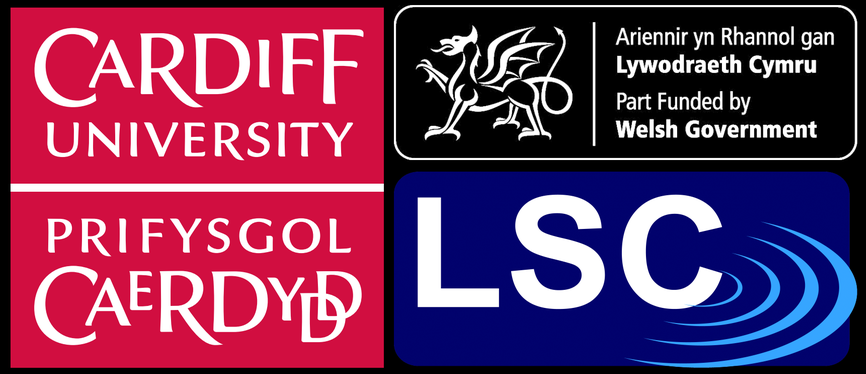


GWpy

Duncan Macleod
Sêr Cymru Research Fellow, Cardiff University

Why Python?

It’s free

It looks like pseudocode (most of the time)

It’s already available on most Unix machines

Widely used in astronomy and astrophysics already

It’s free

What is 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 LIGO/Virgo data (public and restricted), process, and visualise them

Nothing really specific to GW data other than data access routines

GWpy in the LSC-Virgo

Available to all users on the LIGO Data Grid machines

Widely used in detector characterisation

- Daily data summaries
- Detector/environment noise analysis
- Data-quality veto generation and testing

I/O / visualisation used in a few data analysis pipelines

LIGO-Virgo Summary Pages

GWpy used heavily to produce daily summary pages for internal use

- One-stop shop for information on detector status
- ~800 figures of merit for each LIGO interferometer every day
- Detector sensitivity, operating time, calibration
- Environmental sensor displays, transient noise analysis

LIGO-Virgo Summary Pages

LOSC hosting public Summary Pages

- Power Spectral Density for LIGO and Virgo
- Sensitive range for binary neutron star inspiral
- Operating segments
- Trends of ground motion, internal/external temperature
- https://losc.ligo.org/detector_status/

GWpy Quickstart

Install:

Import the class that represents the data you want to study

Fetch some open data from the OSC

Make a plot:

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

Import the class that represents the data you want to study

Fetch some open data from the OSC

Make a plot:

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

Import the class that represents the data you want to study

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

Fetch some open data from the OSC

Make a plot:

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

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:

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

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')
```

detector **start time of request** **end time of request**

Make a plot:

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

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')
```

detector **start time of request** **end time of request**

Make a plot:

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

Display the plot:

GWpy Quickstart

Install:

```
$ pip install gwpy
```

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')
```

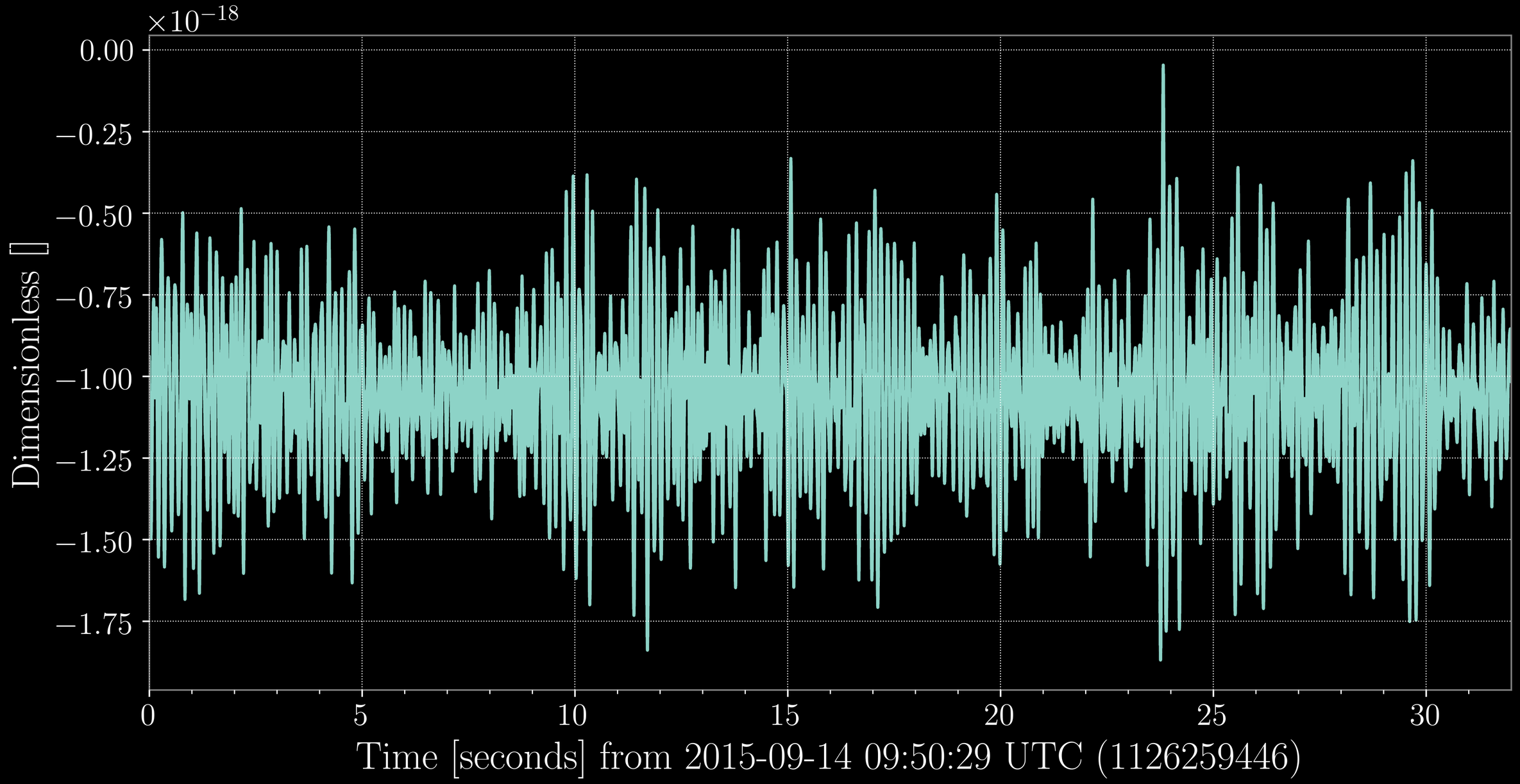
detector **start time of request** **end time of request**

Make a plot:

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

Display the plot:

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


$\times 10^{-18}$
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
Dimensionless []
0
5
10
15
20
25
30
Time [seconds] from 2015-09-14 09:50:29 UTC (1126259446)

Data access with GWpy

Read directly from files:

```
>>> data = TimeSeries.read('mydata.gwf')
```

Query for open data:

[LIGO/Virgo members can] query available data by ‘channel’ name

(provides access to all LIGO/Virgo data, including auxiliary instrumental and environment sensors)

Data access with GWpy

Read directly from files:

```
>>> data = TimeSeries.read('mydata.txt')
```

Query for open data:

[LIGO/Virgo members can] query available data by ‘channel’ name

(provides access to all LIGO/Virgo data, including auxiliary instrumental and environment sensors)

Data access with GWpy

Read directly from files:

```
>>> data = TimeSeries.read('mydata.h5')
```

Query for open data:

[LIGO/Virgo members can] query available data by ‘channel’ name

(provides access to all LIGO/Virgo data, including auxiliary instrumental and environment sensors)

Data access with GWpy

Read directly from files:

```
>>> data = TimeSeries.read('mydata.h5')
```

Query for open data:

```
>>> data = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478)
```

[LIGO/Virgo members can] query available data by ‘channel’ name

(provides access to all LIGO/Virgo data, including auxiliary instrumental and environment sensors)

Data access with GWpy

Read directly from files:

```
>>> data = TimeSeries.read('mydata.h5')
```

Query for open data:

```
>>> data = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478)
```

[LIGO/Virgo members can] query available data by ‘channel’ name

```
>>> data = TimeSeries.get('H1:GDS-CALIB_STRAIN', 1126259446, 1126259478)
```

(provides access to all LIGO/Virgo data, including auxiliary instrumental and environment sensors)

Signal processing with GWpy

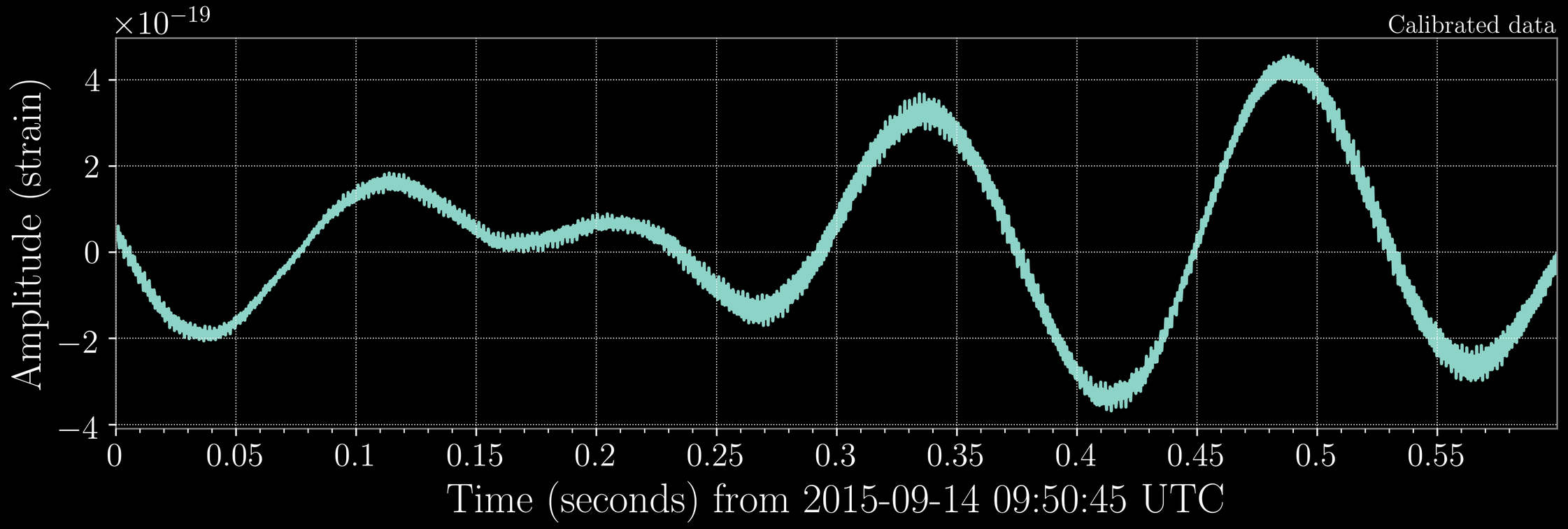
‘Raw’ gravitational-wave strain isn’t very useful for making pretty plots

GWpy provides simple signal-processing methods to filter data

- bandpass, lowpass, highpass
- notch
- arbitrary zero-pole-gain format definition

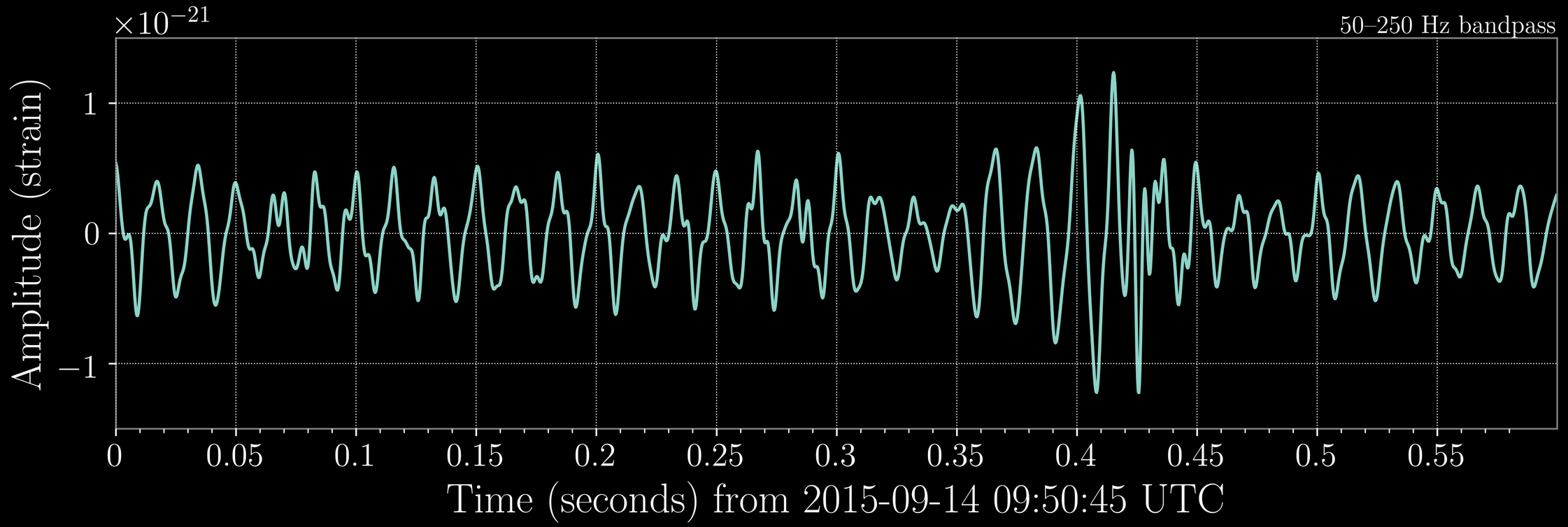
Signal processing with GWpy

```
>>> data = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478)
```



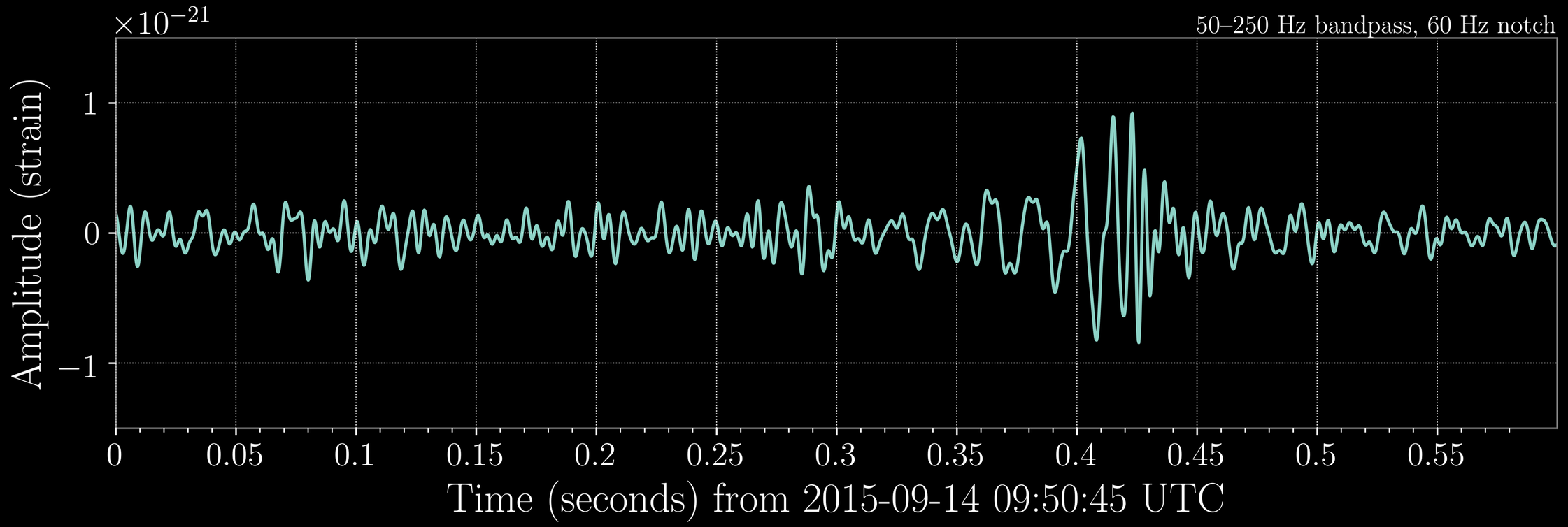
Signal processing with GWpy

```
>>> data = data.bandpass(50, 250)
```



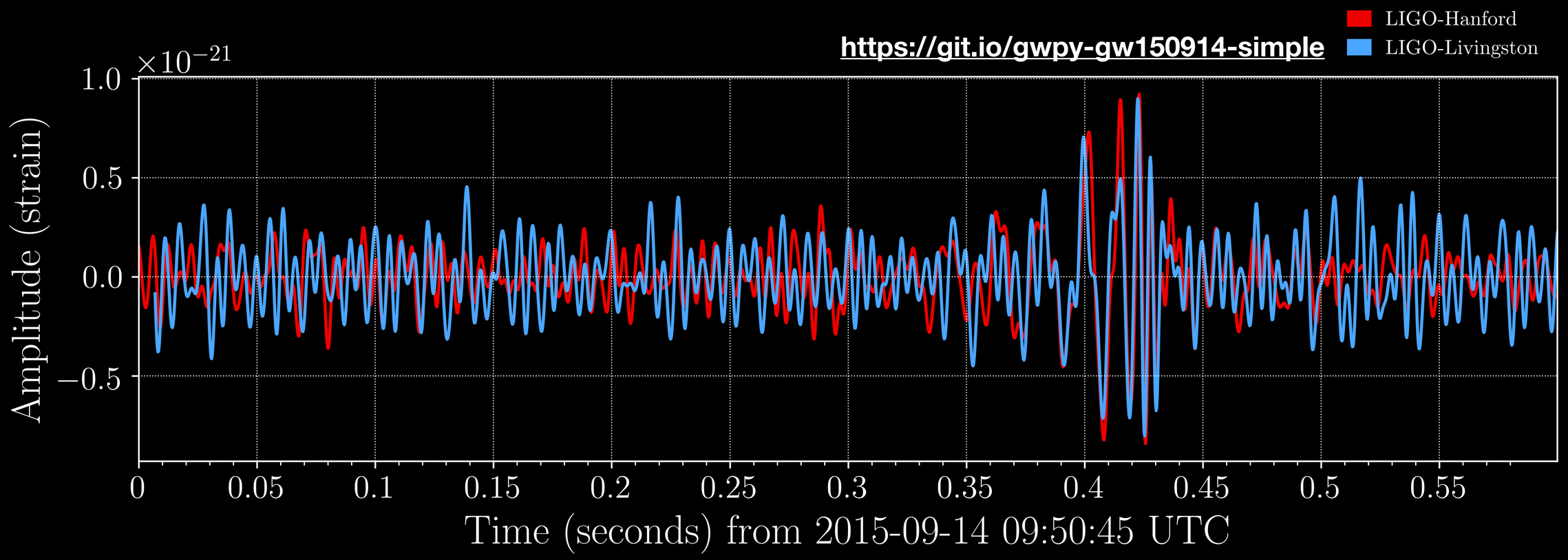
Signal processing with GWpy

```
>>> data = data.notch(60)
```



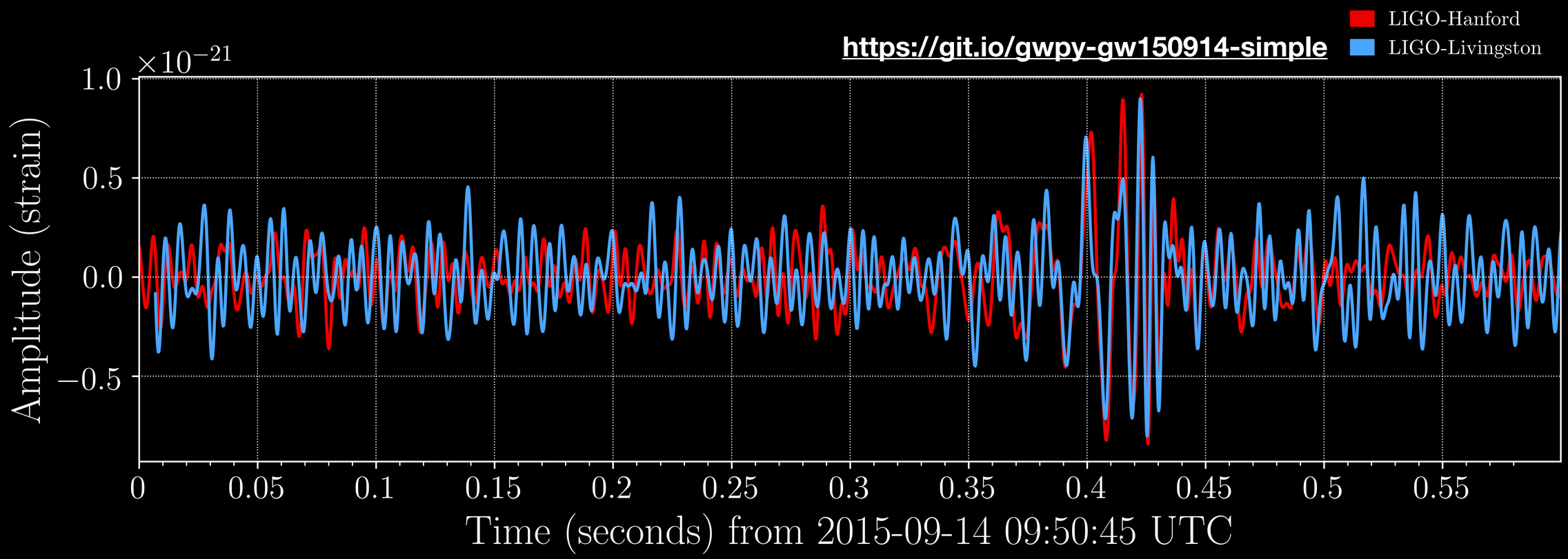
Signal processing with GWpy

A few extra lines (mainly figure formatting) give:



Signal processing with GWpy

A few extra lines (mainly figure formatting) give:

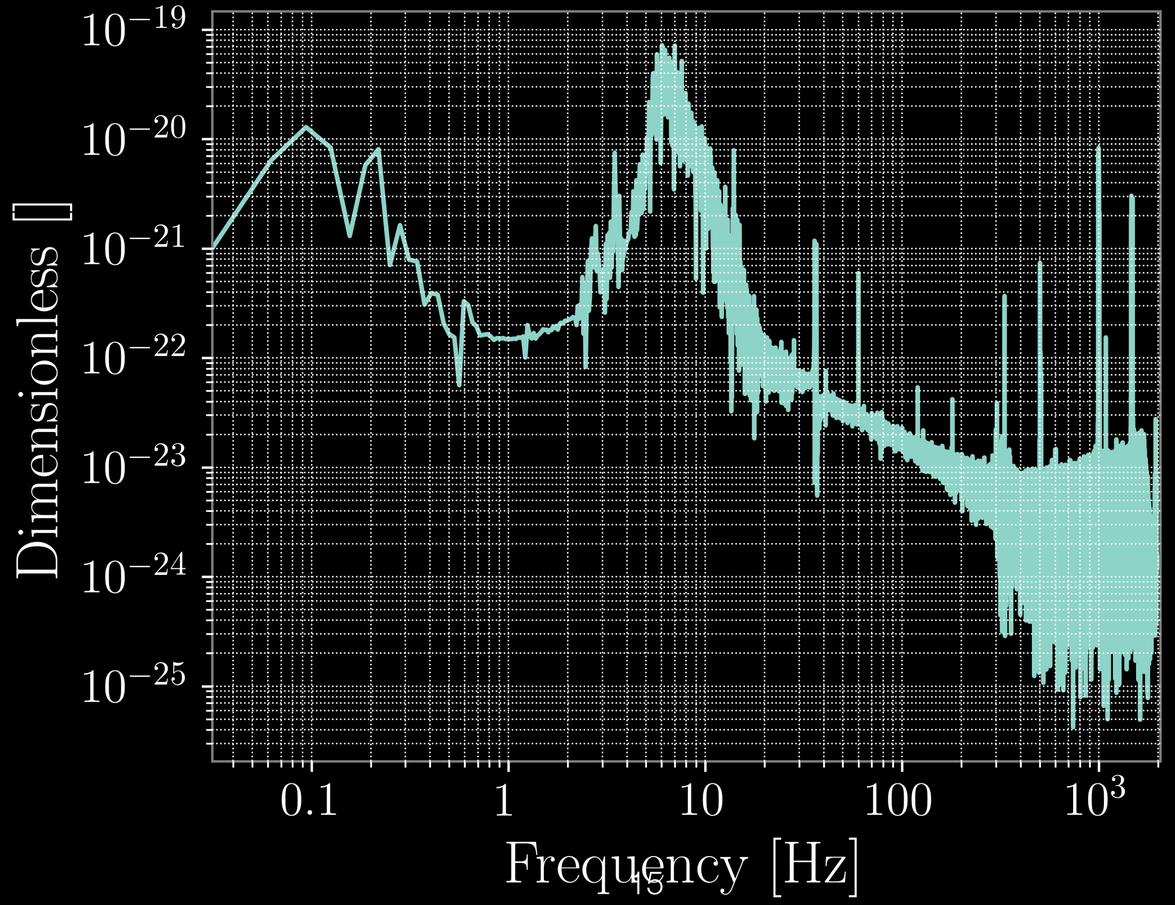


24 lines of functional code
I/O: 3 SP: 9 Plot: 12

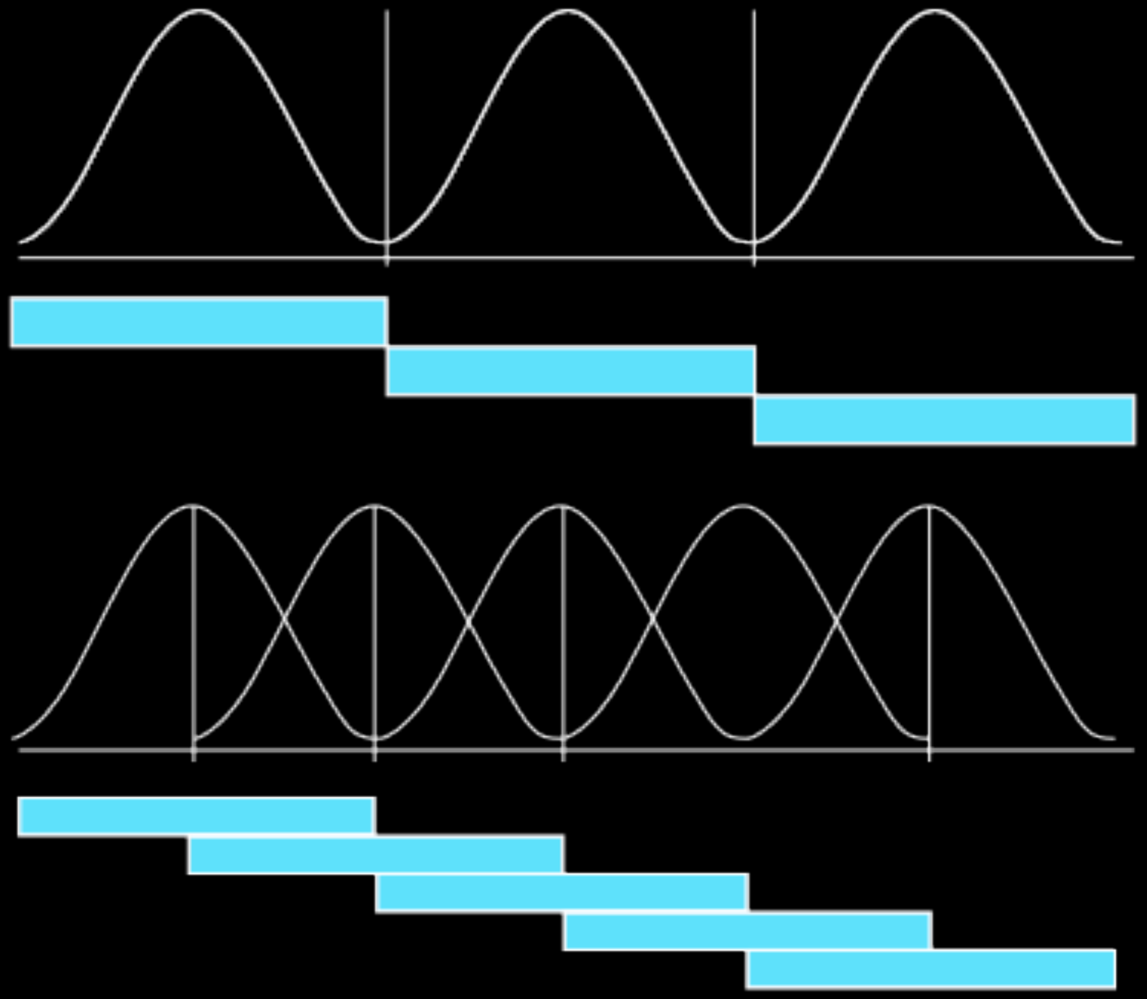
Signal processing with GWpy

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

```
>>> fft = data.fft()
```



Signal processing with GWpy



WINDOWING AND AVERAGING WITH
0% OVERLAP

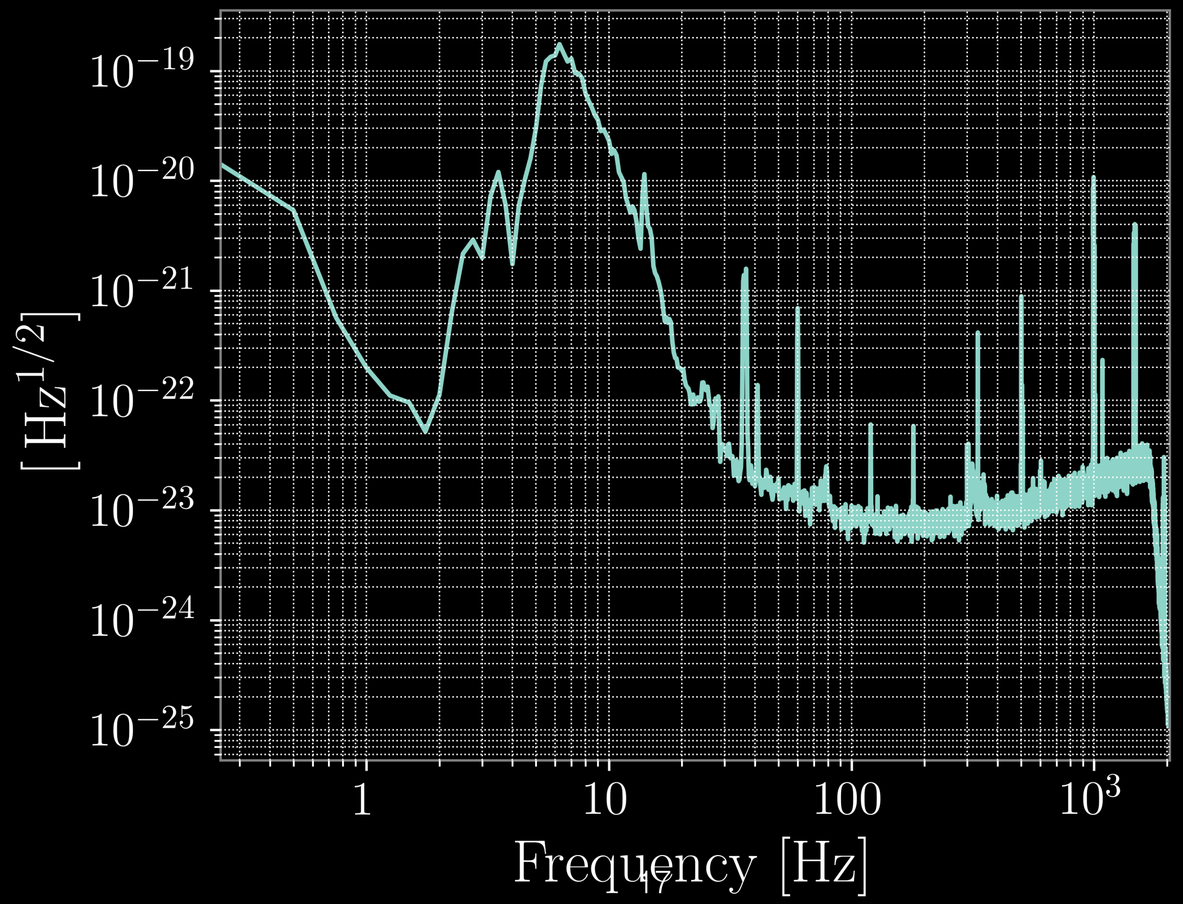
WINDOWING AND AVERAGING WITH
50% OVERLAP

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

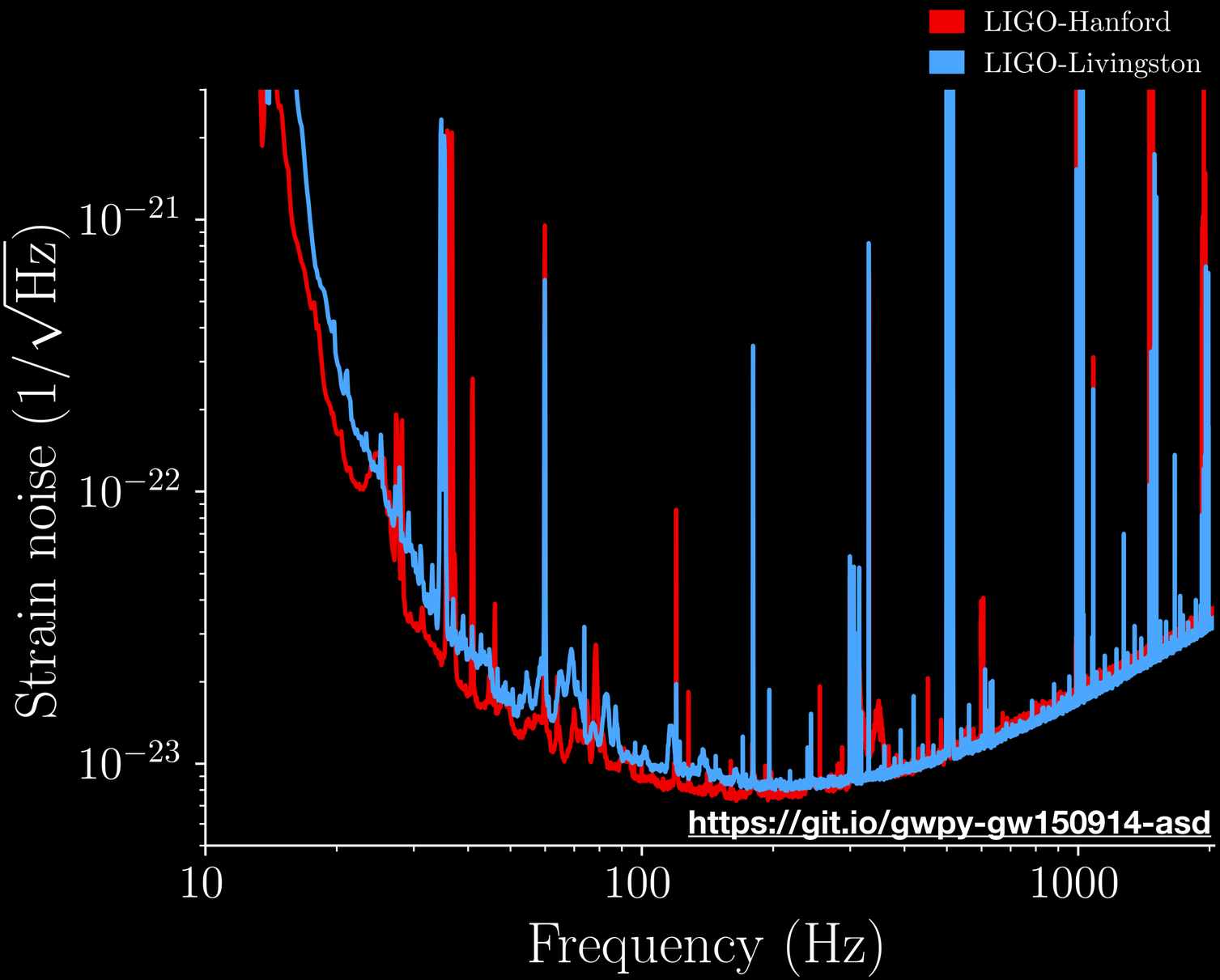
Signal processing with GWpy

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

```
>>> asd = data.asd(fftlength=4)
```



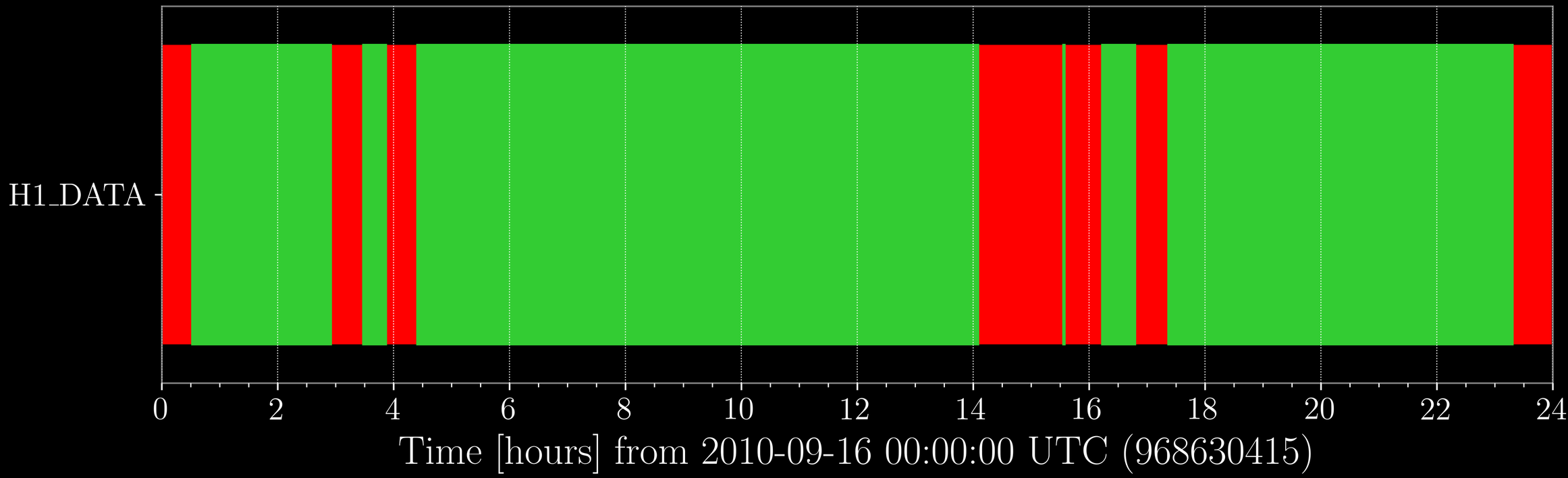
Signal processing with GWpy



Observing segments in GWpy

GWpy provides an interface to the LOSC Timeline segments:

```
>>> from gwpy.segments import DataQualityFlag
>>> segments = DataQualityFlag.fetch_open_data('H1_DATA', 'Sep 16 2010', 'Sep 17 2010')
>>> plot = segments.plot()
>>> plot.show()
```



Data tables in GWpy

GWpy builds upon Astropy's excellent `Table` object to provide simple routines to read, filter, and plot tabular data

Consider the following `catalogue.csv` file:

```
event,gps,mass1,mass2,distance
GW150914,1126259462,36,29,410
LVT151012,1128678900,23,12,1000
GW151226,1135136350,14,8,440
GW170104,1167559936,31,19,880
GW170608,1181786494,12,7,340
GW170814,1186741861,31,25,540
GW170817,1187008882,1.48,1.27,40
```

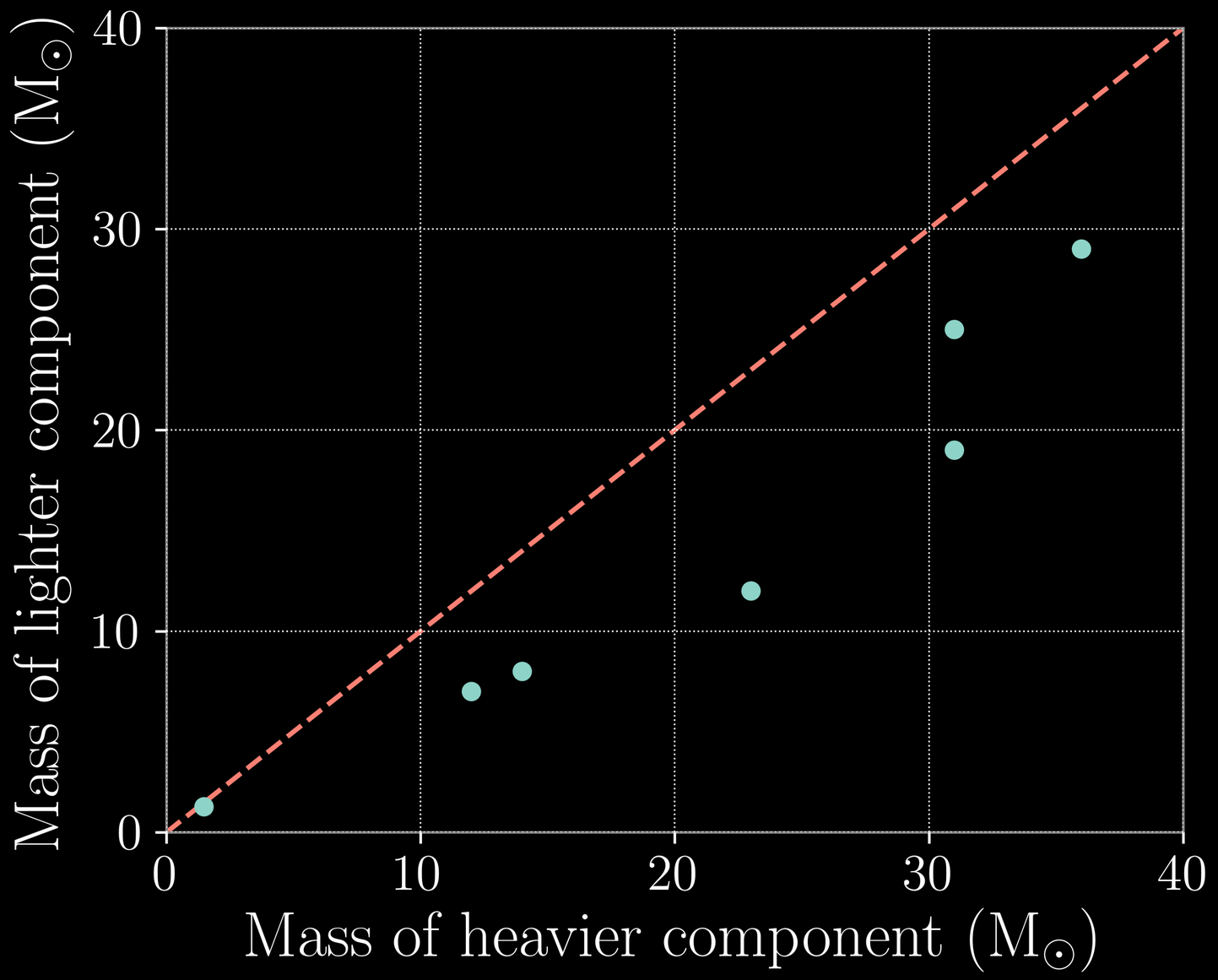
Data tables in GWpy

```
>>> from gwpy.table import EventTable

>>> table = EventTable.read('catalogue.csv', format='ascii')
>>> plot = table.plot('mass1', 'mass2')

>>> ax = plot.gca()
>>> ax.set_xlim(0, 40)
>>> ax.set_xlabel(r'Mass of heavier component (M$_\odot$)')
>>> ax.set_ylim(0, 40)
>>> ax.set_ylabel(r'Mass of lighter component (M$_\odot$)')
>>> ax.plot((0, 40), (0, 40), color='C3', linestyle='--', zorder=0)
>>> plot.show()
```

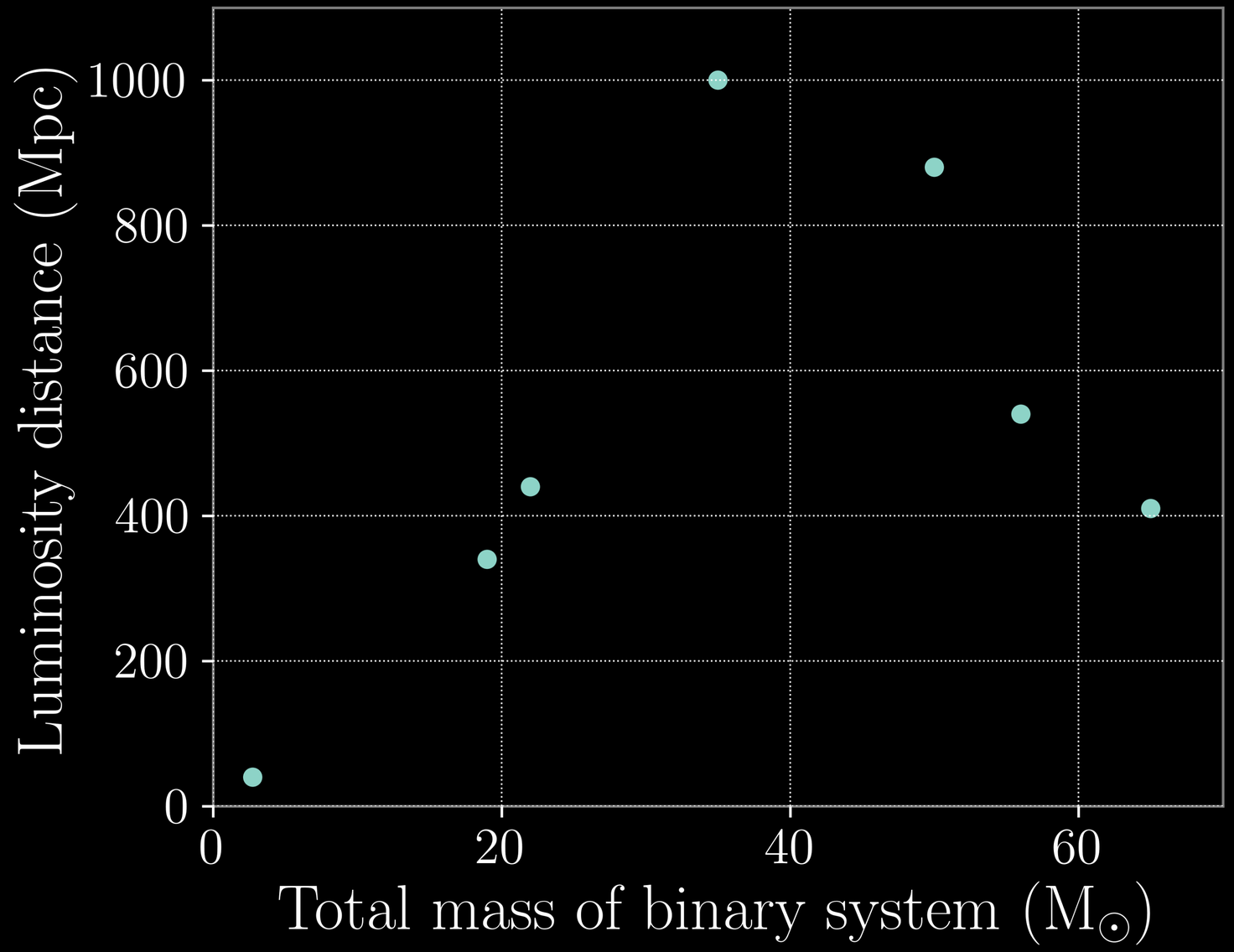
Data tables in GWpy



Data tables in GWpy

```
>>> table.add_column(table['mass1'] + table['mass2'], name='mtotal')
>>> plot = table.plot('mtotal', 'distance')
>>> ax = plot.gca()
>>> ax.set_xlim(0, 70)
>>> ax.set_ylim(0, 1100)
>>> ax.set_xlabel(r'Total mass of binary system (M$_\odot$)')
>>> ax.set_ylabel('Luminosity distance (Mpc)')
>>> plot.show()
```

Data tables in GWpy



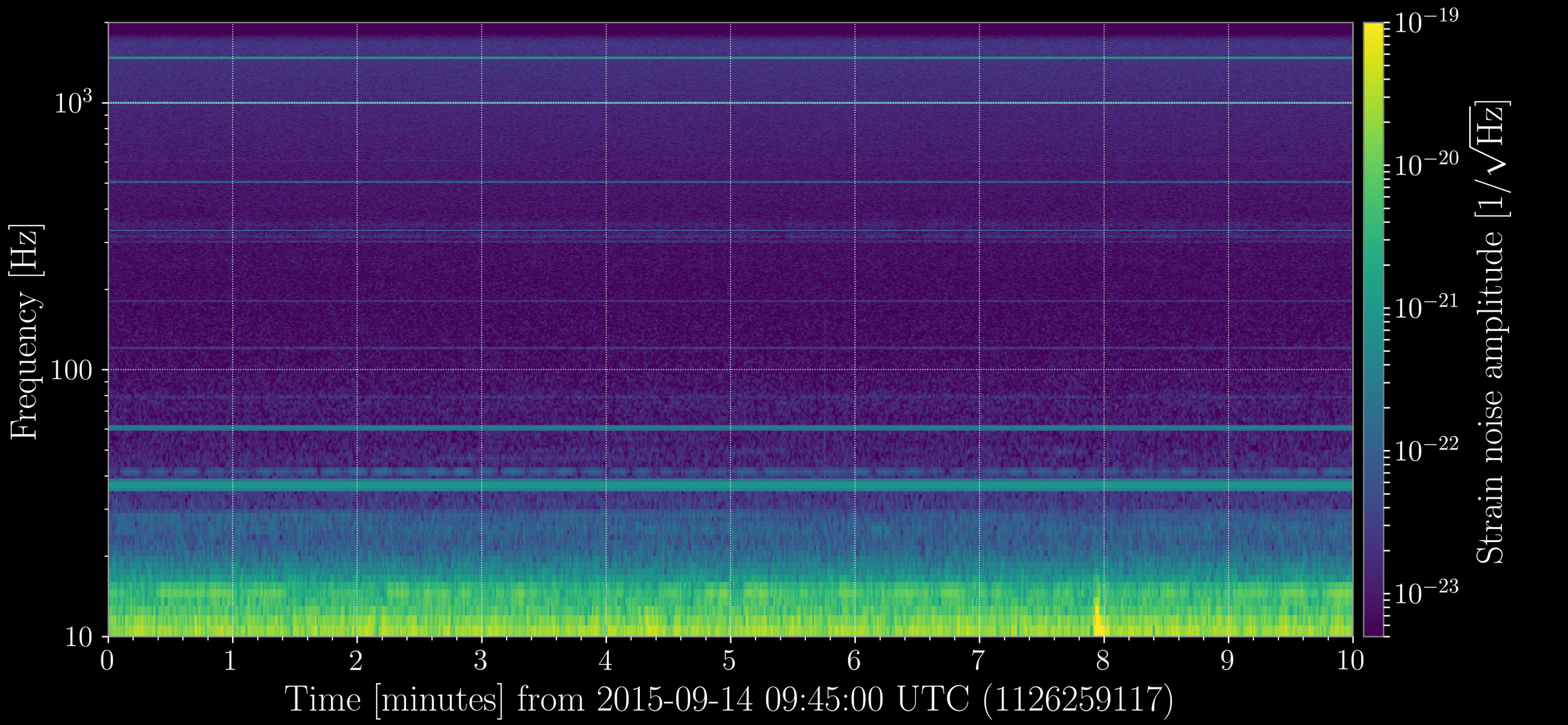
GWpy

Many examples available from

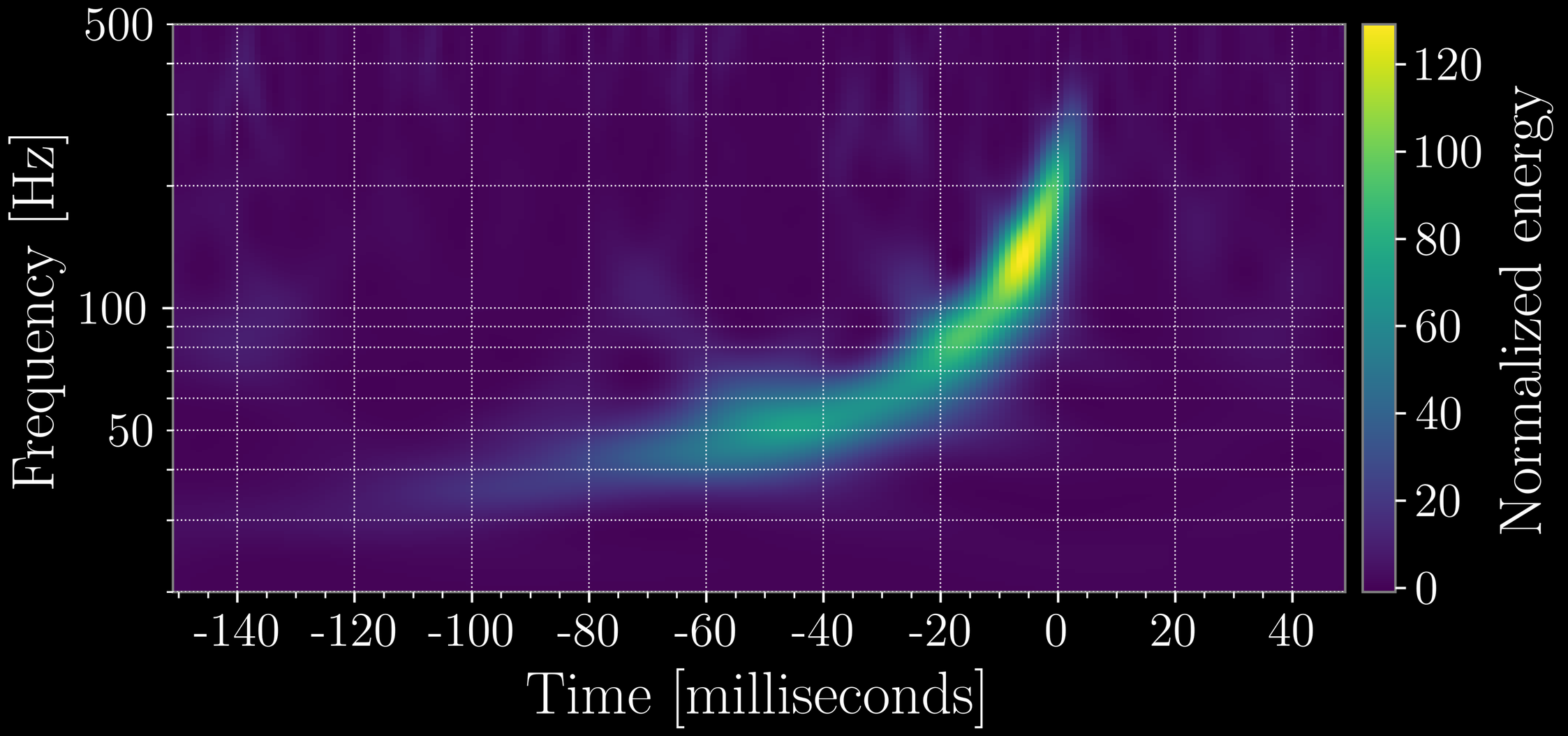
https://gwpy.github.io/docs/stables/examples/

- Spectrograms
- Q-transforms

GWpy


Frequency [Hz]
10^3
100
10
0
1
2
3
4
5
6
7
8
9
10
Time [minutes] from 2015-09-14 09:45:00 UTC (1126259117)
10^{-19}
10^{-20}
10^{-21}
10^{-22}
10^{-23}
Strain noise amplitude [$1/\sqrt{\text{Hz}}$]

GWpy


500
100
50
Frequency [Hz]
-140
-120
-100
-80
-60
-40
-20
0
20
40
Time [milliseconds]
120
100
80
60
40
20
0
Normalized energy

Contributing to GWpy

GWpy is an open-source library, hosted on GitHub, which means

- Anyone can see the code
- Anyone can copy the code and play around with it
- Anyone can report problems, post questions, or suggest new features
- Anyone can post bug fixes, improvements, or new additions

There's no such thing as a 'silly question' or a 'trivial contribution'

https://github.com/gwpy/gwpy/