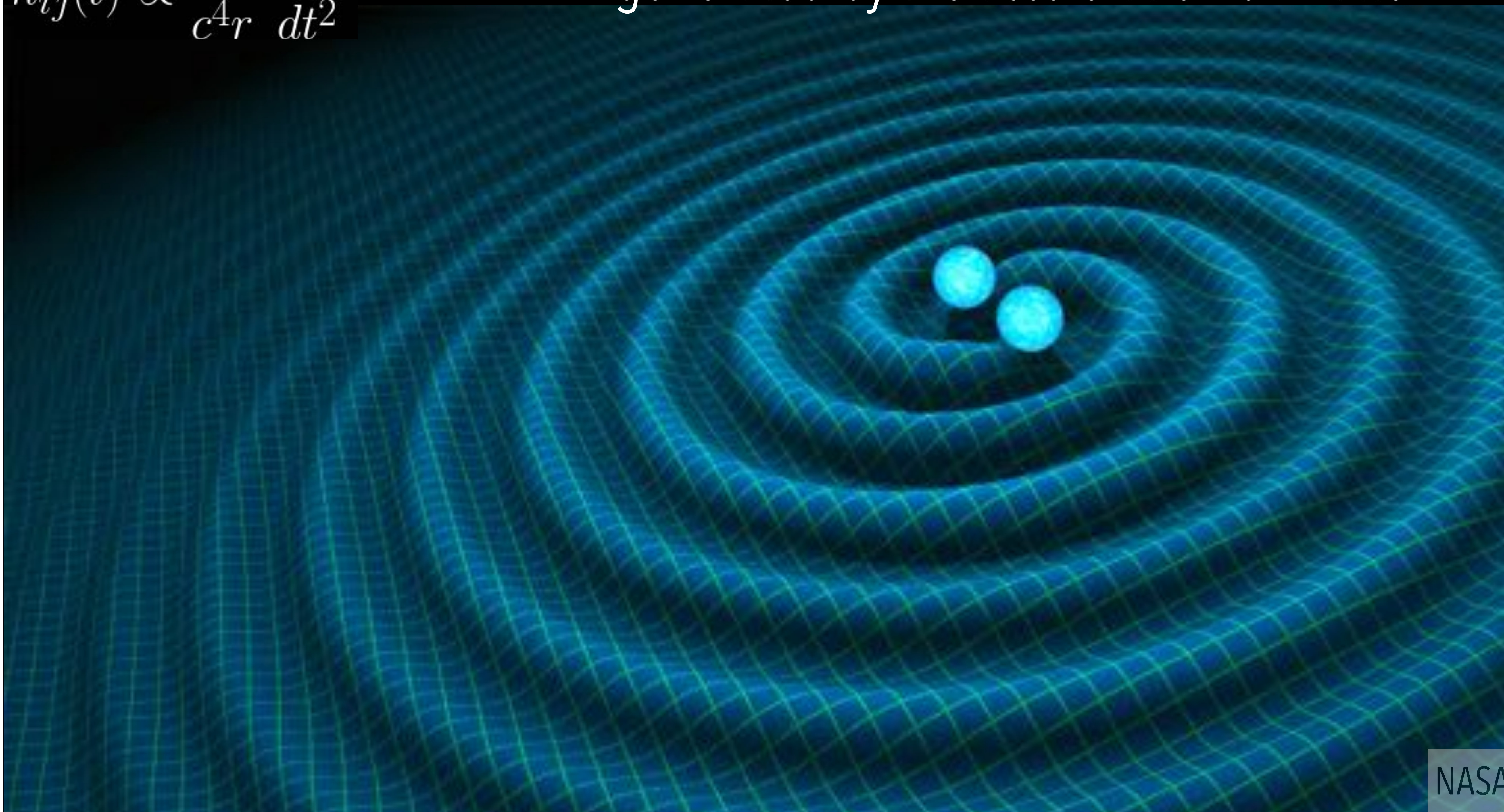
- The Fourier transform
- Spectra of time series data
- Fast Fourier Tranforms (FFTs) and averaging
- Time-frequency representations

Examples

Gravitational waves

Ripples in the fabric of spacetime
generated by the acceleration of matter

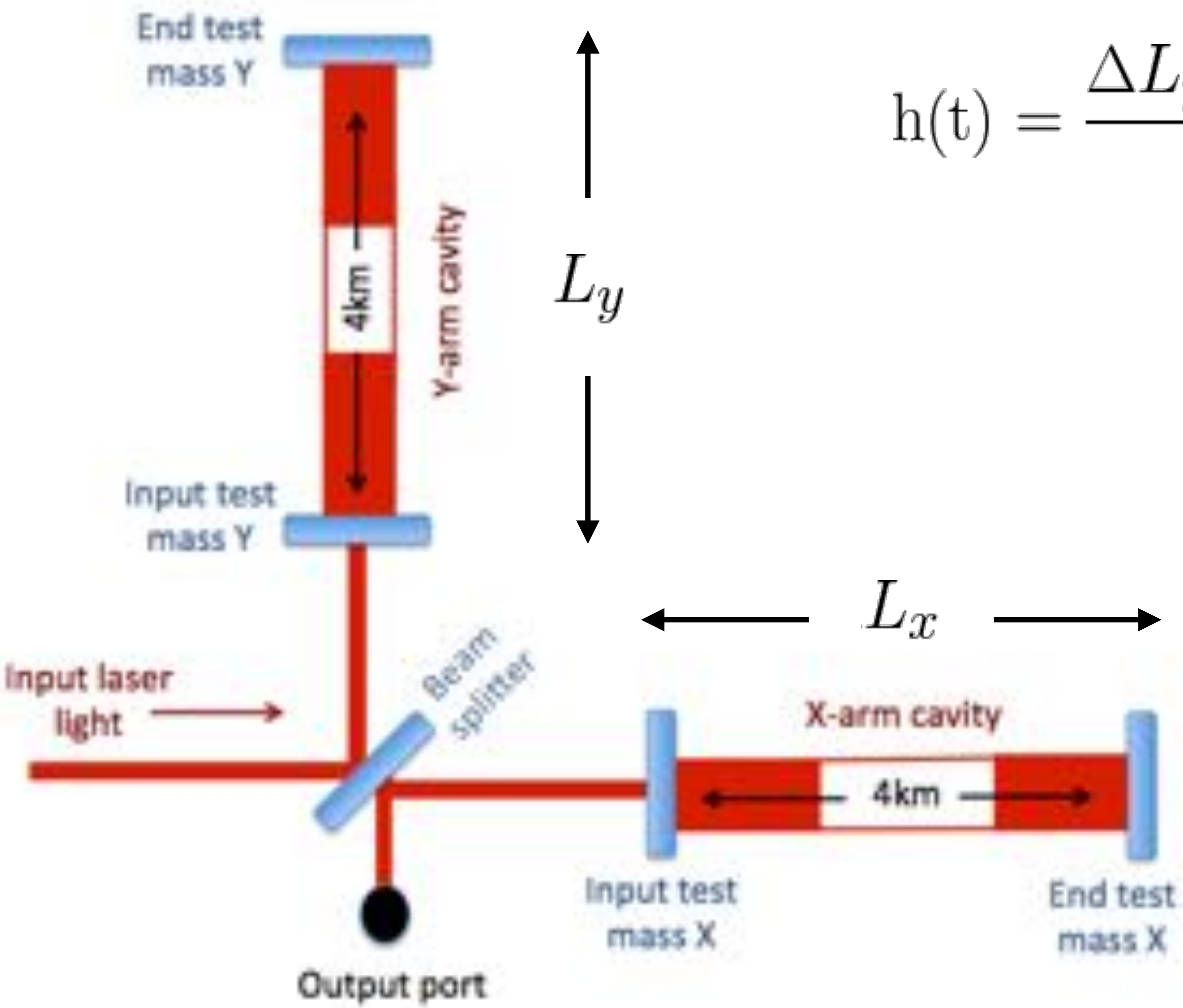
$$h_{ij}(t) \propto \frac{G}{c^4 r}\frac{d^2 I_{ij}}{dt^2}$$



Gravitational wave propagation

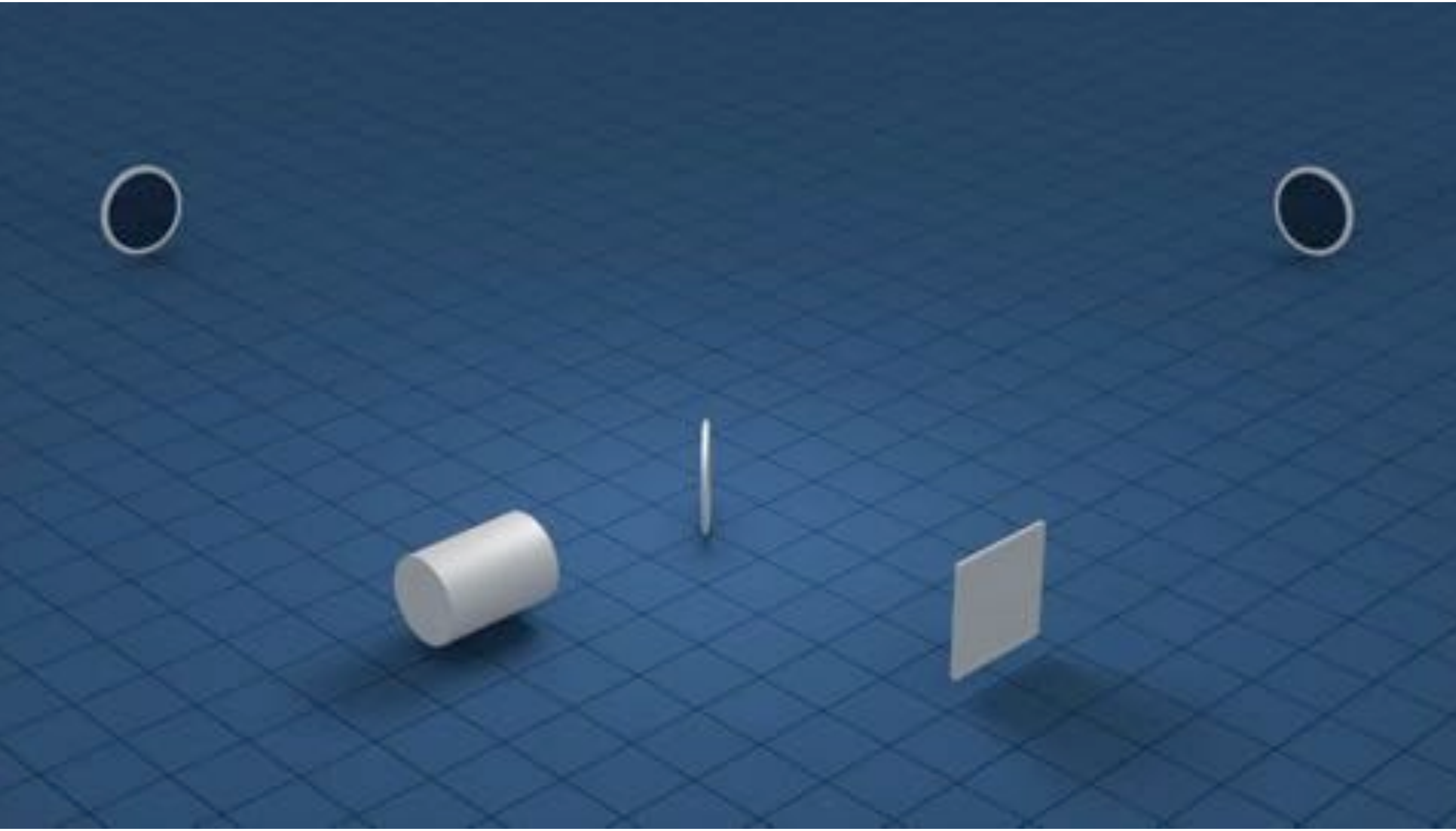
Spacetime strain h(t) measured as $\frac{\Delta L}{L}$

LIGO strain data



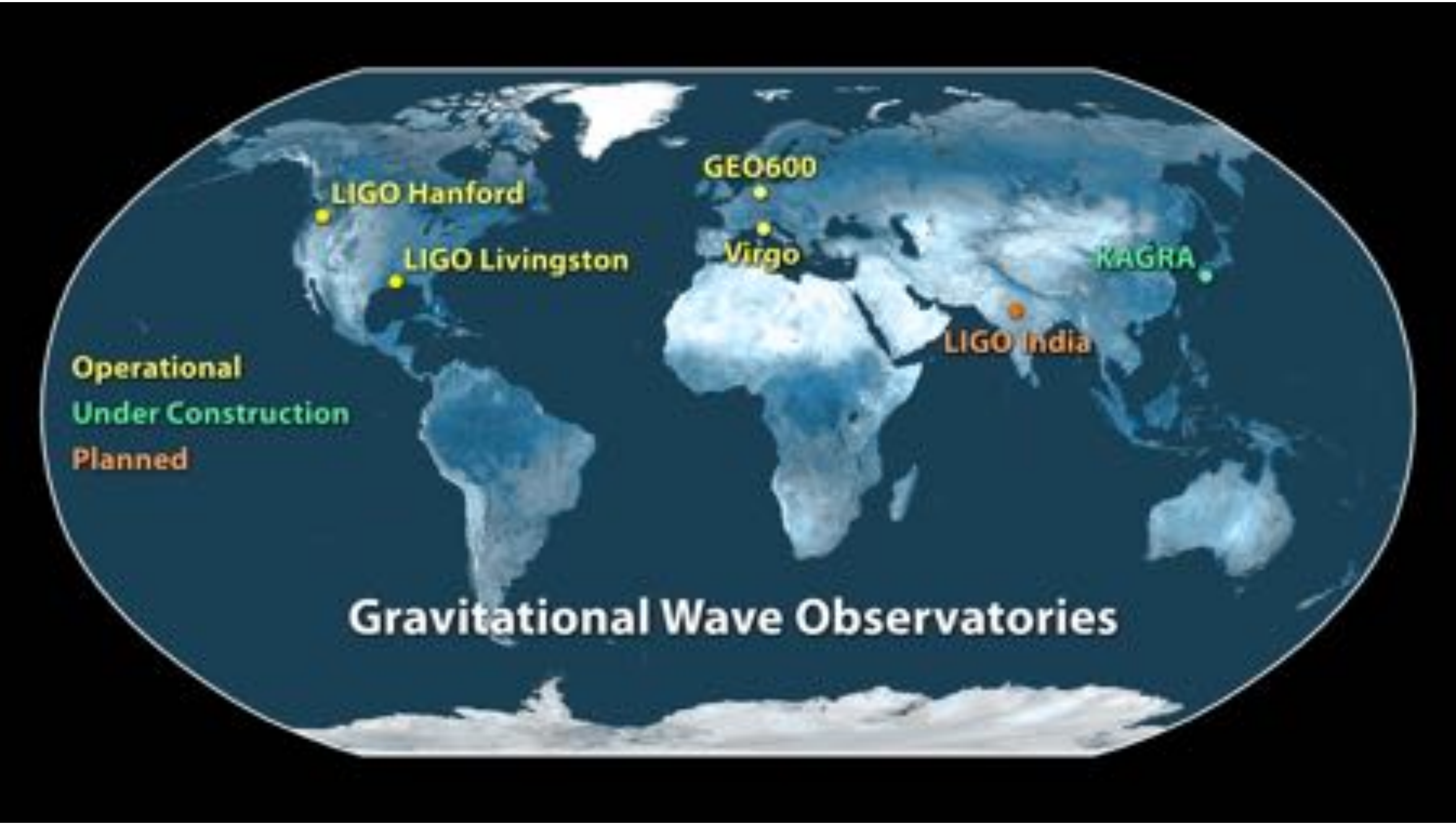
$$\mathrm{h(t)} = \frac{\Delta L_y - \Delta L_x}{L}$$

LIGO strain data



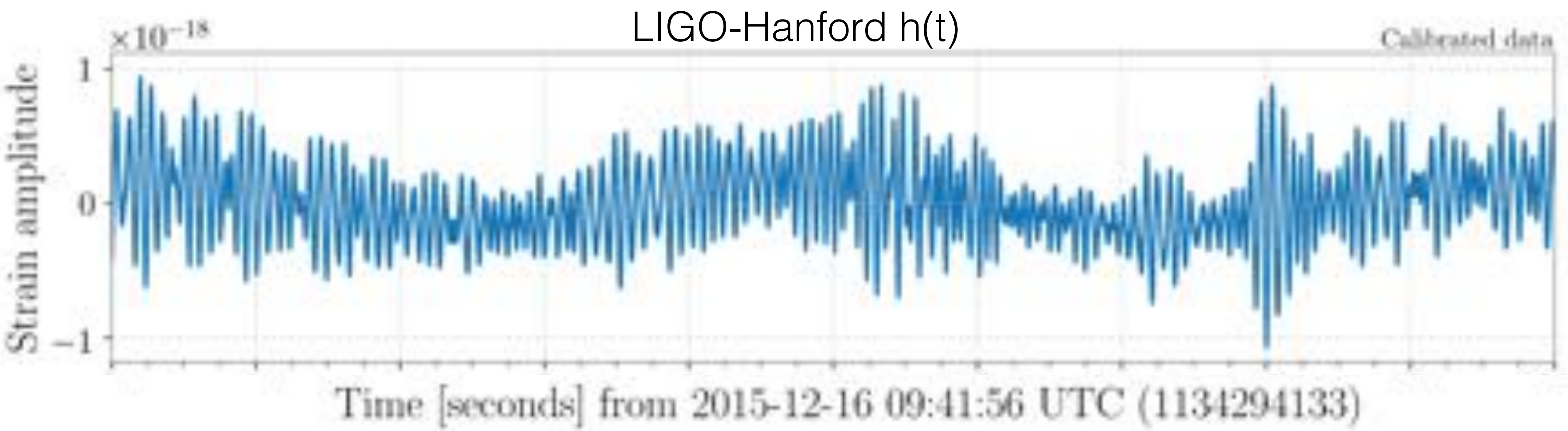
LIGO/Caltech

The global network of current gen interferometers



LIGO/Caltech

What does LIGO data look like?



Made with GWpy!

What's in a LIGO data file?

meta: Meta-data for the file. This is **basic information** such as the GPS times covered, which instrument, etc.

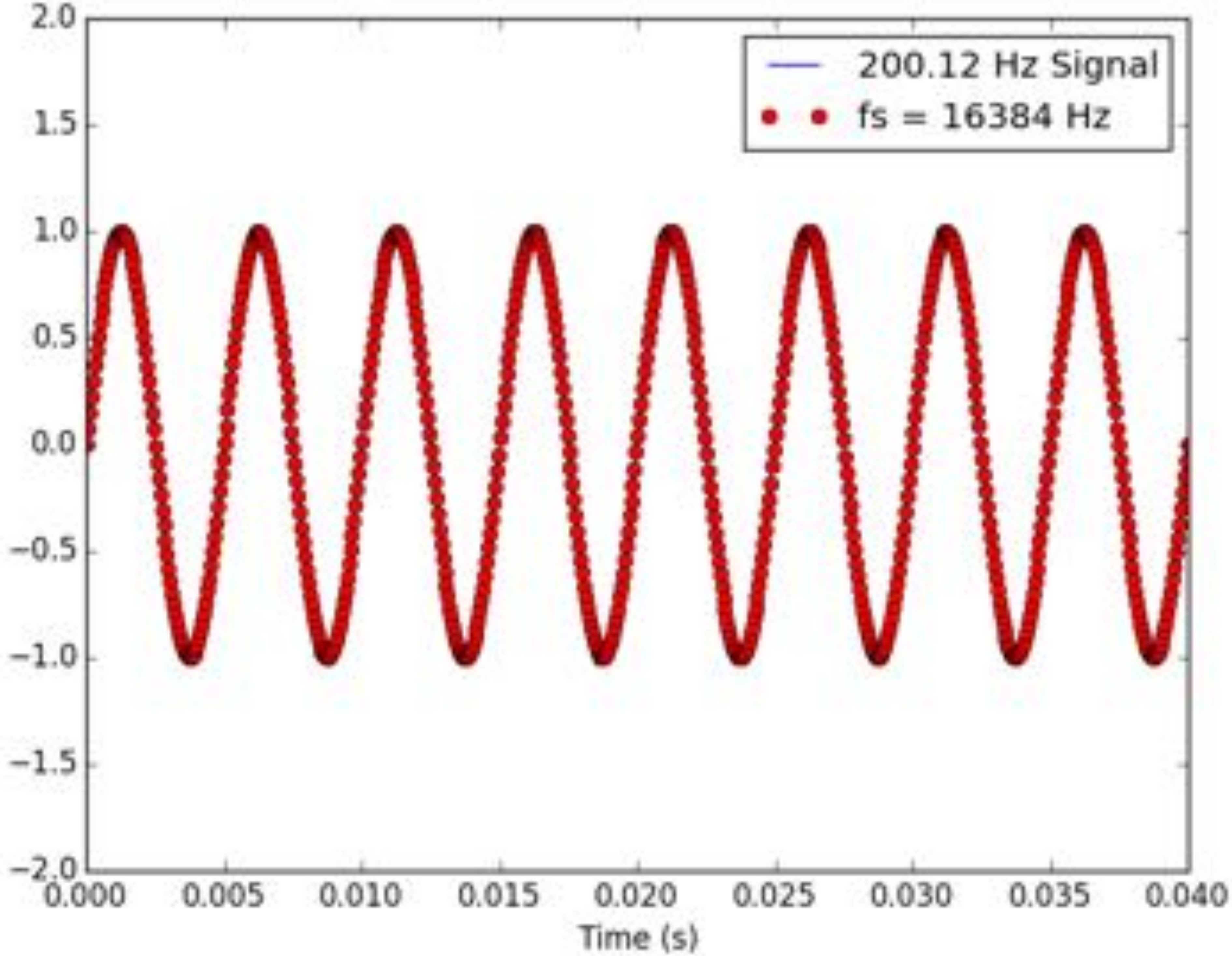
strain: Strain data from the interferometer. This is "the data", the **main measurement of spacetime strain** recorded by the LIGO detectors.

quality: A 1 Hz time series describing the **data quality** for each second of data.

h(t) **sampling rate** for LIGO detectors: 16384 Hz
Open data: 4096 Hz and 16384 Hz

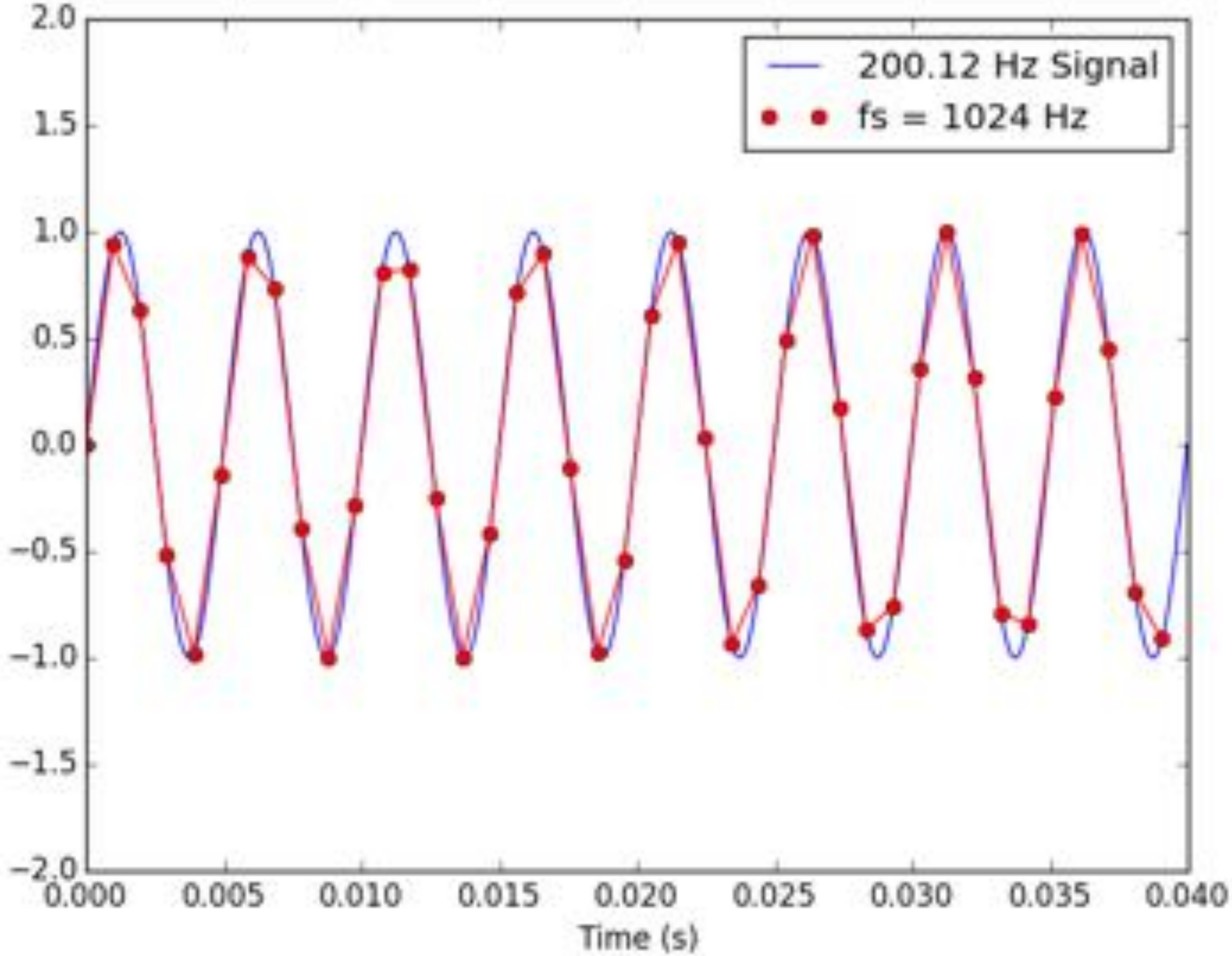
Why do we care about sampling rate, f_S ?

Discrete Time Samples



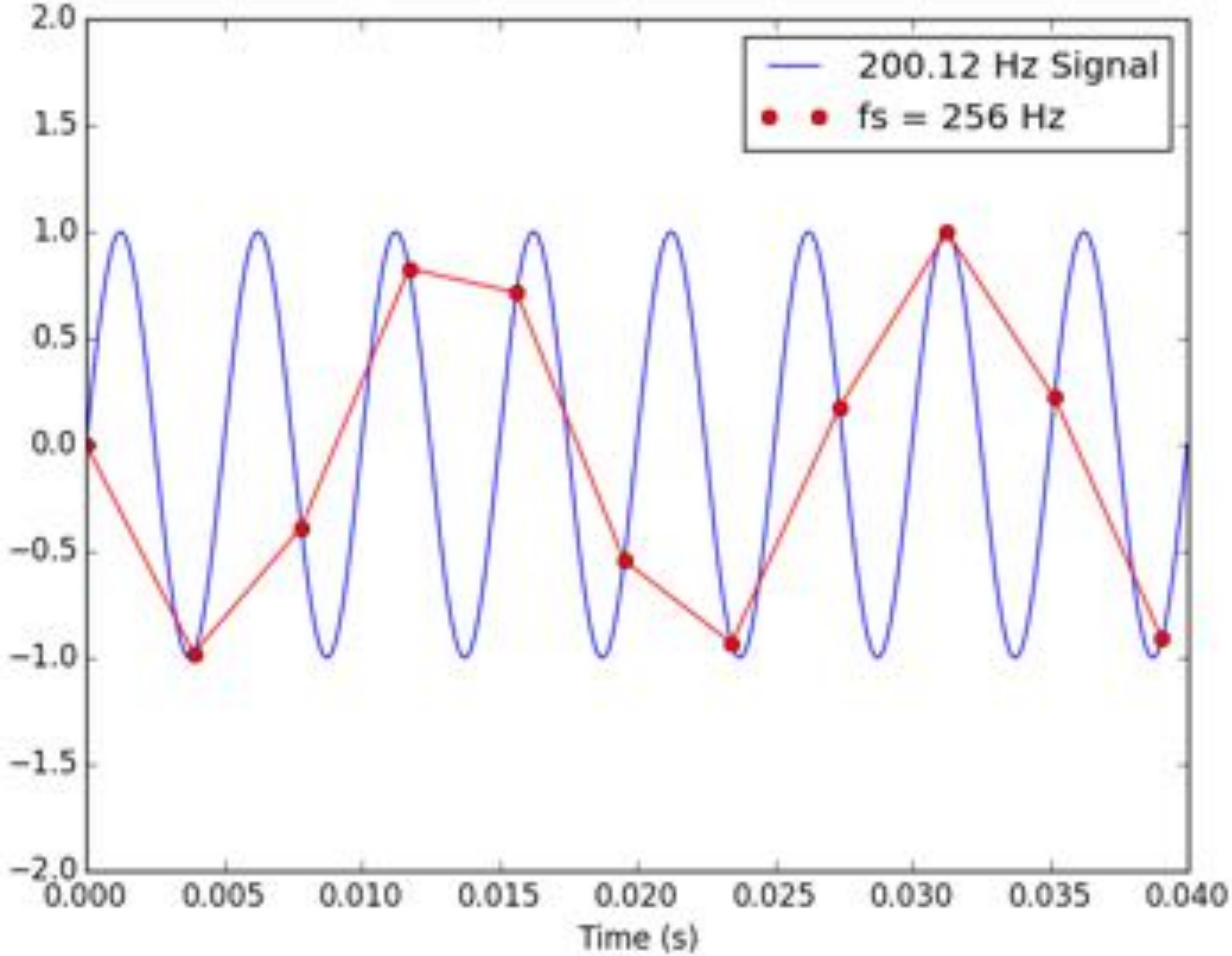
Slide from J. Kanner

Discrete Time Samples



Slide from J. Kanner

Discrete Time Samples



Slide from J. Kanner

Nyquist Frequency

- Nyquist Frequency = $\frac{f_s}{2}$
- Data can only accurately represent frequency content below the *Nyquist frequency*
- Higher frequency signals will be lost or “aliased” to lower frequencies

Introduction to GWpy

A python package for gravitational-wave astrophysics

https://gwpy.github.io

Heavily dependent on numpy, scipy, astropy, matplotlib

Provides intuitive object-orientated methods to access GW detector data, process, and visualize them

Not specific to GW data other than data access routines

Slide from D. Macleod

GWpy Quickstart

Import the class that represents the data you want to study

```
>>> from gwpy.timeseries import TimeSeries
```

Fetch some open data from the OSC

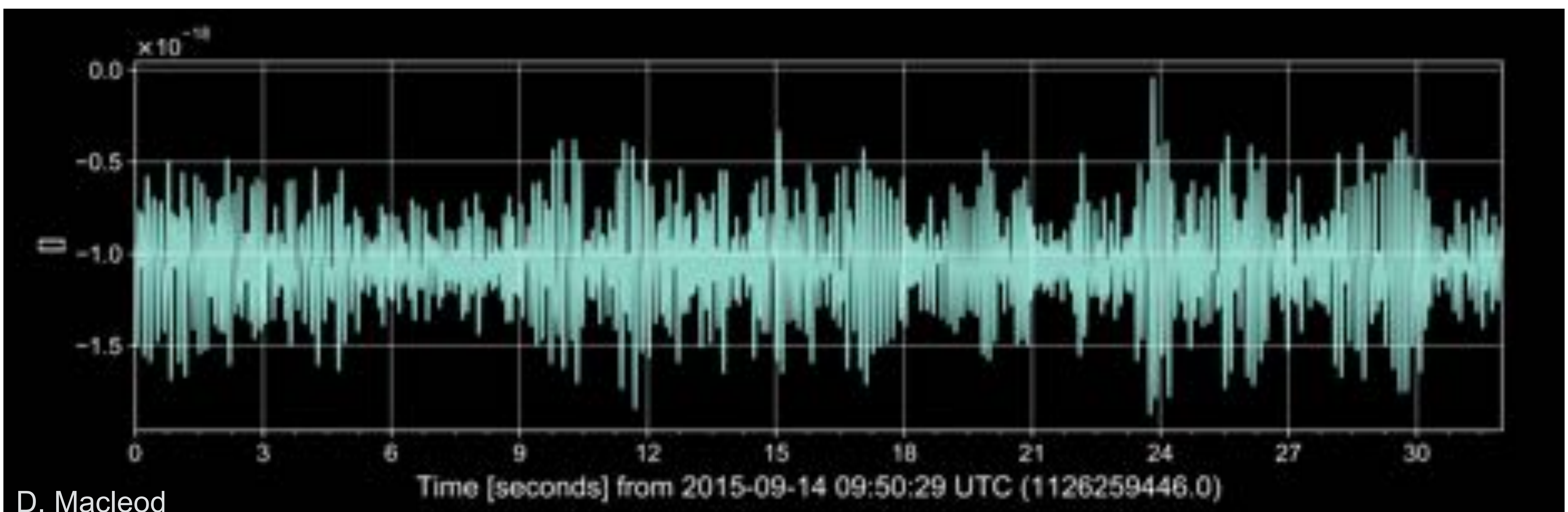
```
>>> data = TimeSeries.fetch_open_data('L1', 'Sep 14 2015 09:50:29', 'Sep 14 2015 09:51:01')
```

Make a plot

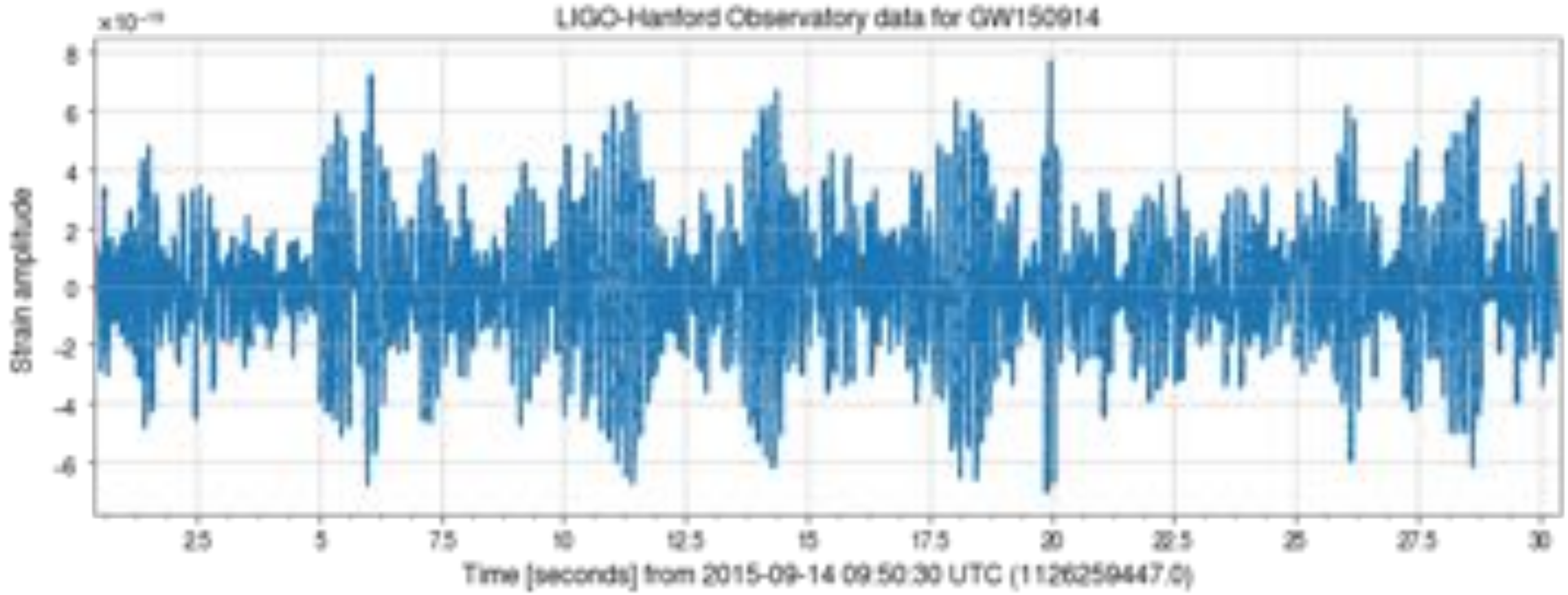
```
>>> plot = data.plot()
```

Display the plot

```
>>> plot.show()
```



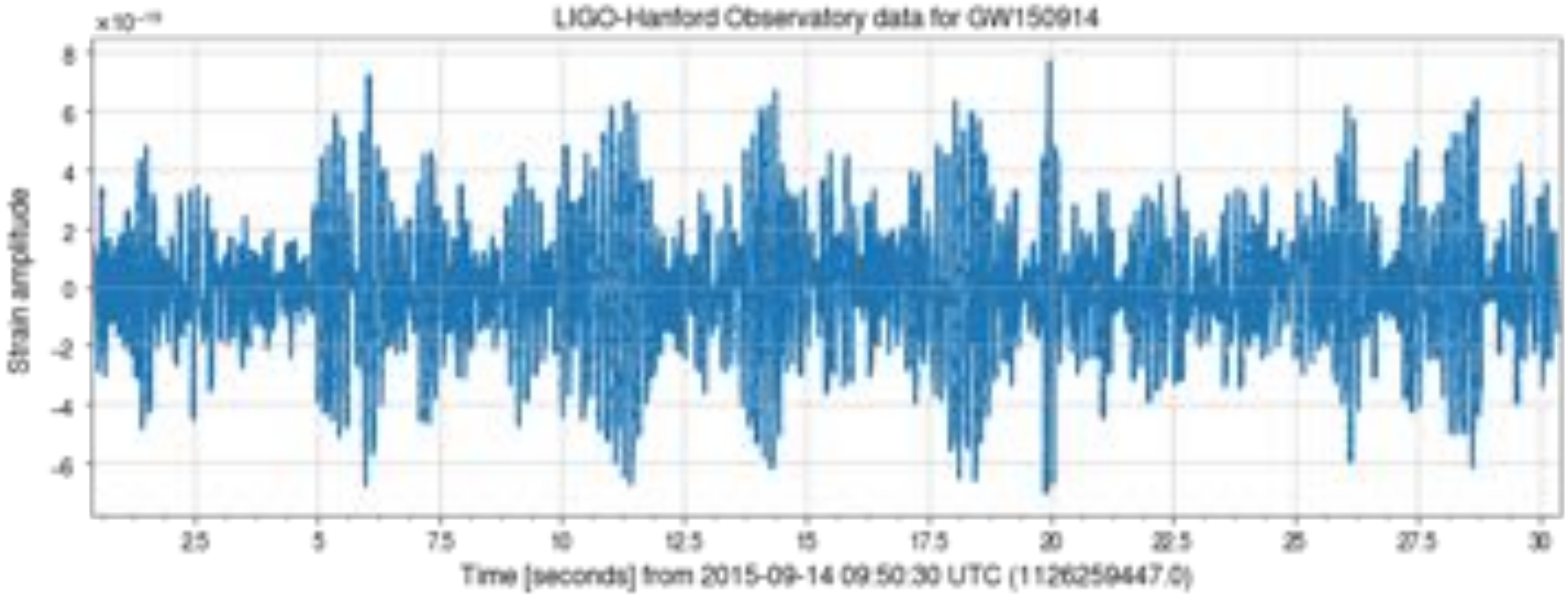
Time domain



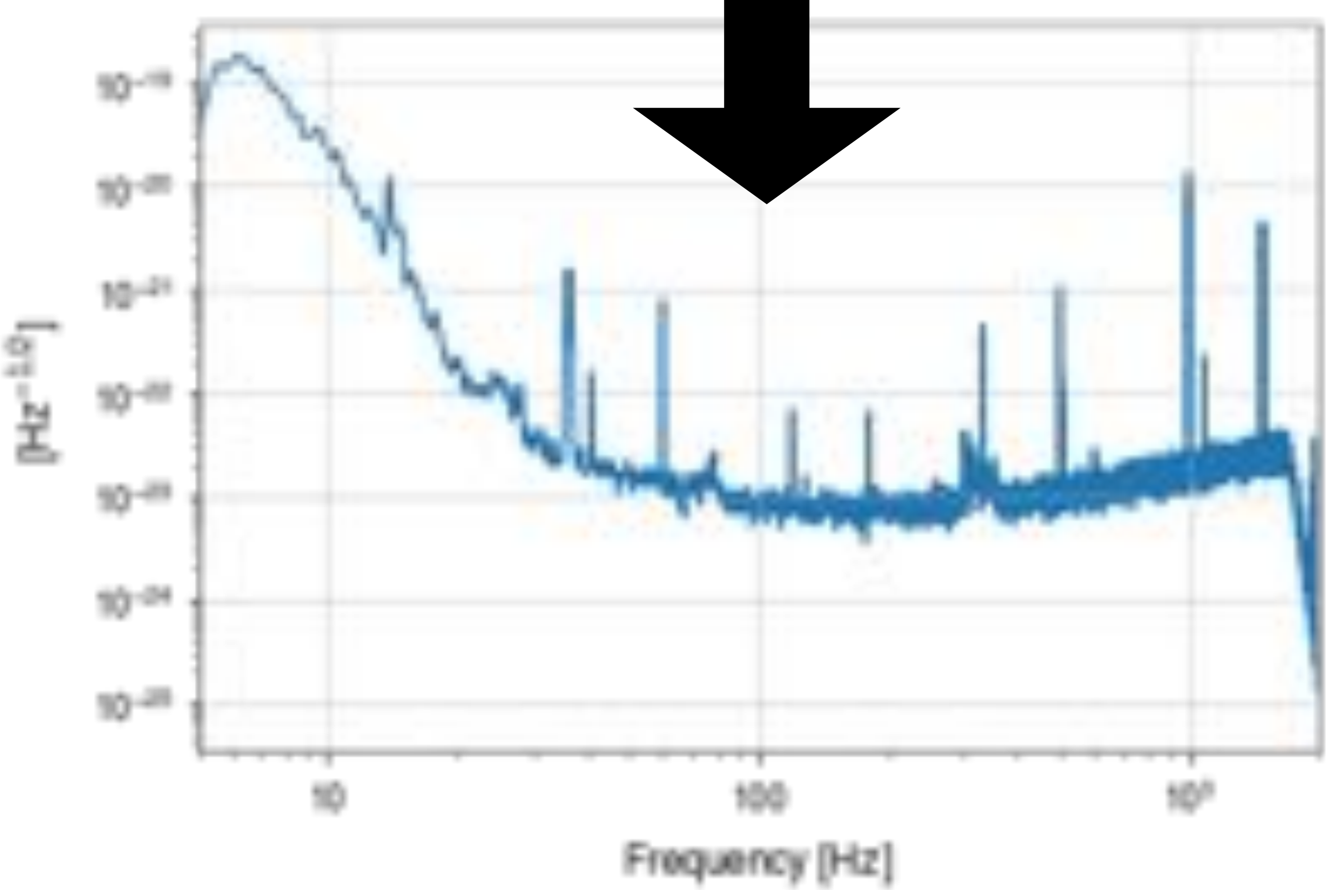
LIGO-Hanford Observatory data for GW150914
Strain amplitude
Time [seconds] from 2015-09-14 09:50:30 UTC (1126259447.0)

Time domain → Frequency domain

Time series (strain vs time)



Amplitude spectral density ($Hz^{-\frac{1}{2}}$ vs. frequency)



The Fourier Series

Any function can be represented as a sum of sines and cosines (with some coefficients that can also be functions).

$$f(x) = \sum_{n=0}^{\infty} A_n \cos\left(\frac{n\pi x}{L}\right) + \sum_{n=1}^{\infty} B_n \sin\left(\frac{n\pi x}{L}\right)$$

The Fourier Transform

When we transform our function or time (or space) into the "frequency domain", we are **projecting f(x) onto an orthogonal basis of sines and cosines**.

Fourier transform $$\tilde{x}(f) = \int_{-\infty}^{\infty} dt\, x(t) e^{-i2\pi f t}$$

The Fourier Transform

When we transform our function or time (or space) into the “frequency domain”, we are **projecting f(x) onto an orthogonal basis of sines and cosines**.

Fourier transform

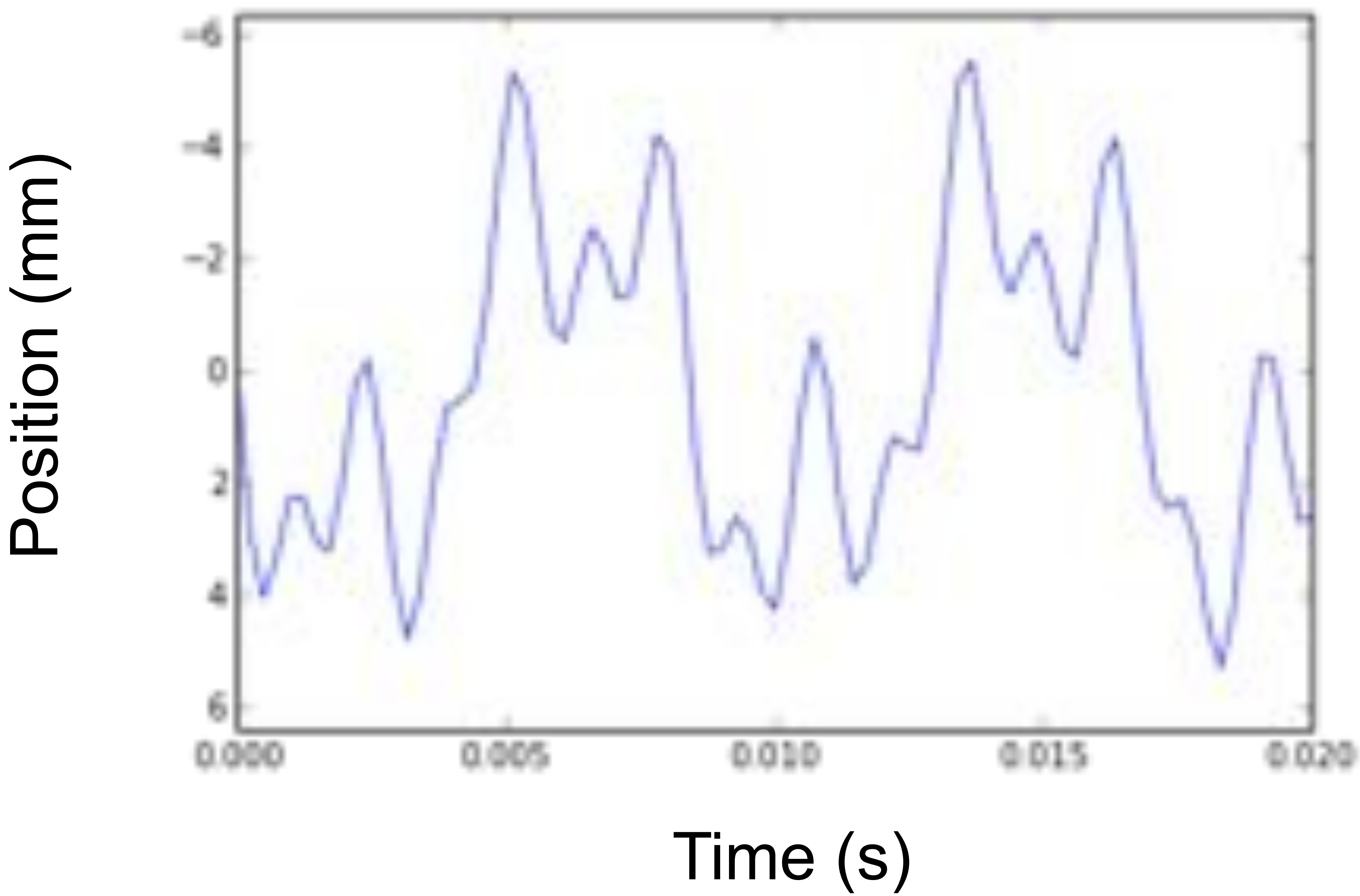
$$\tilde{x}(f) = \int_{-\infty}^{\infty} dt\, x(t) e^{-i2\pi f t}$$

Inverse Fourier transform

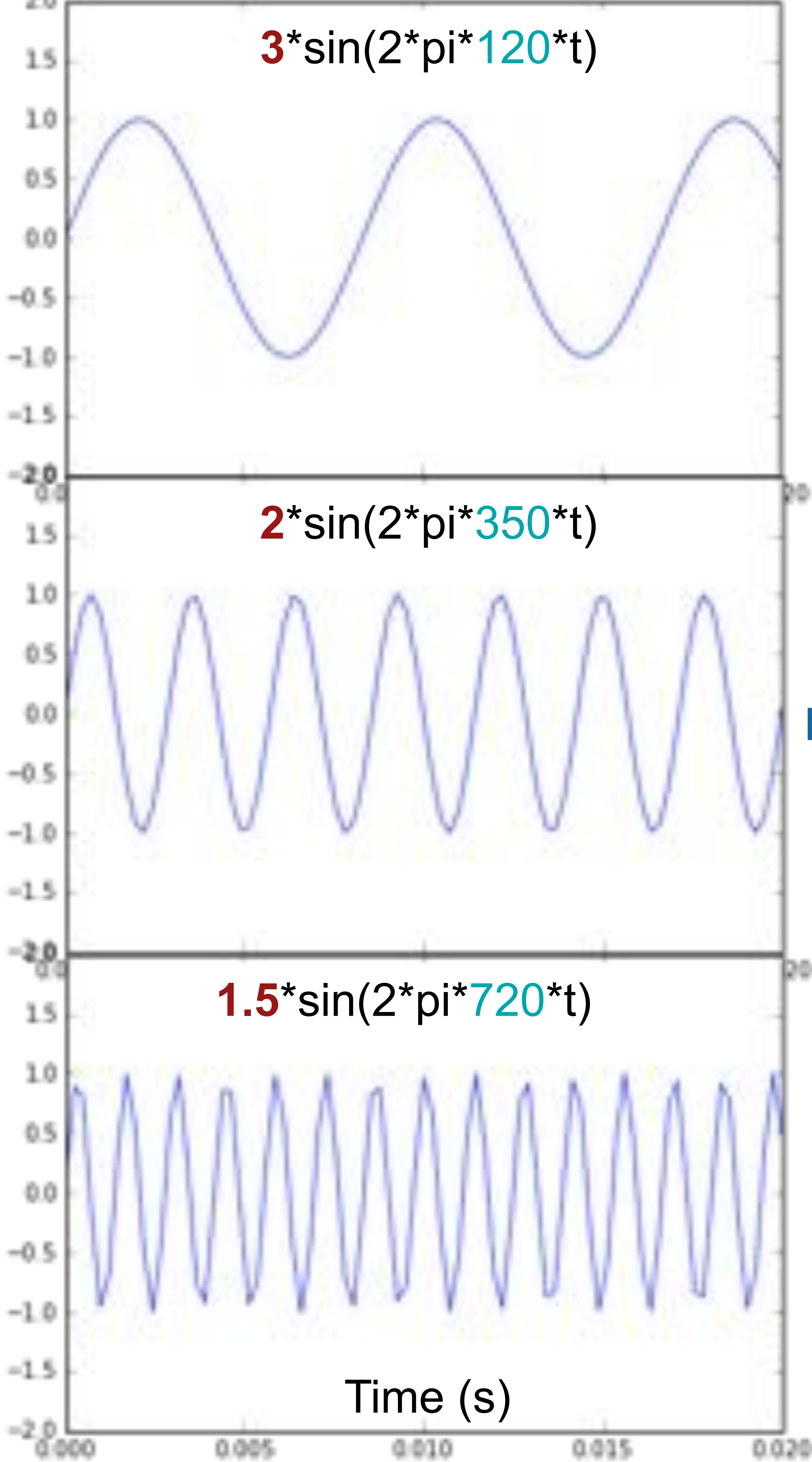
$$x(t) = \int_{-\infty}^{\infty} df\, \tilde{x}(f) e^{i2\pi f t}$$

Another way to think about it: when we take a Fourier transform **we are decomposing the function into its component frequencies**.

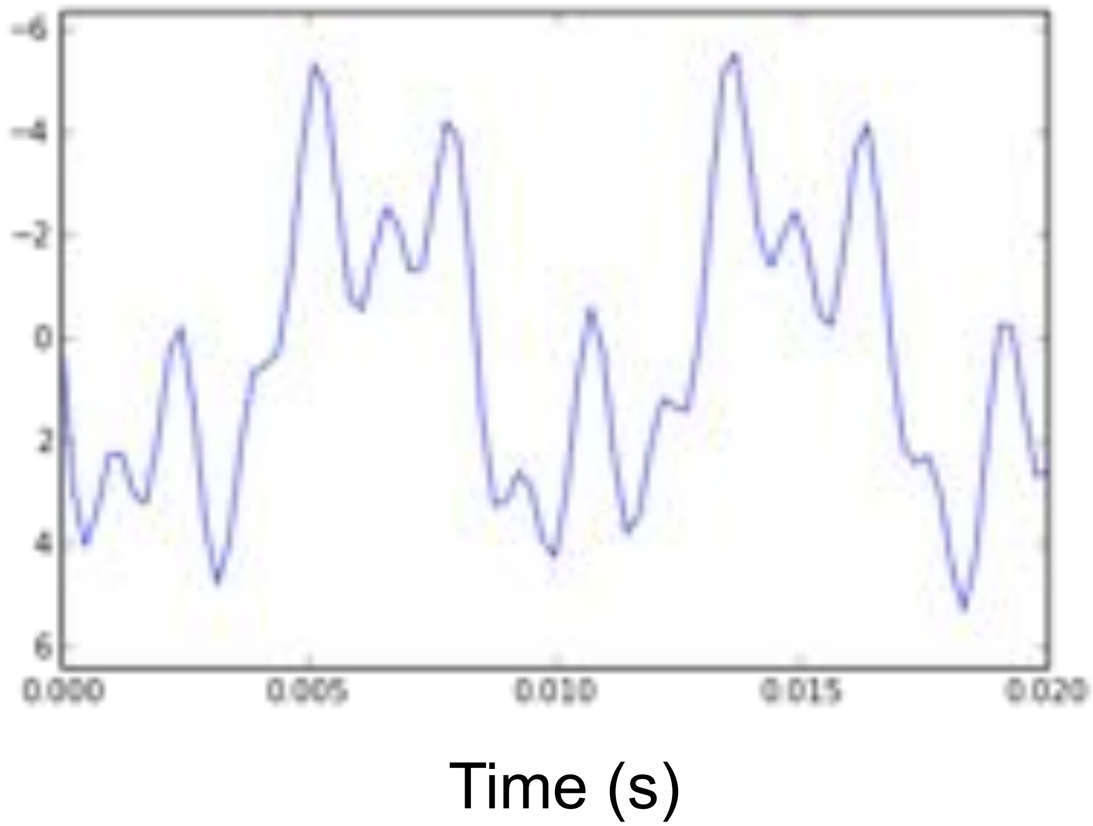
How would you describe this function?



Slide from J. Kanner

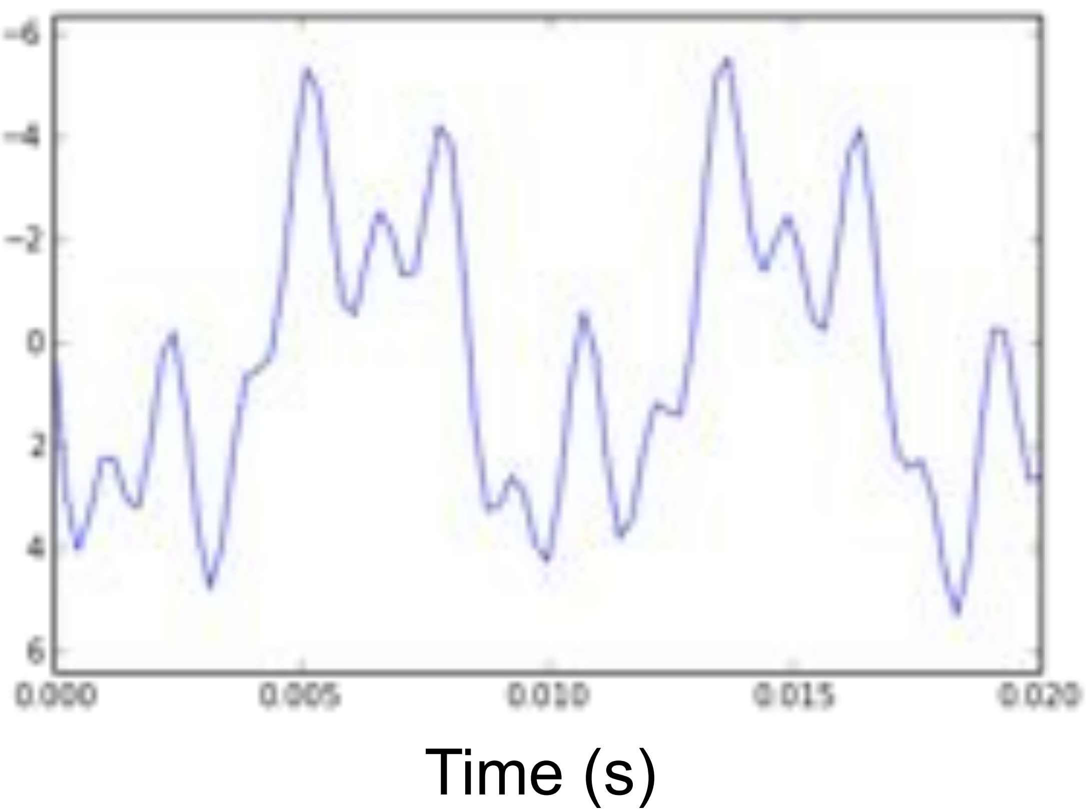


Our original function



Slide from J. Kanner

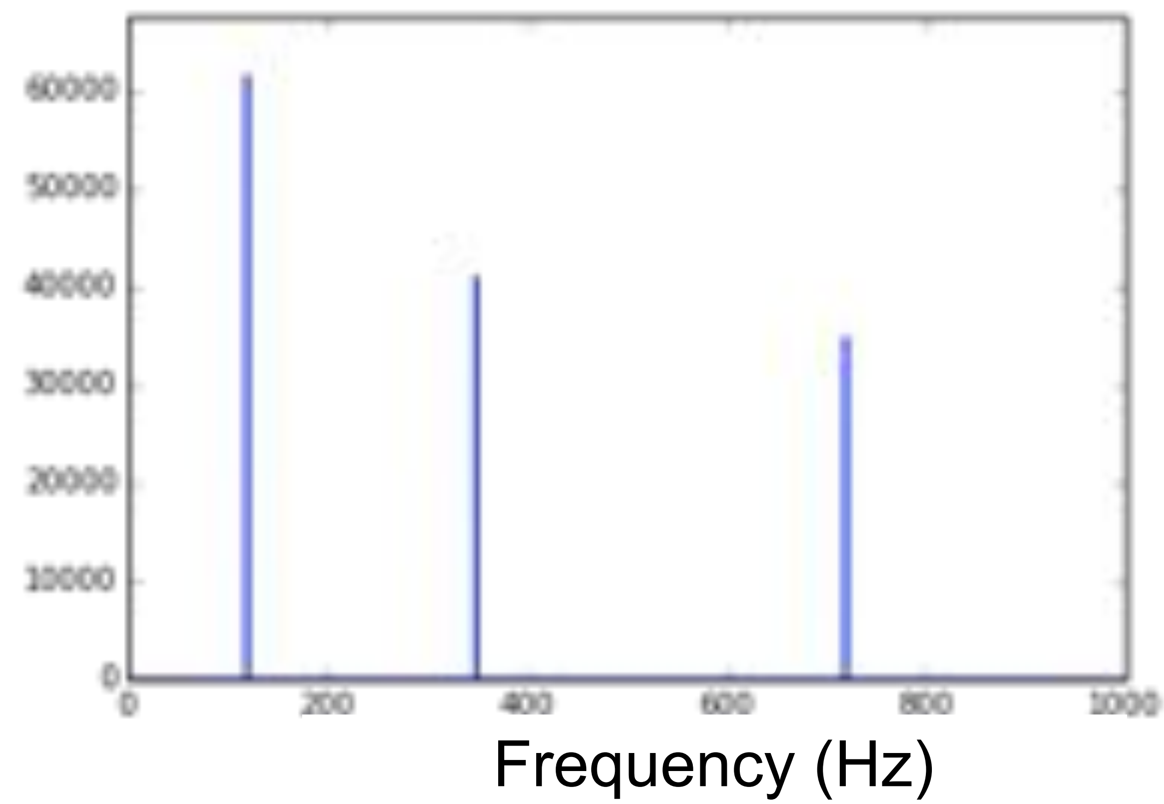
Time Domain



h(t) – Position as a function of time

h(t) = 3 * sin(2*pi*120*t) +
2 * sin(2*pi*350*t) +
1.5* sin(2*pi*720*t)

Frequency Domain



H(f) – Amplitude as a function of frequency

|H(120 Hz)| = 3
|H(350 Hz)| = 2
|H(720 Hz)| = 1.5
H(f) = 0 otherwise

Fourier Transform

Slide from J. Kanner

Power Spectral Density

Parseval's theorem:

$$\int_{-\infty}^{\infty} dt\, |x(t)|^2 = \int_{-\infty}^{\infty} df\, |\tilde{x}(f)|^2$$

$\Rightarrow$ Total energy in the data can be calculated in either time domain or frequency domain

Units:

$|\tilde{x}(f)|^2$ Energy spectral density
(normalize by 1/T to get power)
Signal energy per unit frequency (per Hz)

$|\tilde{x}(f)|$ $\propto$ Amplitude spectral density
(sqrt of power for each discrete frequency)
Signal amplitude per unit frequency (per *sqrt Hz*)

Slide adapted from P. Shawhan

Estimating the PSD

Step 0: Take a **Fast Fourier Transform (FFT)**, which is any algorithm useful for quickly estimating the **Discrete Fourier Transform** that describes a **discrete time series**.

$$
\begin{aligned}
X_k &= \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N}kn} \\
&= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)],
\end{aligned}
$$

Need to shift our thinking to discretized data; **frequency bins instead of continuous smooth sinusoids**

Estimating the PSD

Step 1: Apply a window to your data (if it's linear! as a time series is) to prevent **spectral leakage** from the assumption the signal is periodic.

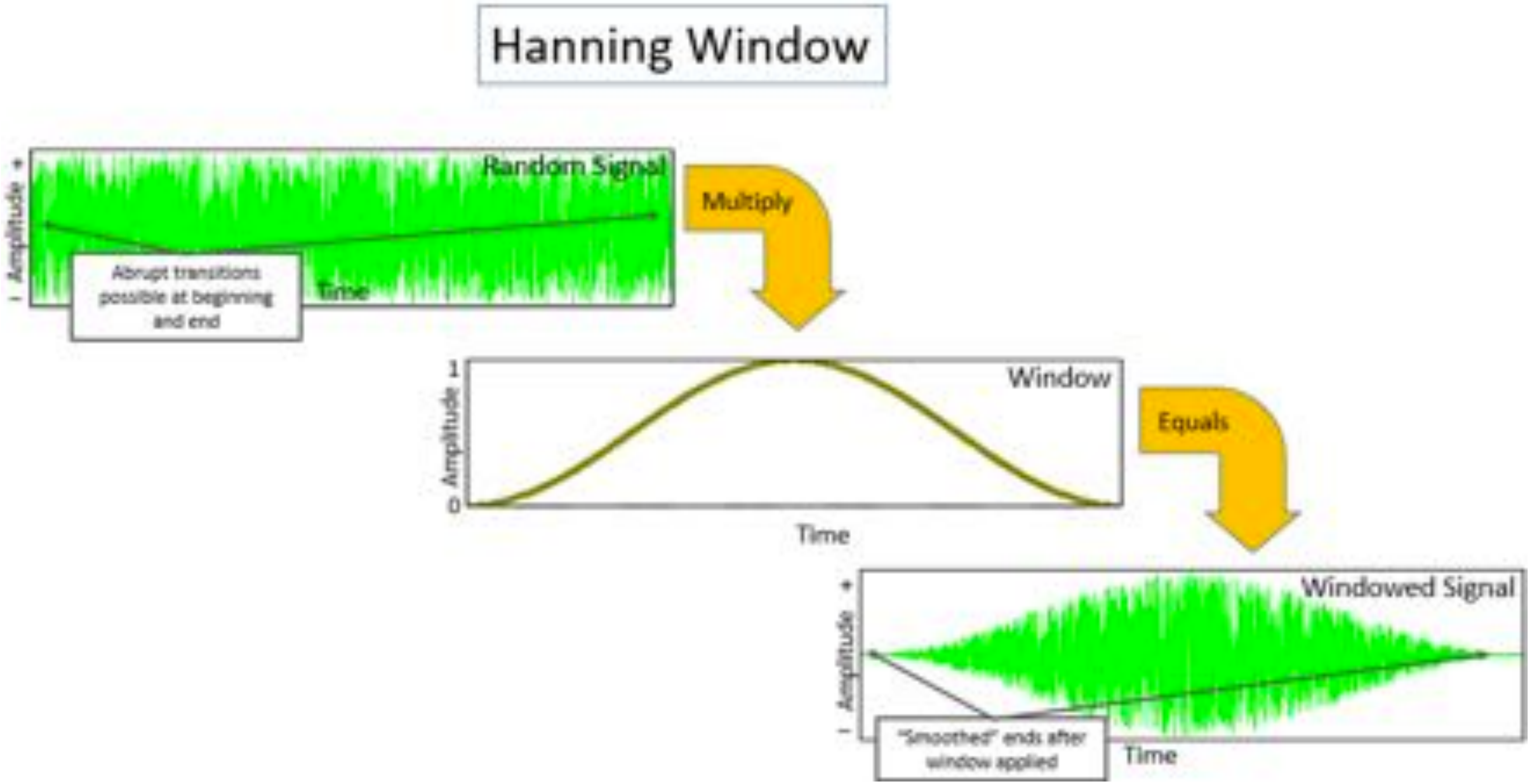


Image: Siemens.

Estimating the PSD

A single windowed FFT is unbiased (i.e. will give the correct mean PSD), but has **high variance**.

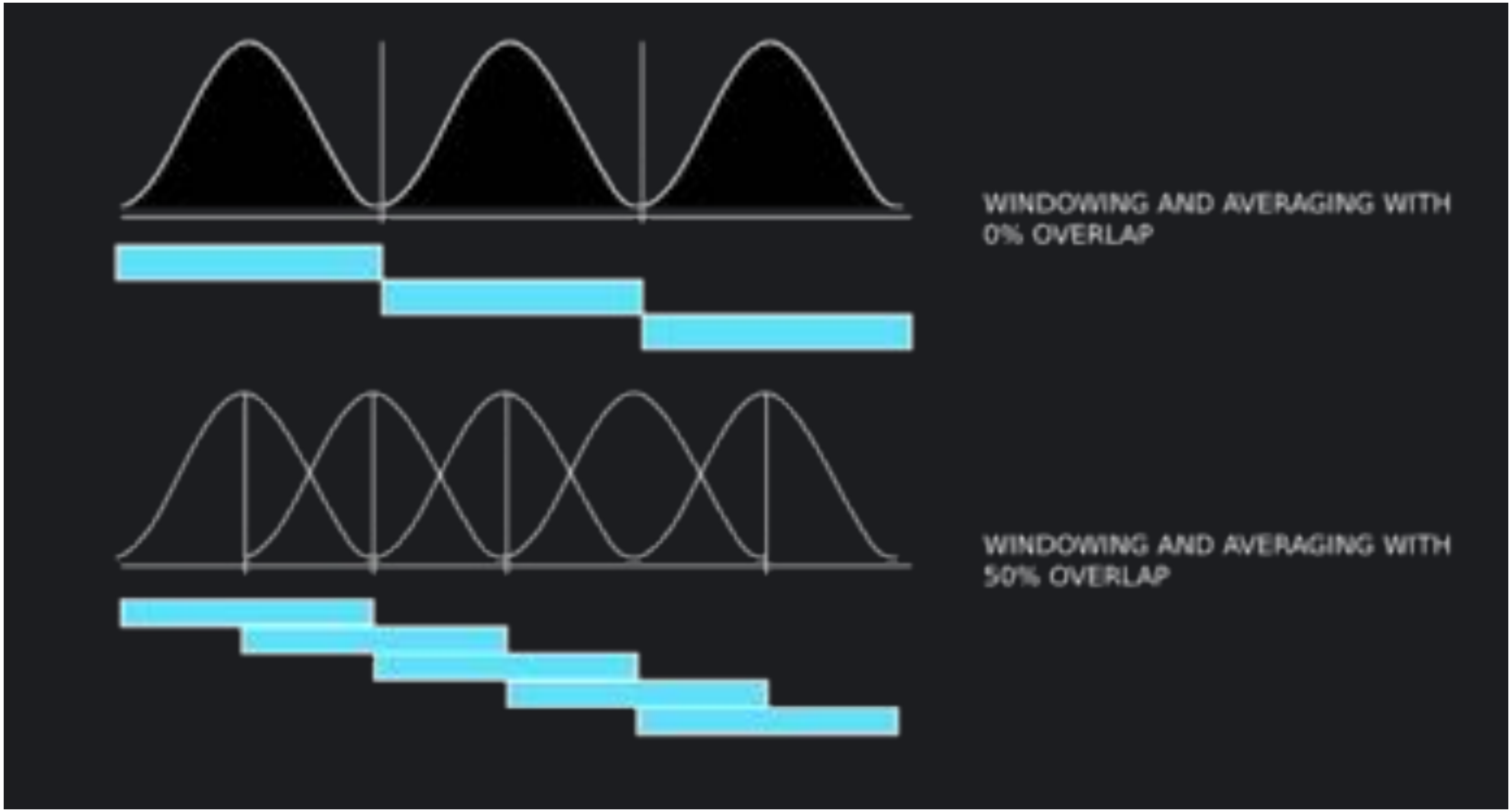
Solution: average several FFTs!

Step 2: Divide your data into shorter time segments; take a windowed FFT of each, and average these together.

Note you lose some frequency resolution this way.

Welch's method averages the mean value for each frequency bin across FFTs, with some overlap in the data analyzed.

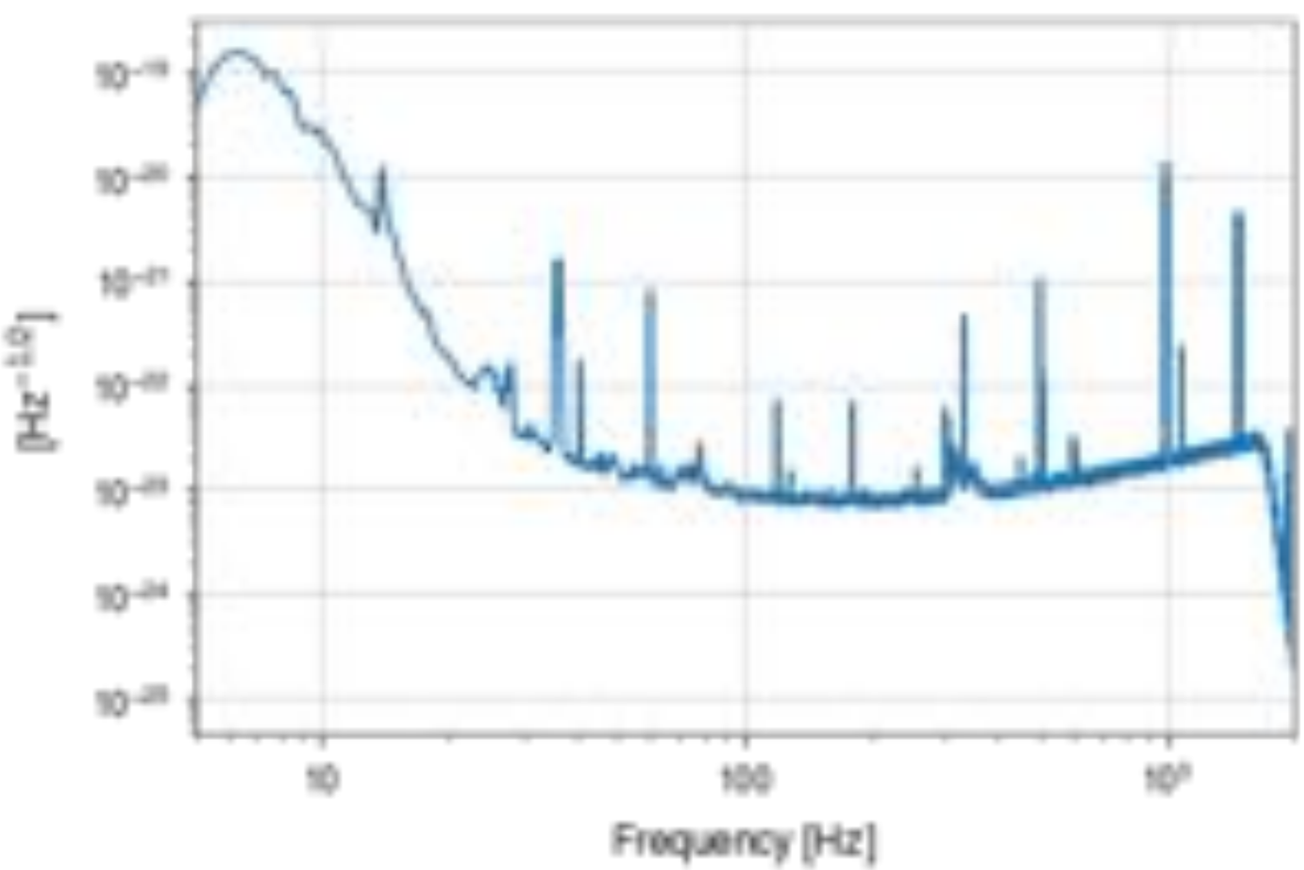
Averaging FFTs



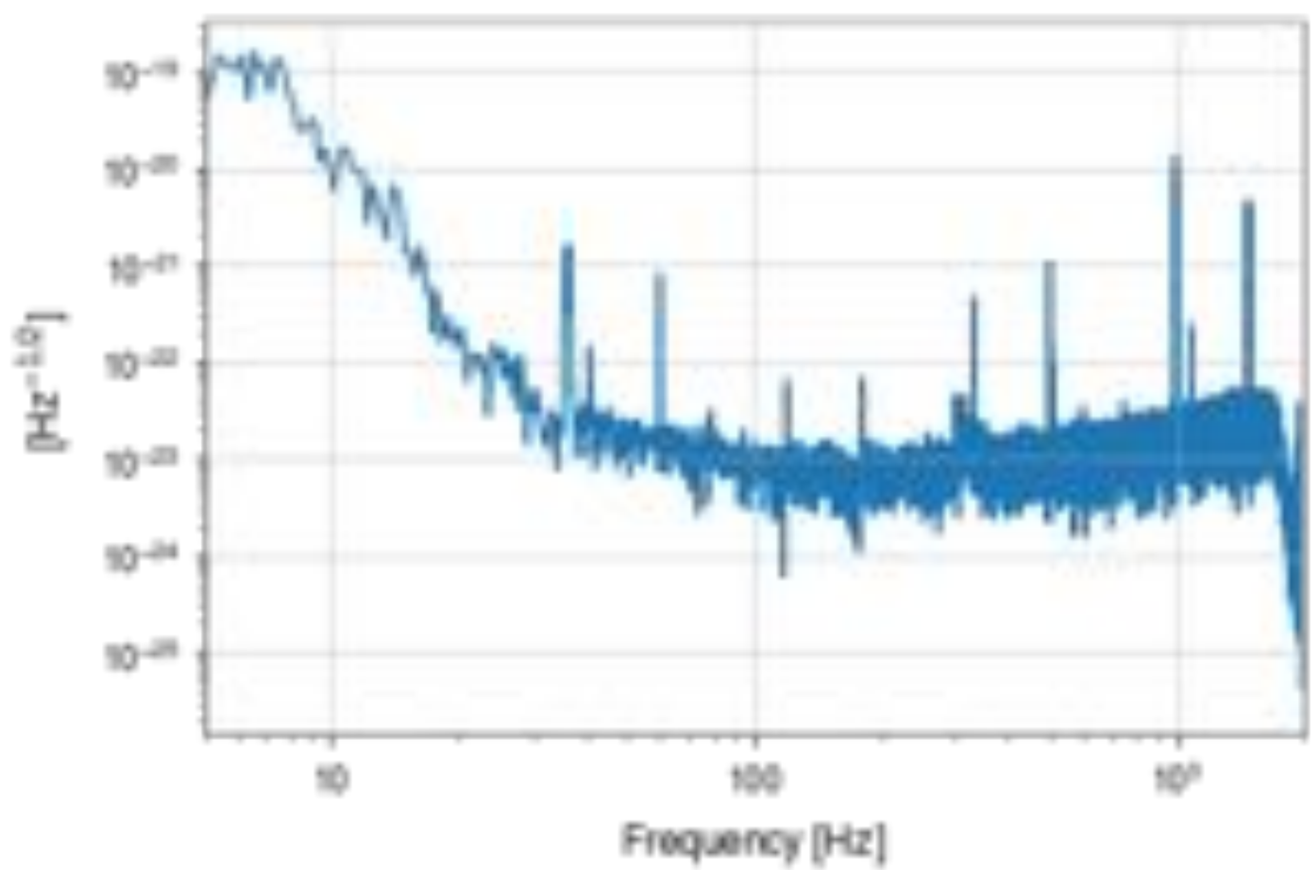
https://www.dewesoft.com/pro/course/spectral-analysis-using-the-fft-29?page=10

Example: Averaging FFTs

FFT length = 5 seconds
Overlap = 2 seconds
120 averages

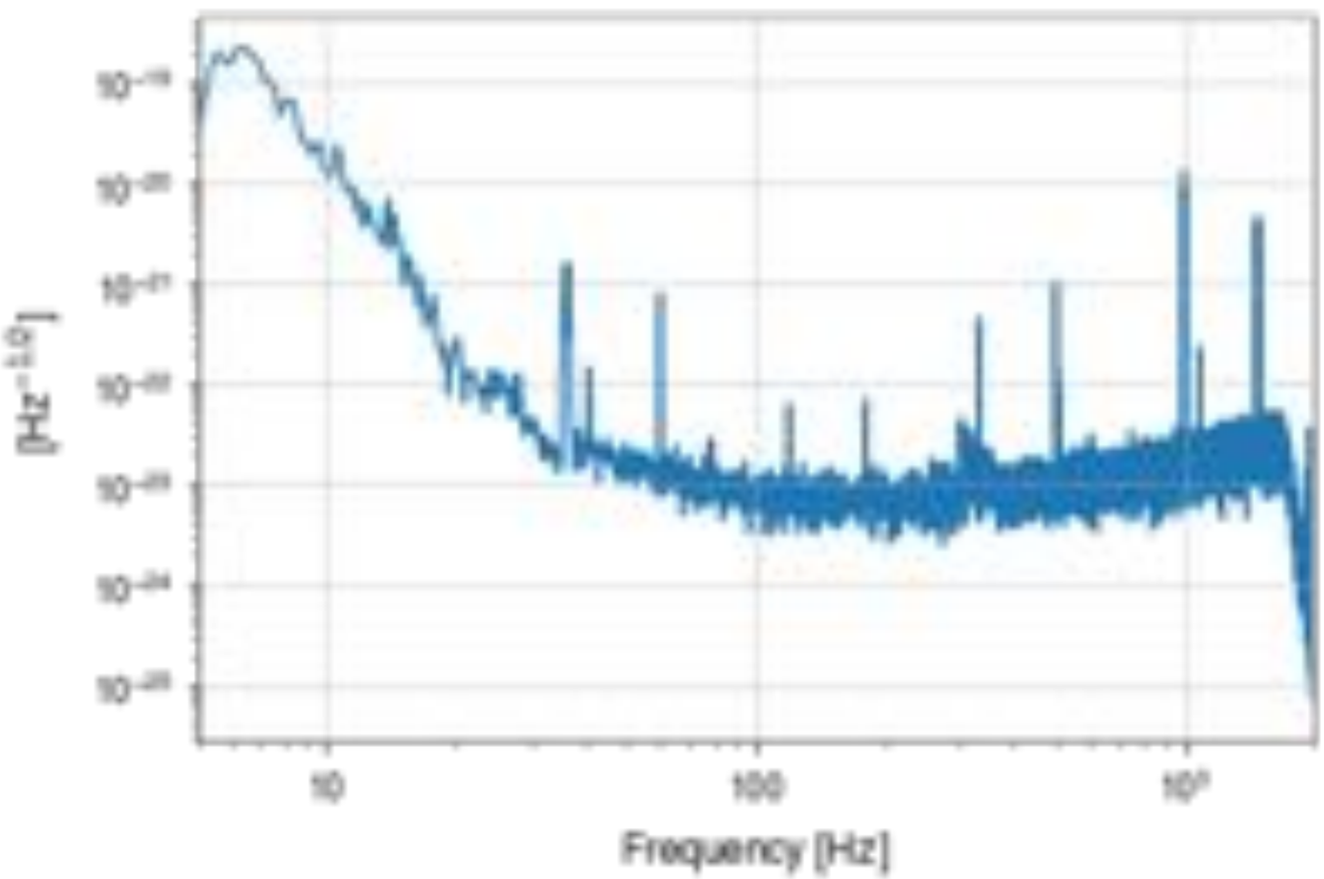


FFT length = 5 seconds
Overlap = 2 seconds
4 averages

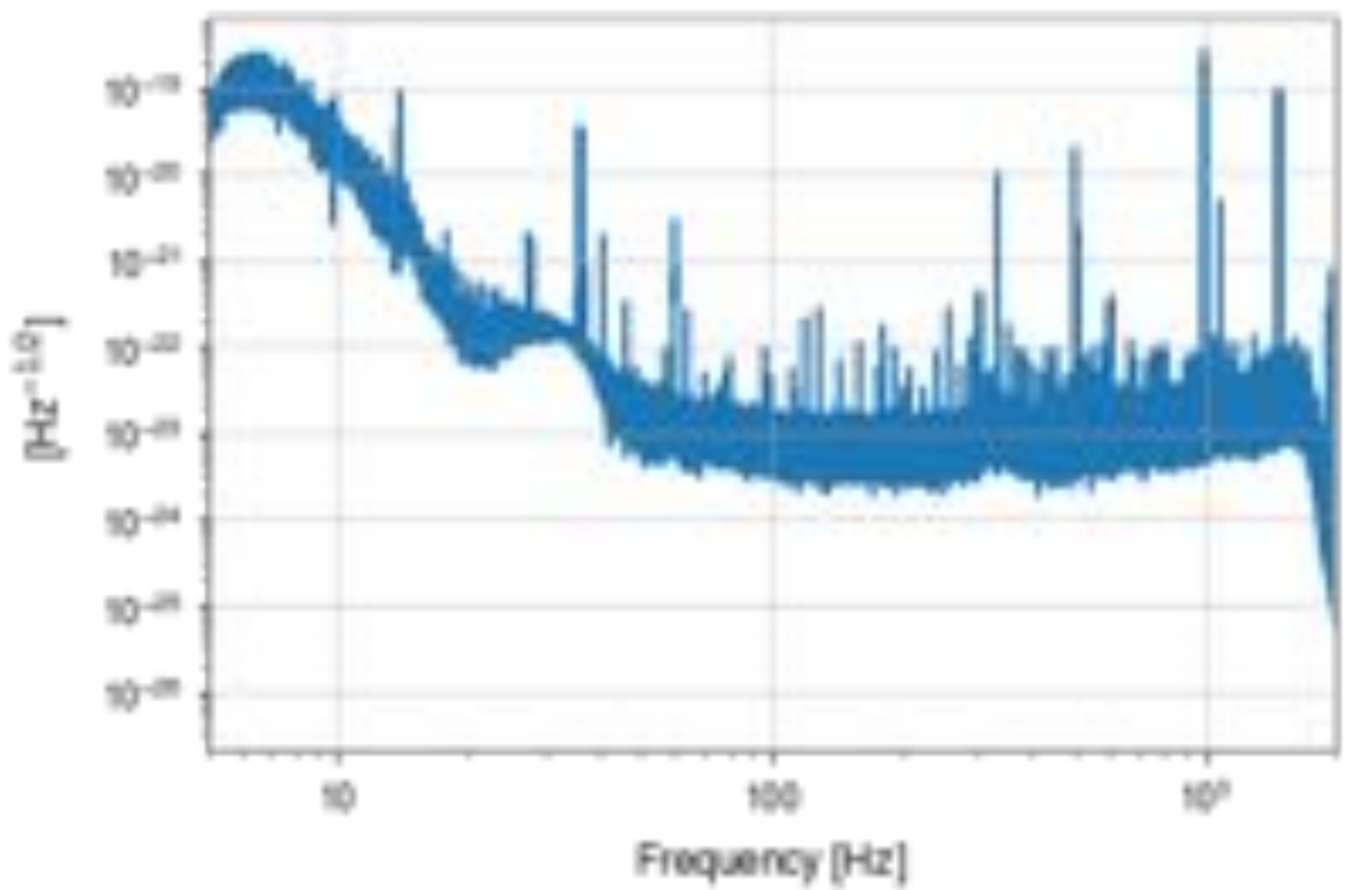


Example: FFT length

FFT length = 5 seconds
Overlap = 2 seconds
6 averages



FFT length = 2048 seconds
Overlap = 1024 seconds
6 averages



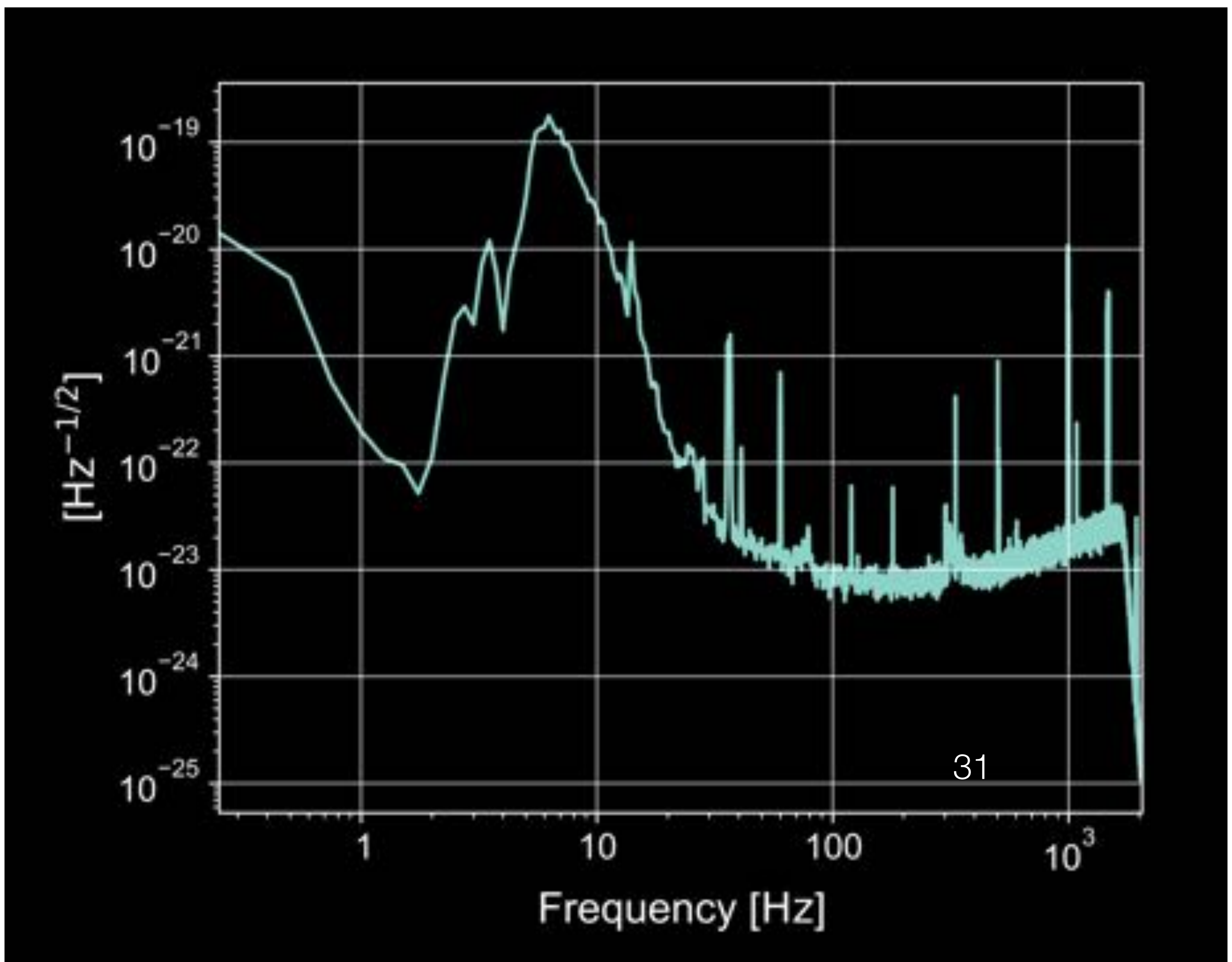
Signal processing with GWpy

GWpy provides **FFT wrappers** to estimate frequency-domain content of data:

FFT length (s)

Overlap between averages (s)

```
>>> asd = data.asd(4, 2)
```



Can also specify:

Time window
(default = Hanning)

Averaging method
(default = Welch)

Signal processing with GWpy

GWpy provides simple signal-processing methods to filter data

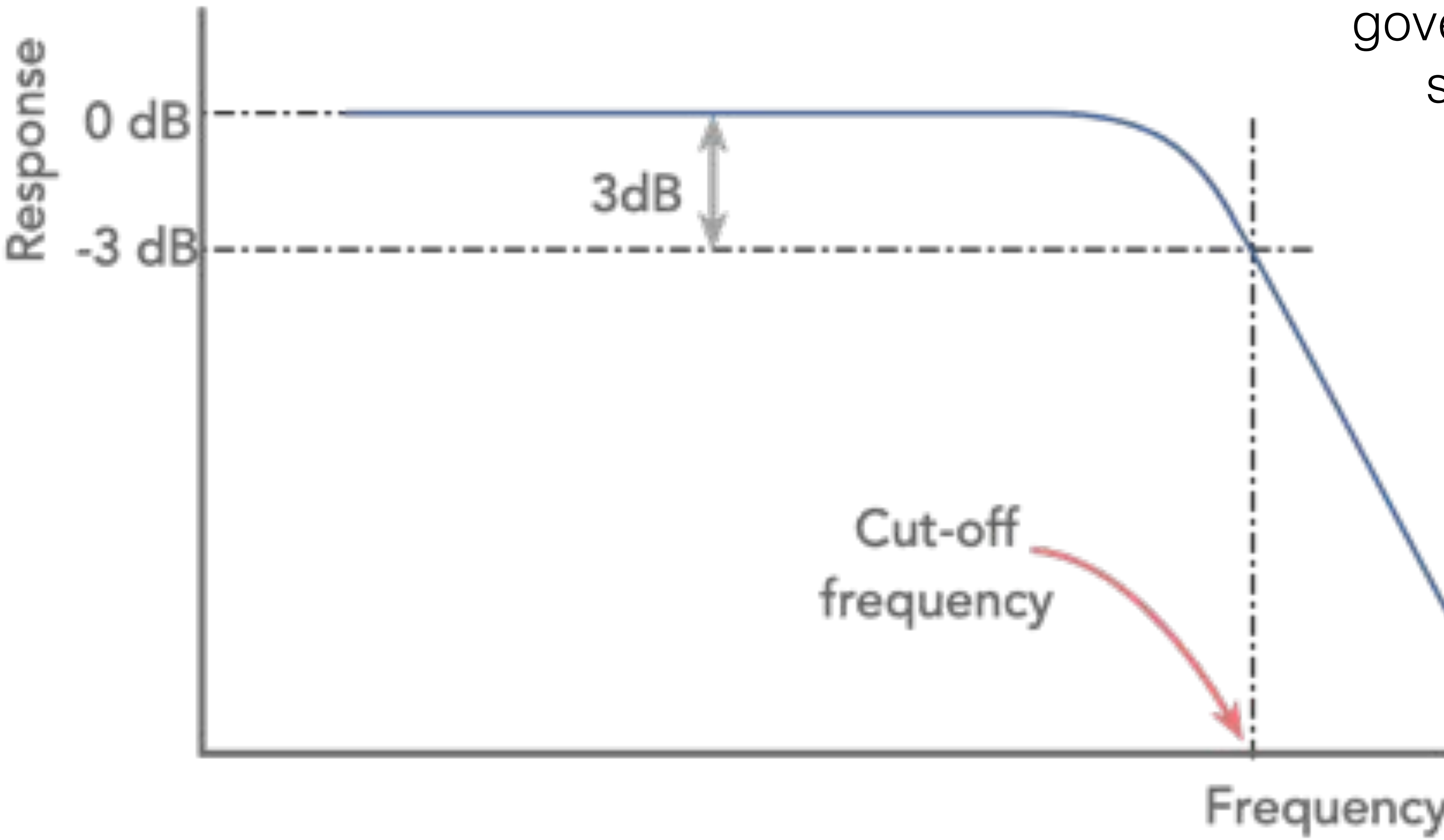
- lowpass, highpass
- bandpass
- notch
- whitening

You will use these tools to design frequency domain filters later!

High pass and low pass filters

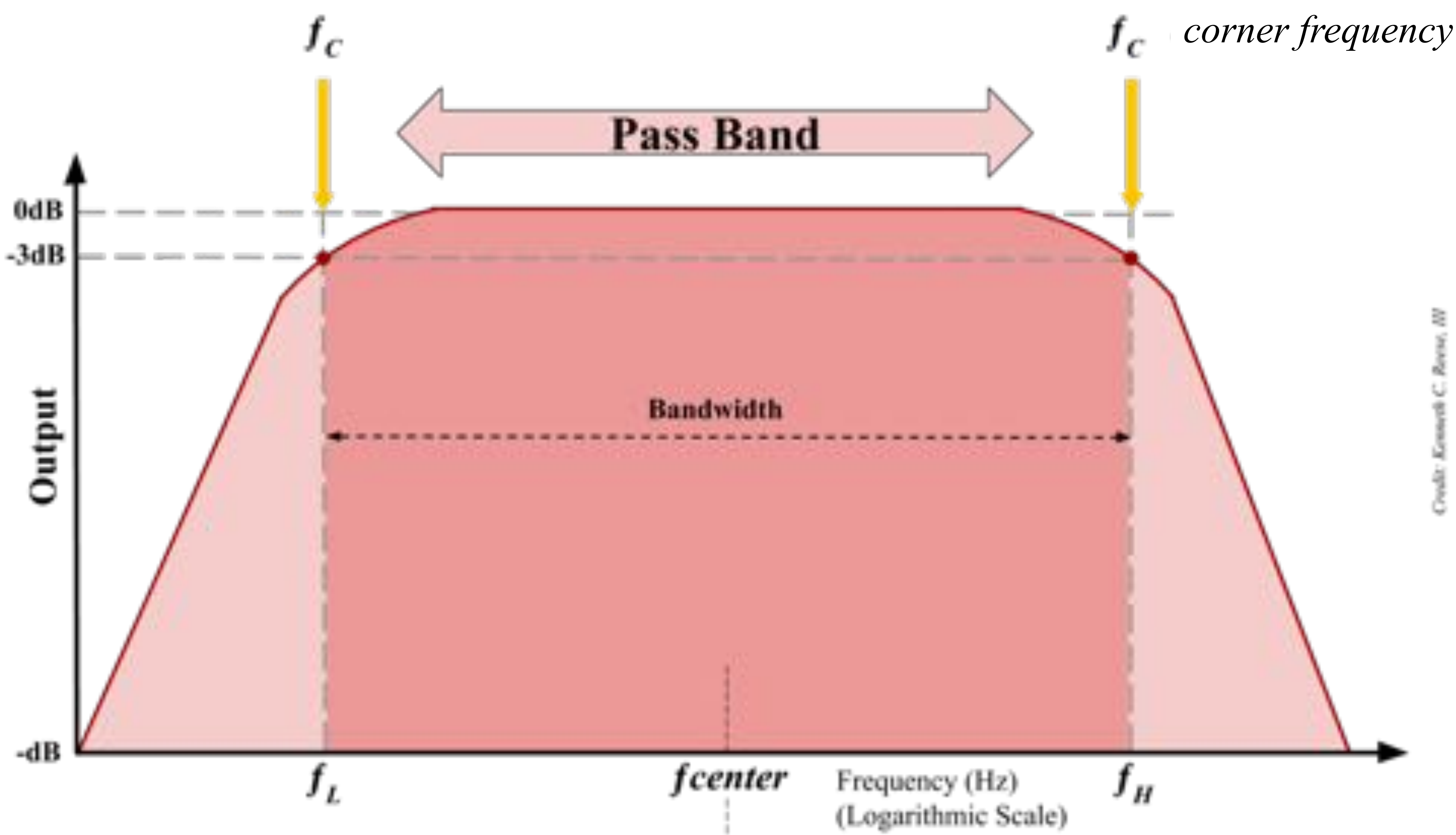
A “low pass” filter

Filter “order” governs the slope



Credit: electronics-notes

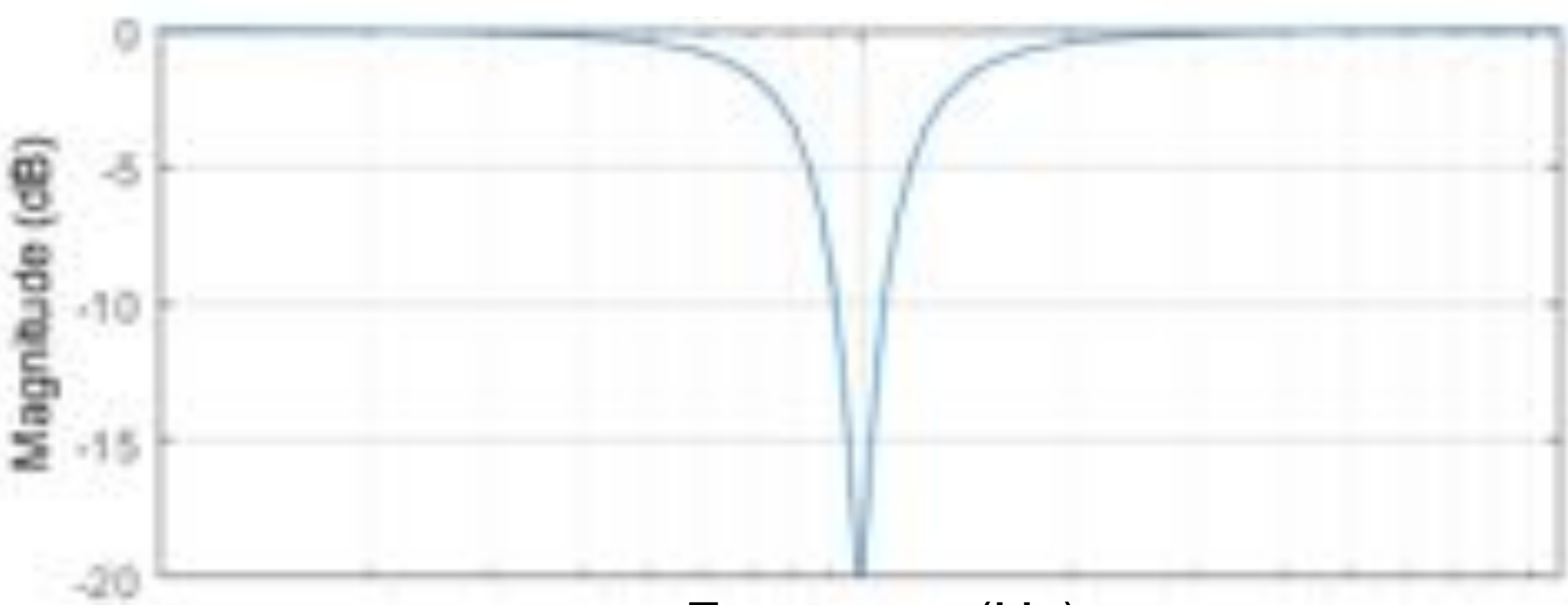
Bandpass filter



Notch filter

Not quite the inverse of the bandpass filter

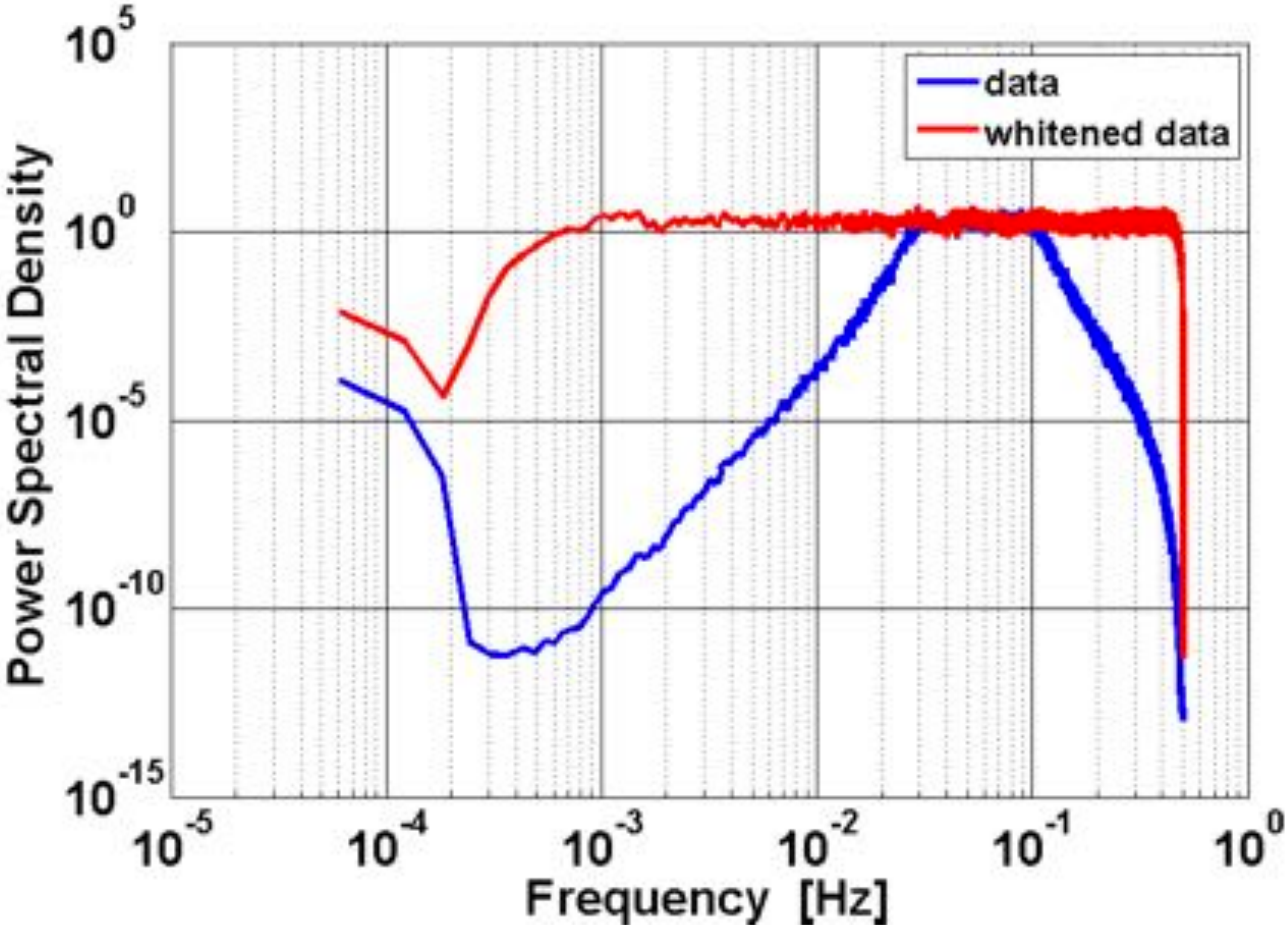
Only described by one frequency (and the filter order)



Frequency (Hz)

Credit: electronics-notes

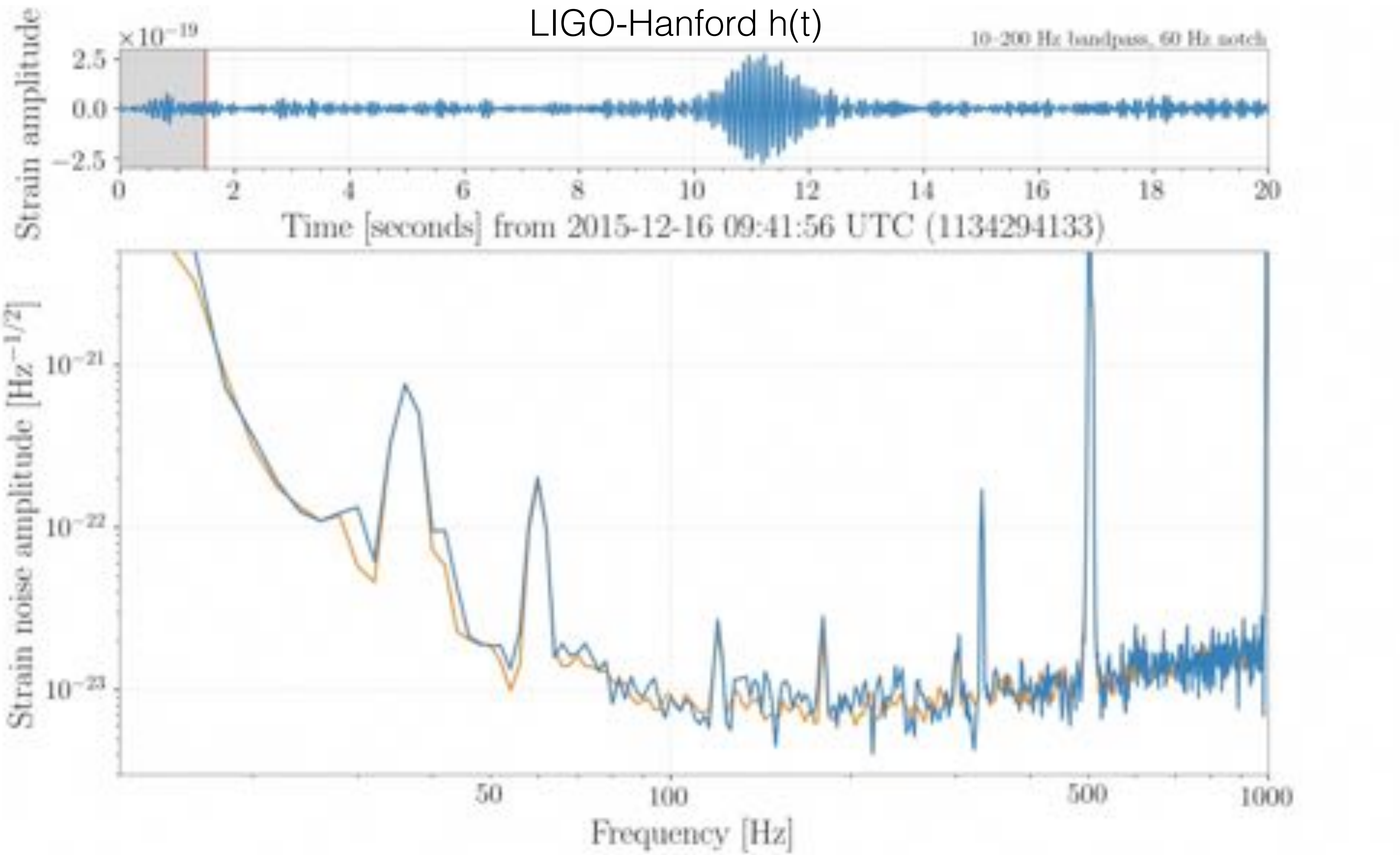
Whitening



Credit: LISA data tools

LIGO data in time and frequency

LIGO-Hanford h(t)

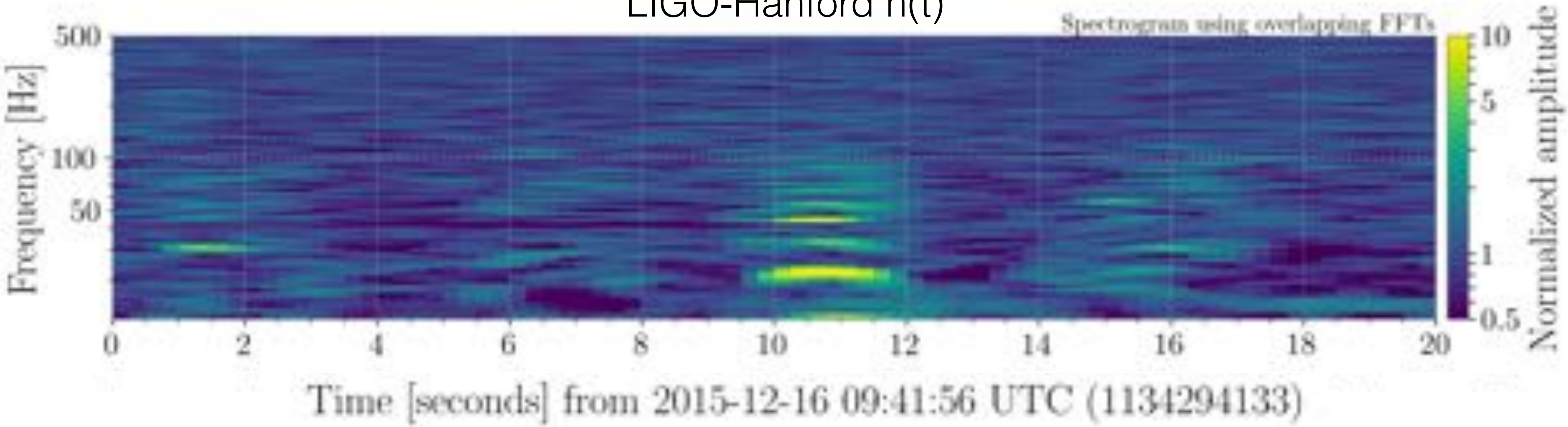


Made with GWpy by Duncan Macleod. Code: https://git.io/gwpy-ligo-scattering-animation

0.5 second FFT; 5 averages covering 1.5 seconds; 50% overlap

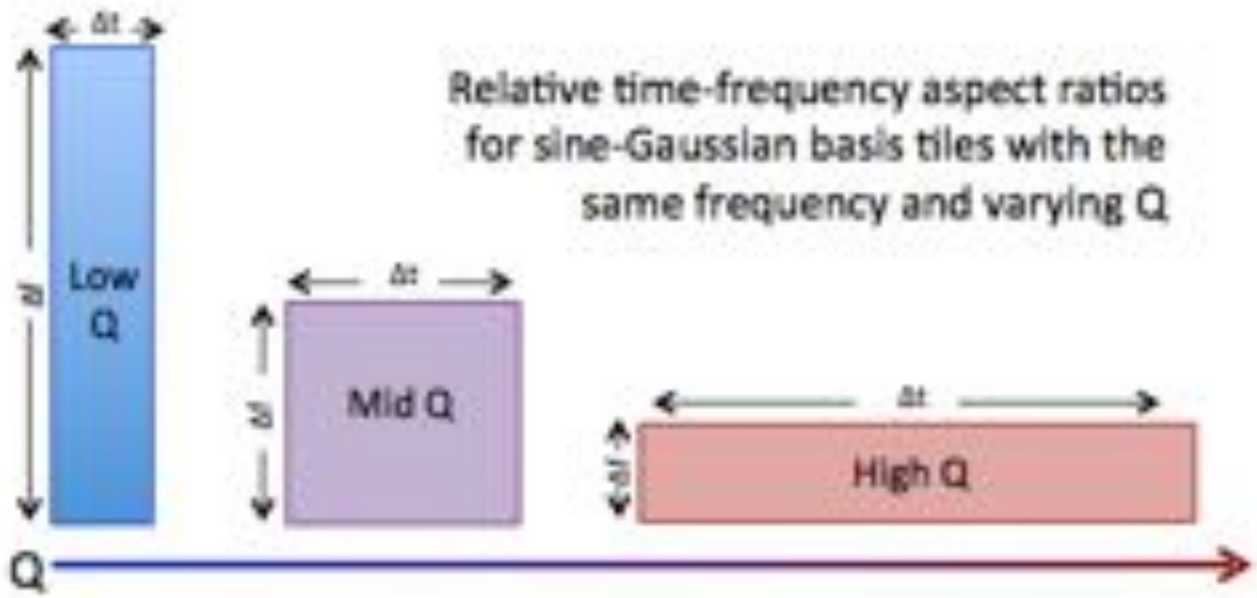
Time-frequency spectrogram

LIGO-Hanford h(t)



Made with GWpy by Duncan Macleod

The Q transform

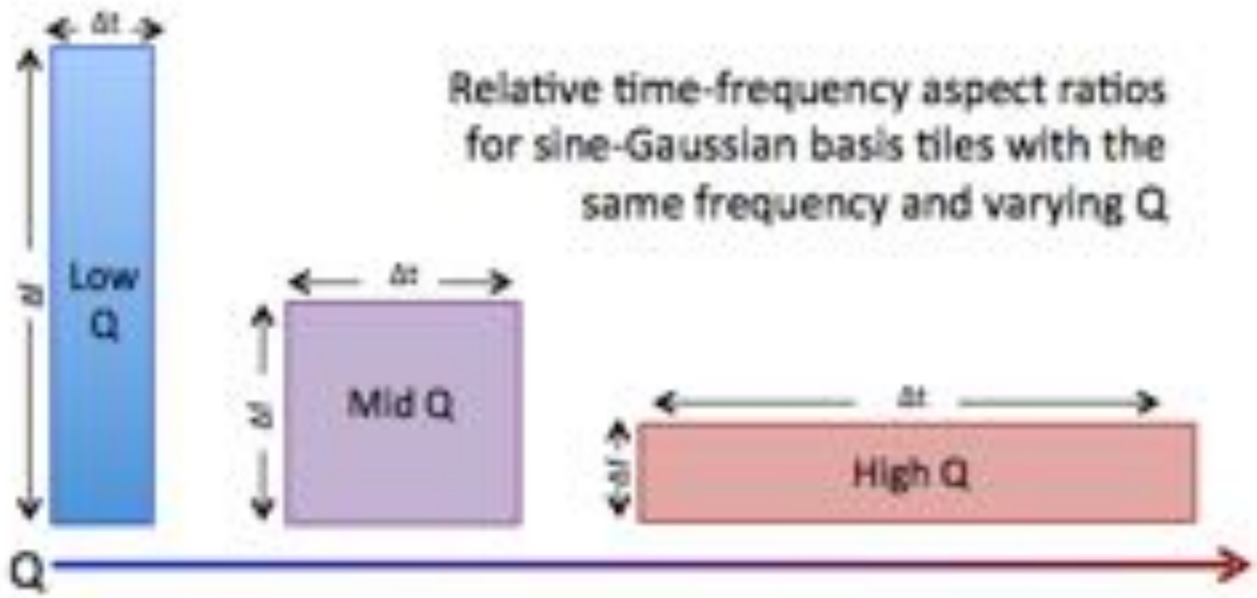


S. Chatterji et al. CQG (2010)
Images: McIver

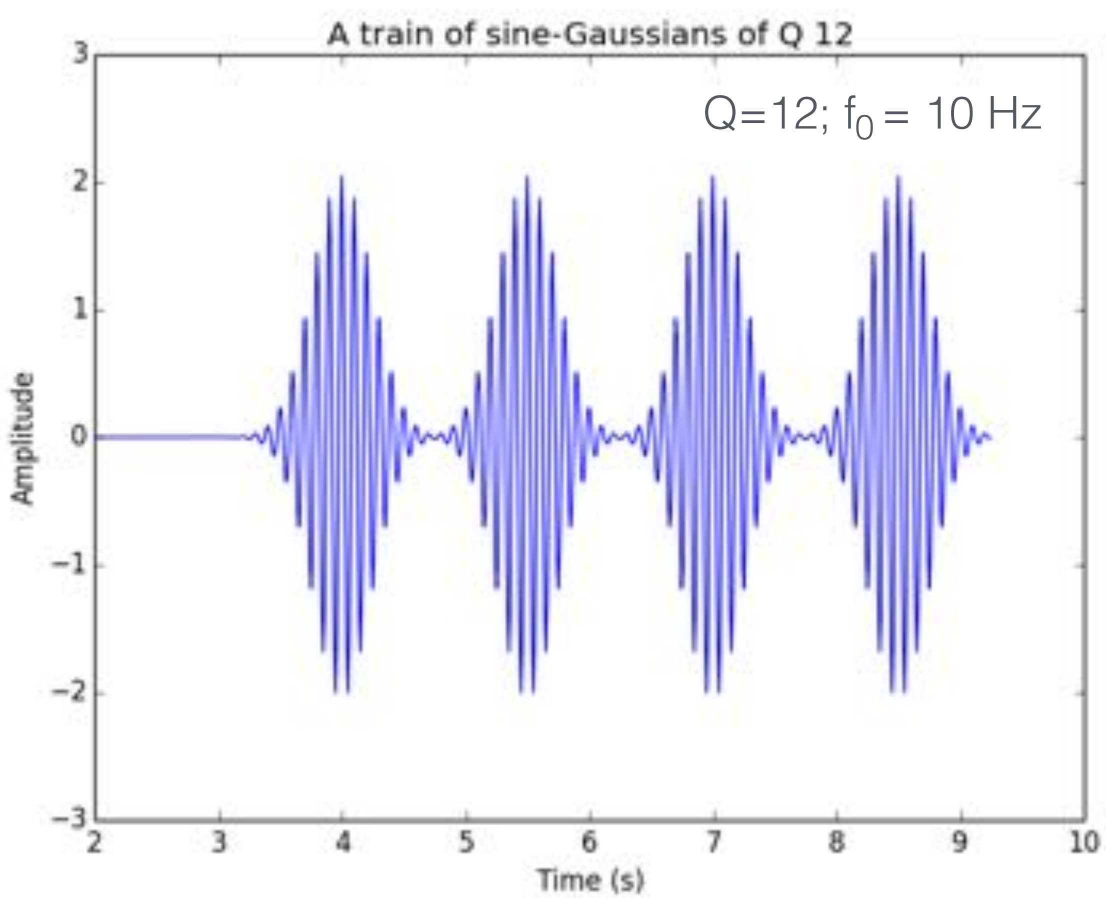
$$Q = \frac{f_0}{\Delta f}$$

The Q transform

S. Chatterji et al. CQG (2010)
Images: McIver

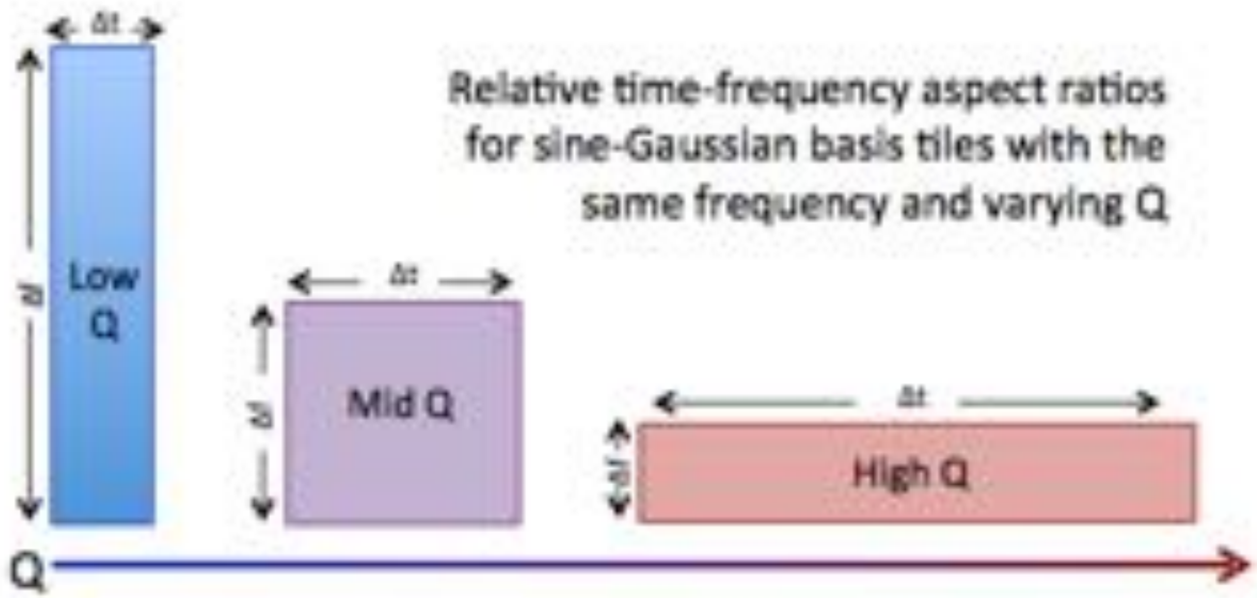


$$Q = \frac{f_0}{\Delta f}$$

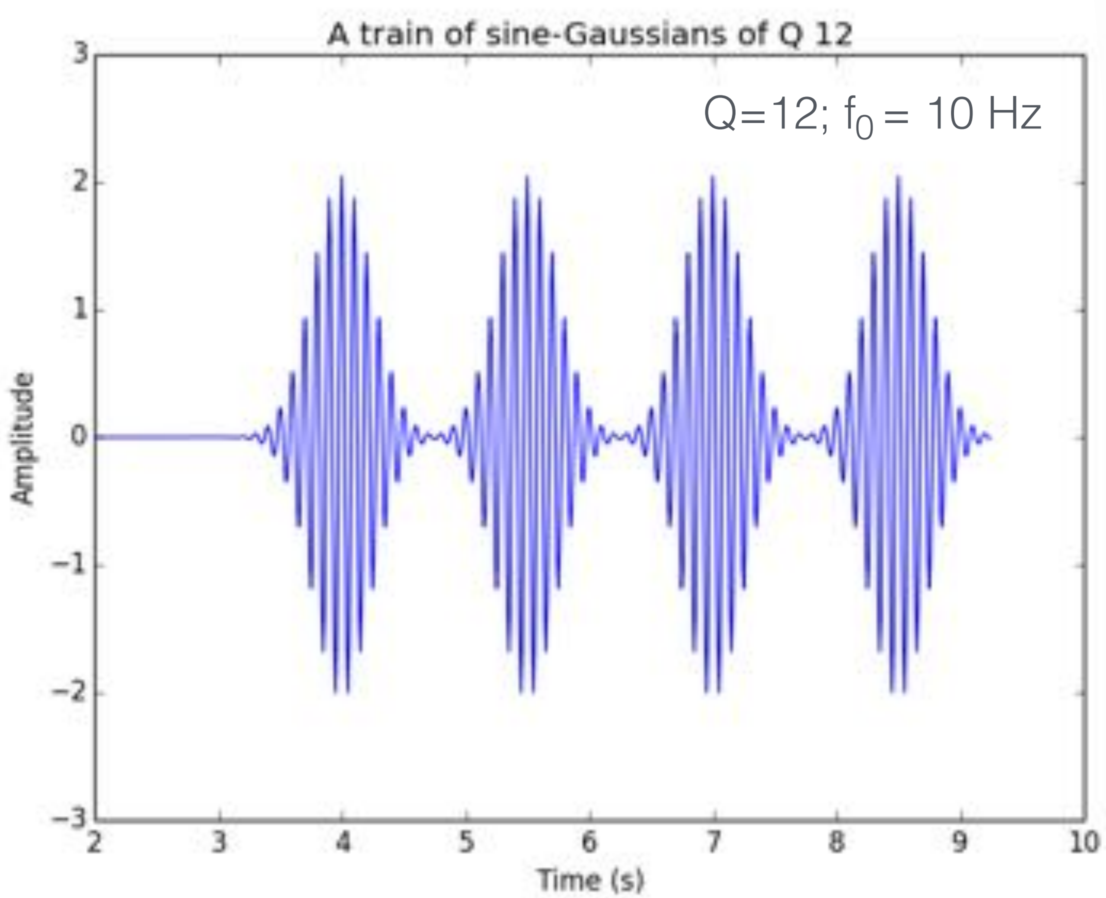


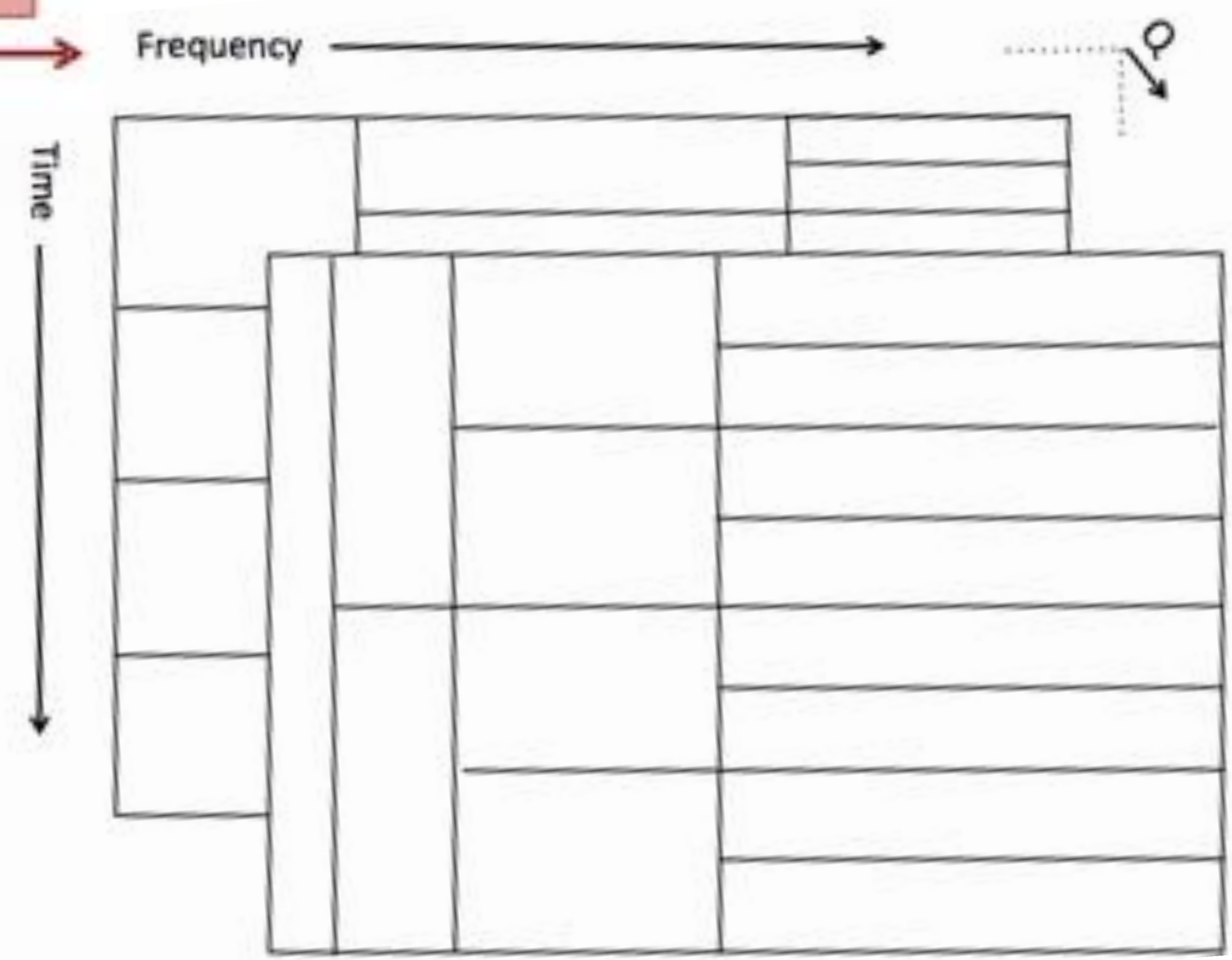
The Q transform

S. Chatterji et al. CQG (2010)
Images: McIver



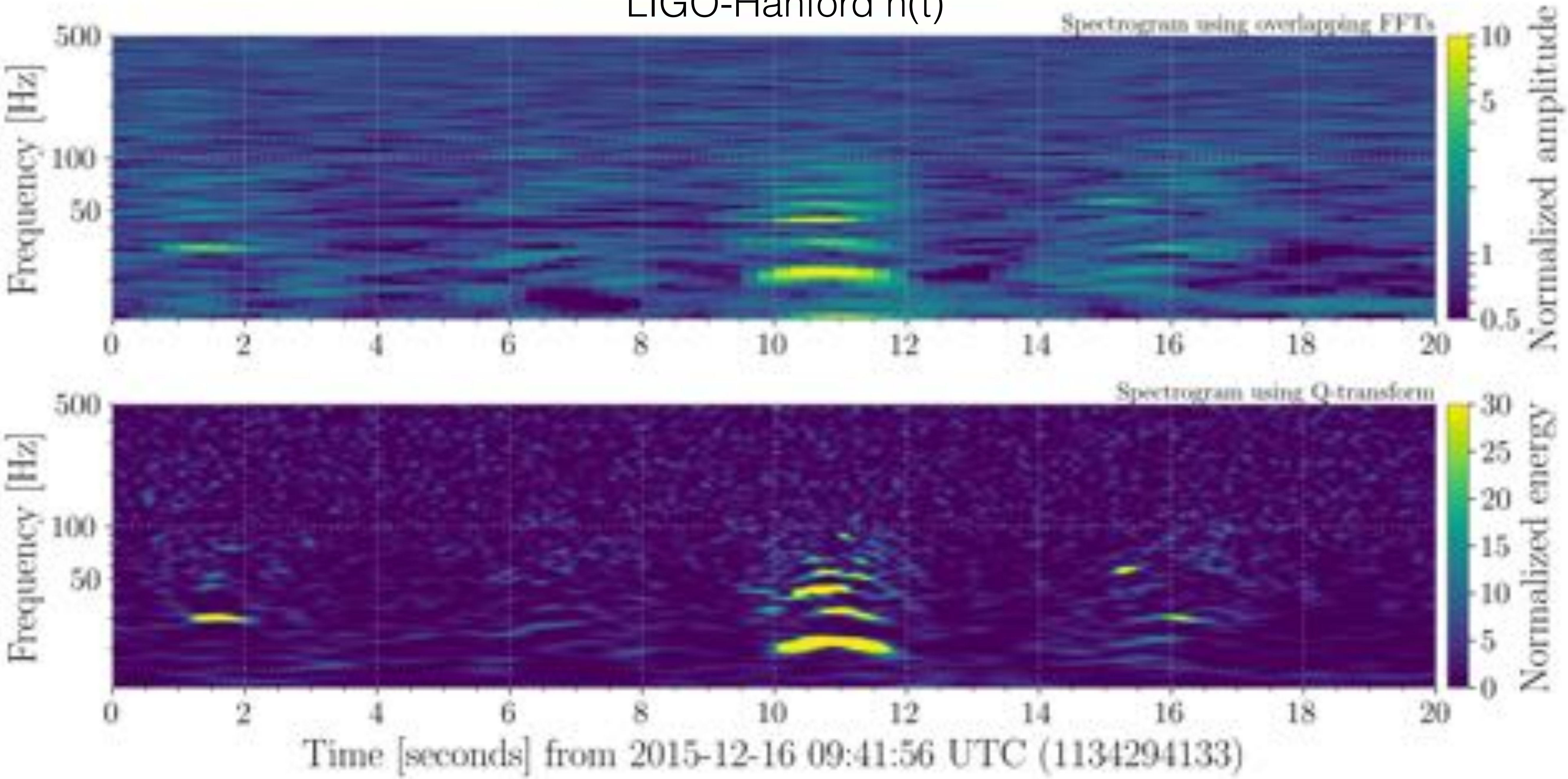
$$Q = \frac{f_0}{\Delta f}$$



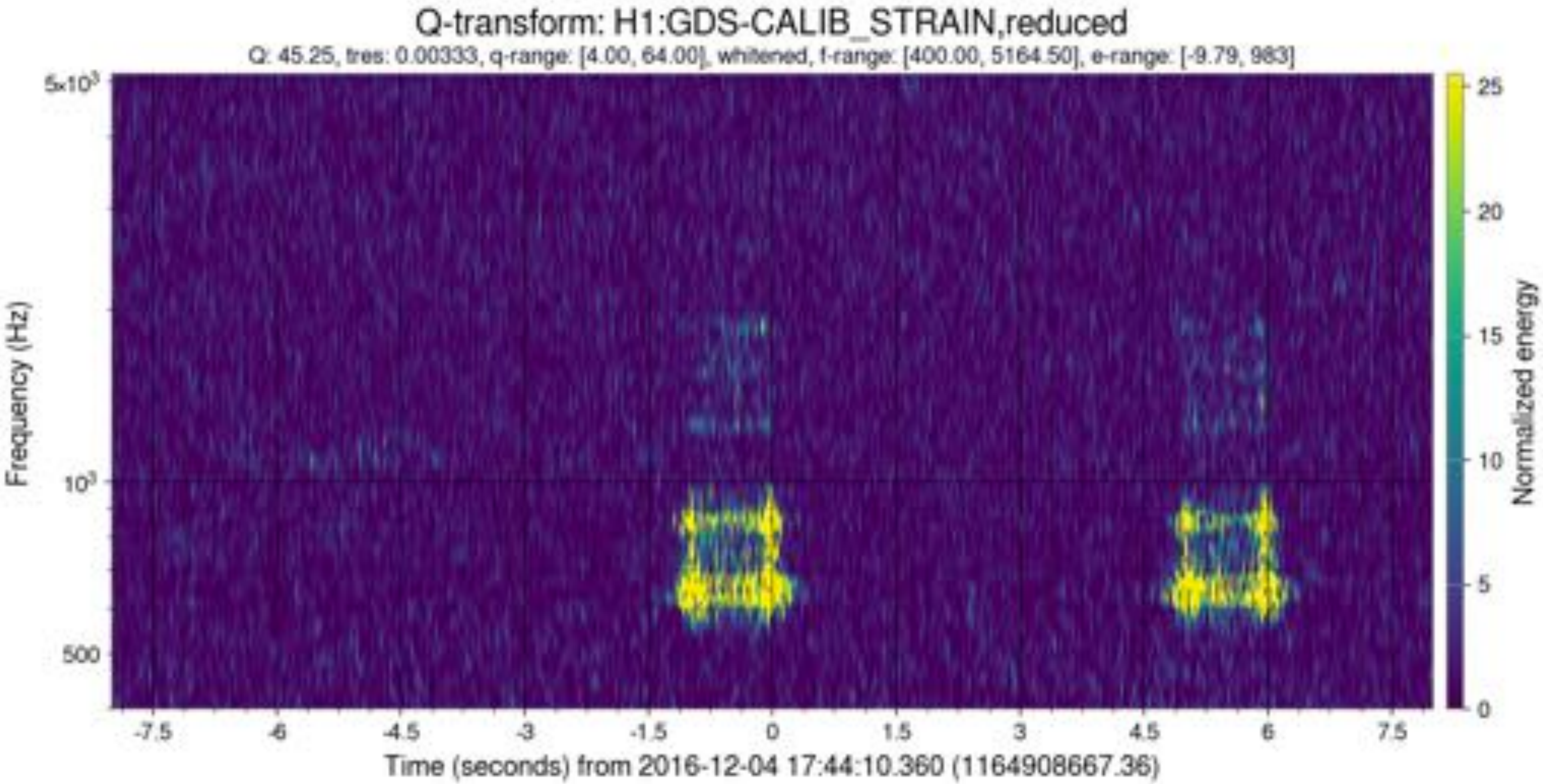


Time-frequency spectrograms

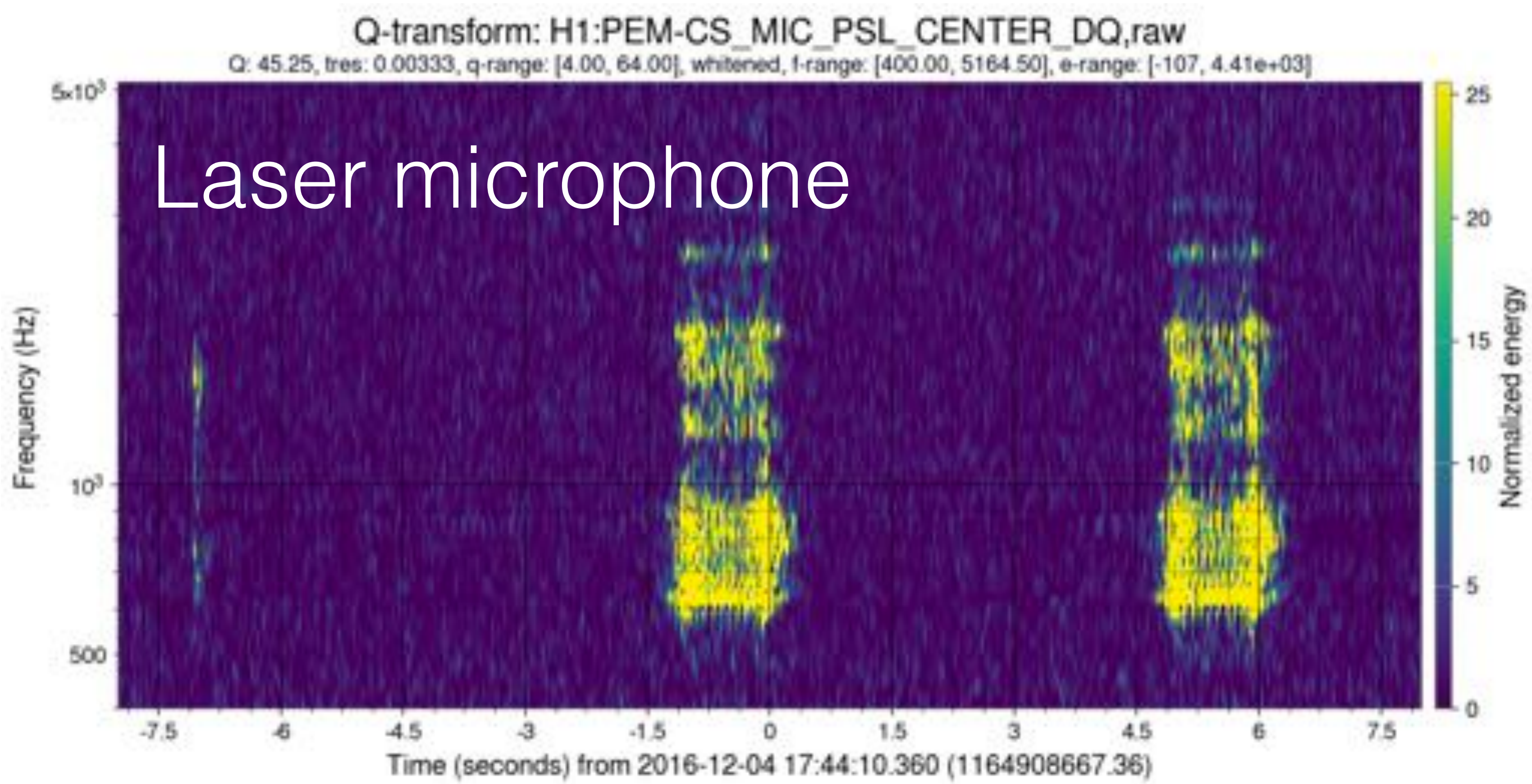
LIGO-Hanford h(t)



Sleuthing breakout problem: laser glitches!



Laser glitches - h(t) vs. microphones



LHO alog #32503: https://alog.ligo-wa.caltech.edu/aLOG/index.php?callRep=32503

Useful resources

GWpy examples: https://gwpy.github.io/docs/latest/examples/index.html

The Gravitational Wave Open Science Center (GWOSC): https://www.gw-openscience.org/

Tutorials: https://www.gw-openscience.org/tutorials/

Public interferometer status monitoring: https://www.gw-openscience.org/detector_status/

LIGO-Virgo alerts guide: https://www.gw-openscience.org/alerts/