Tuto 1.2 Open Data access with GWpy

November 17, 2020

1 Gravitational Wave Open Data Workshop #3

Tutorial 1.2: Introduction to GWpy This tutorial will briefly describe GWpy, a python package for gravitational astrophysics, and walk-through how you can use this to speed up access to, and processing of, GWOSC data.

Click this link to view this tutorial in Google Colaboratory

This notebook were generated using python 3.7, but should work on python 2.7, 3.6, or 3.7.

1.1 Installation (execute only if running on a cloud platform or if you haven't done the installation already!)

Note: we use `pip`, but **it is recommended** to use conda on your own machine, as explained in the installation instructions. This usage might look a little different than normal, simply because we want to do this directly from the notebook.

[2]:
```
# -- Uncomment following line if running in Google Colab
#! pip install -q 'gwpy==1.0.1'
```

```
ERROR: pesummary 0.2.4 requires configparser, which is not installed.
ERROR: hveto 1.0.1 has requirement gwdetchar>=1.0.0, but you'll have

gwdetchar 0+unknown which is incompatible.
ERROR: gwsumm 1.0.1 has requirement gwdetchar>=1.0.0, but you'll have

gwdetchar 0+unknown which is incompatible.
```

Important: With Google Colab, you may need to restart the runtime after running the cell above.

1.2 Initialization

[1]:
```
import gwpy
print(gwpy.__version__)
```

```
2.0.1
```

1.3 A note on object-oriented programming

Before we dive too deeply, its worth a quick aside on object-oriented programming (OOP). GWpy is heavily object-oriented, meaning almost all of the code you run using GWpy is based around an object of some type, e.g. `TimeSeries`. Most of the methods (functions) we will use are attached to an object, rather than standing alone, meaning you should have a pretty good idea of what sort of data you are dealing with (without having to read the documentation!).

For a quick overview of object-oriented programming in Python, see this blog post by Jeff Knupp.

1.4 Handling data in the time domain

Finding open data We have seen already that the `gwosc` module can be used to query for what data are available on GWOSC. The next thing to do is to actually read some open data. Let's try to get some for GW150914, the first direct detection of an astrophysical gravitational-wave signal from a BBH (binary black hole system).

We can use the `TimeSeries.fetch_open_data` method to download data directly from https://www.gw-openscience.org, but we need to know the GPS times. We can query for the GPS time of an event as follows:

[2]:
```
from gwosc.datasets import event_gps
gps = event_gps('GW150914')
print(gps)
```

```
1126259462.4
```

Now we can build a `[start, end)` GPS segment to 10 seconds around this time, using integers for convenience:

[3]:
```
segment = (int(gps)-5, int(gps)+5)
print(segment)
```

```
(1126259457, 1126259467)
```

and can now query for the full data. For this example we choose to retrieve data for the LIGO-Livingston interferometer, using the identifier `'L1'`. We could have chosen any of

- `'G1`' - GEO600
- `'H1'` - LIGO-Hanford
- `'L1'` - LIGO-Livingston
- `'V1'` - (Advanced) Virgo

In the future, the Japanese observatory KAGRA will come online, with the identifier `'K1'`.

[4]:
```
from gwpy.timeseries import TimeSeries
ldata = TimeSeries.fetch_open_data('L1', *segment, verbose=True)
print(ldata)
```

```
Fetched 1 URLs from www.gw-openscience.org for [1126259457 .. 1126259467))
Reading data… [Done]
```

```
TimeSeries([-9.31087178e-19, -9.75697062e-19, -1.01074940e-18,
             …, -1.13460681e-18, -1.11414494e-18,
             -1.15931435e-18]
            unit: dimensionless,
            t0: 1126259457.0 s,
            dt: 0.000244140625 s,
            name: Strain,
            channel: None)
```

The `verbose=True` flag lets us see that GWpy has discovered two files that provides the data for the given interval, downloaded them, and loaded the data. The files are not stored permanently, so next time you do the same call, it will be downloaded again, however, if you know you might repeat the same call many times, you can use `cache=True` to store the file on your computer.

Notes:

- To read data from a local file instead of from the GWOSC server, we can use `TimeSeries.read` method.

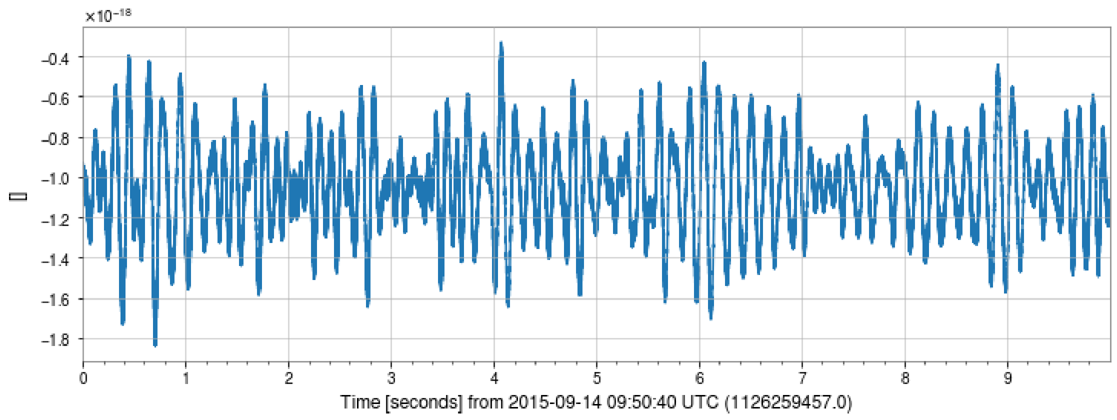
We have now downloaded real LIGO data for GW150914! These are the actual data used in the analysis that discovered the first binary black hole merger.

To sanity check things, we can easily make a plot, using the `plot()` method of the `data` `TimeSeries`.

Since this is the first time we are plotting something in this notebook, we need to make configure `matplotlib` (the plotting library) to work within the notebook properly:

Matplotlib documentation can be found here.

[5]:
```
%matplotlib inline
plot = ldata.plot()
```



Notes: There are alternatives ways to access the GWOSC data.

- `readligo` is a light-weight Python module that returns the time series into a Numpy array.

- The PyCBC package has the `pycbc.frame.query_and_read_frame` and `pycbc.frame.read_frame` methods. We use PyCBC in Tutorial 2.1, 2.2 and 2.3.

1.5 Handling data in the frequency domain using the Fourier transform

The Fourier transform is a widely-used mathematical tool to expose the frequency-domain content of a time-domain signal, meaning we can see which frequencies contian lots of power, and which have less.

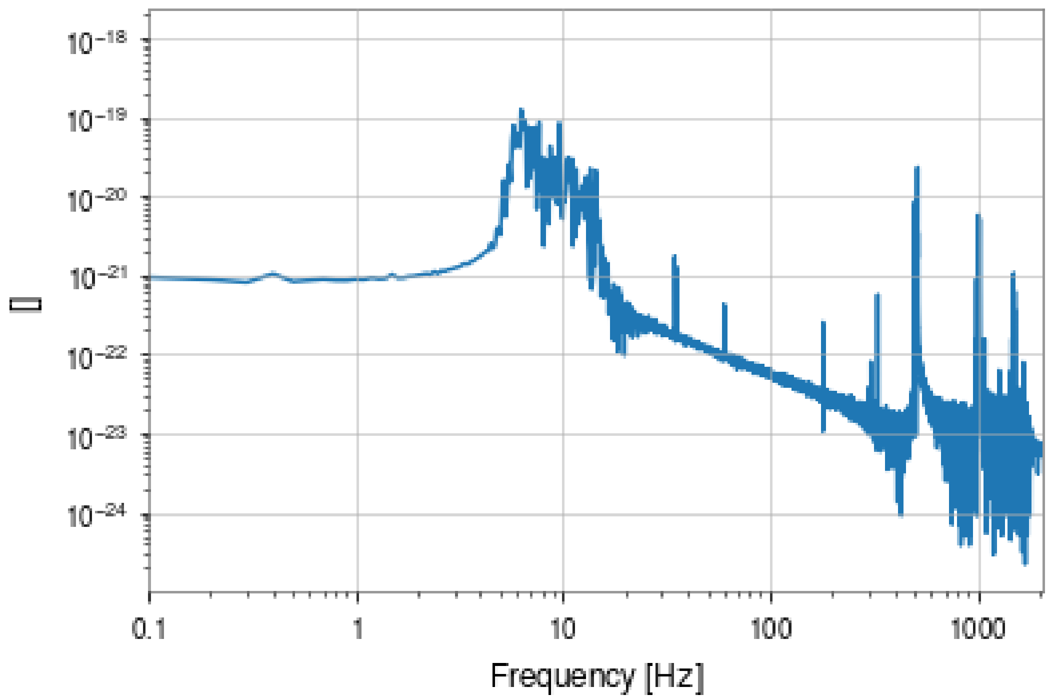
We can calculate the Fourier transform of our `TimeSeries` using the `fft()` method:

[6]:
```
fft = ldata.fft()
print(fft)
```

```
FrequencySeries([-1.05219333e-18+0.00000000e+00j,
                 -9.36314427e-22+6.61514979e-23j,
                 -8.74693847e-22-3.26717194e-23j, …,
                  6.63325463e-24-1.10870240e-26j,
                  6.73663868e-24-7.82178218e-26j,
                  6.69598891e-24+0.00000000e+00j]
                unit: dimensionless,
                f0: 0.0 Hz,
                df: 0.1 Hz,
                epoch: 1126259457.0,
                name: Strain,
                channel: None)
```

The result is a `FrequencySeries`, with complex amplitude, representing the amplitude and phase of each frequency in our data. We can use `abs()` to extract the amplitude and plot that:

[7]:
```
plot = fft.abs().plot(xscale="log", yscale="log")
plot.show(warn=False)
```

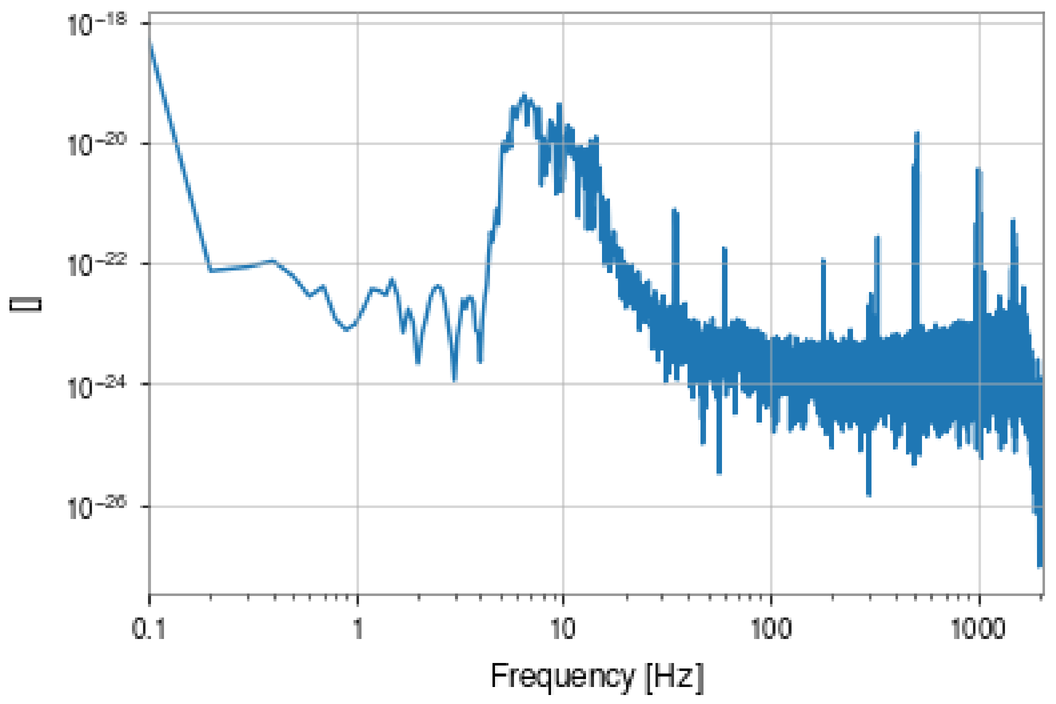


This doesn't look correct at all! The problem is that the FFT works under the assumption that our data are periodic, which means that the edges of our data look like discontinuities when transformed. We need to apply a window function to our time-domain data before transforming, which we can do using the `scipy.signal` module:

[8]:
```
from scipy.signal import get_window
window = get_window('hann', ldata.size)
lwin = ldata * window
```

Let's try our transform again and see what we get

[9]:
```
fftamp = lwin.fft().abs()
plot = fftamp.plot(xscale="log", yscale="log")
plot.show(warn=False)
```

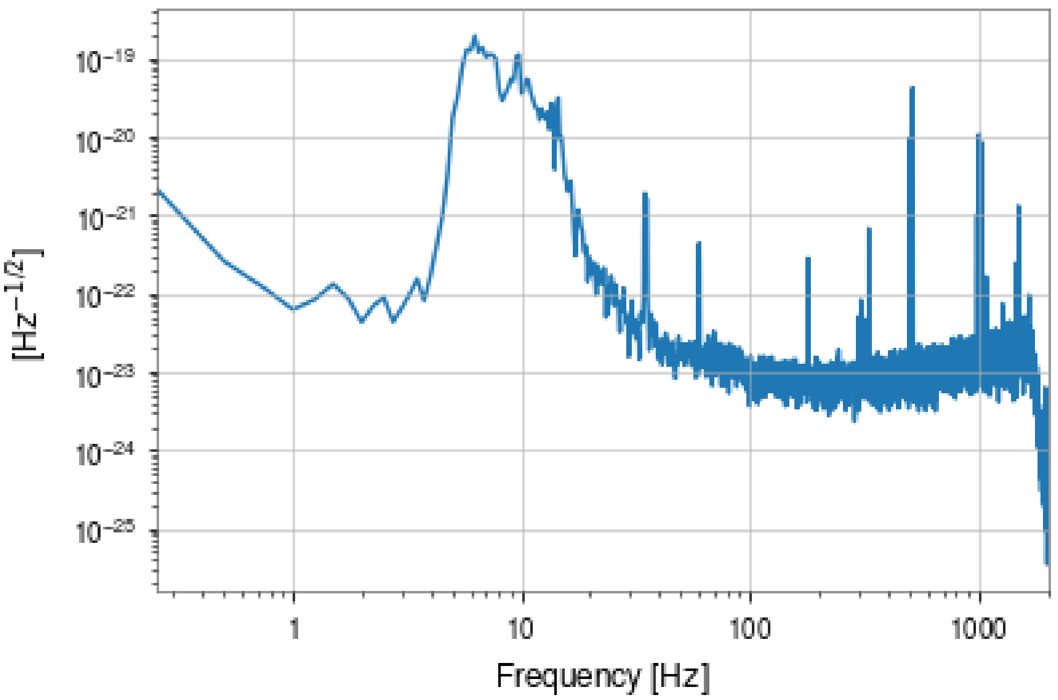


This looks a little more like what we expect for the amplitude spectral density of a gravitational-wave detector.

1.6 Calculating the power spectral density

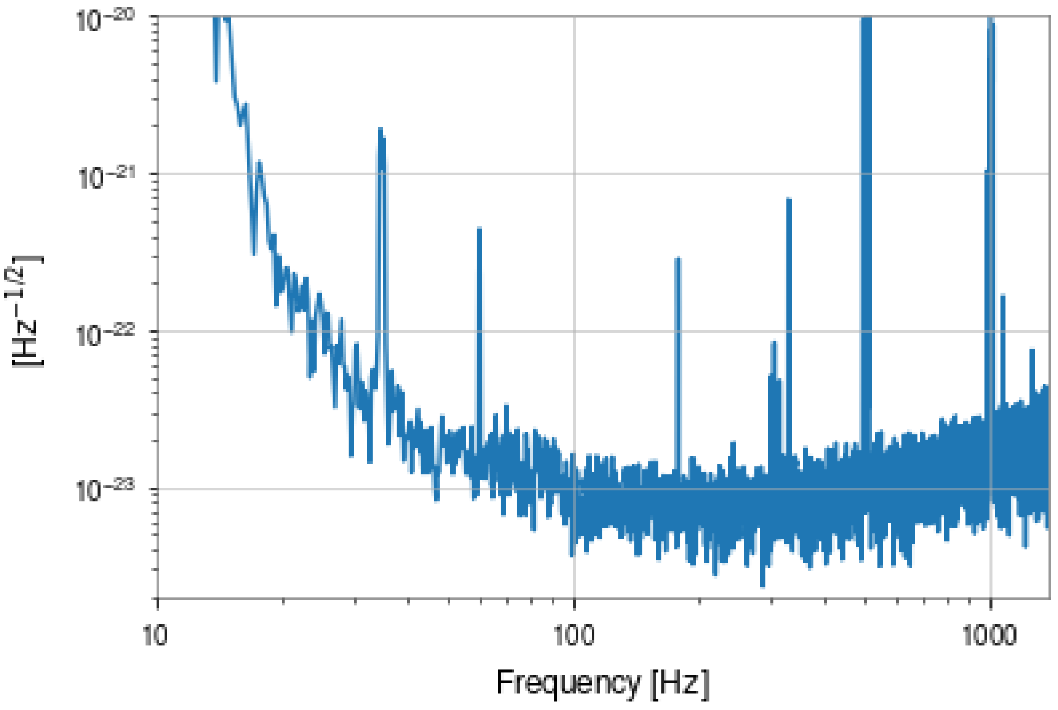
In practice, we typically use a large number of FFTs to estimate an averages power spectral density over a long period of data. We can do this using the `asd()` method, which uses Welch's method to combine FFTs of overlapping, windowed chunks of data.

[10]:
```
asd = ldata.asd(fftlength=4, method="median")
plot = asd.plot()
plot.show(warn=False)
```



```
[11]: ax = plot.gca()
      ax.set_xlim(10, 1400)
      ax.set_ylim(2e-24, 1e-20)
      plot
```

```
[11]:
```



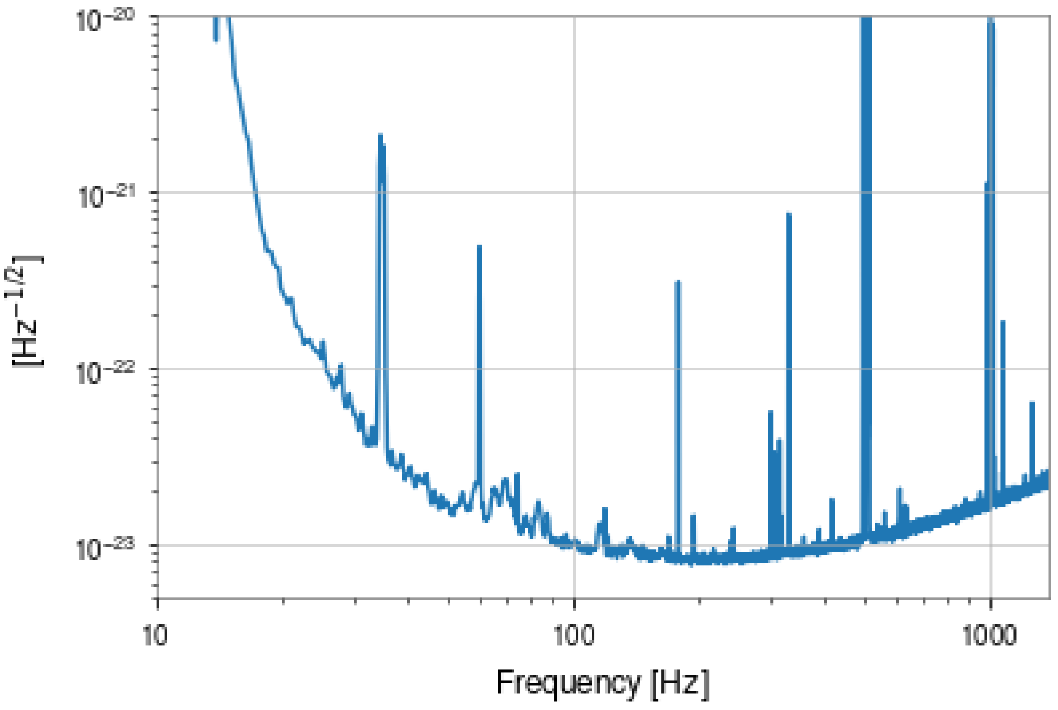
The ASD is a standard tool used to study the frequency-domain sensitivity of a gravitational-wave detector. For the LIGO-Livingston data we loaded, we can see large spikes at certain frequencies, including

- ~300 Hz
- ~500 Hz
- ~1000 Hz

The O2 spectral lines page on GWOSC describes a number of these spectral features for O2, with some of them being forced upon us, and some being deliberately introduced to help with interferometer control.

Loading more data allows for more FFTs to be averaged during the ASD calculation, meaning random variations get averaged out, and we can see more detail:

[12]:
```
ldata2 = TimeSeries.fetch_open_data('L1', int(gps)-512, int(gps)+512,␣
 ↪cache=True)
lasd2 = ldata2.asd(fftlength=4, method="median")
plot = lasd2.plot()
ax = plot.gca()
ax.set_xlim(10, 1400)
ax.set_ylim(5e-24, 1e-20)
plot.show(warn=False)
```

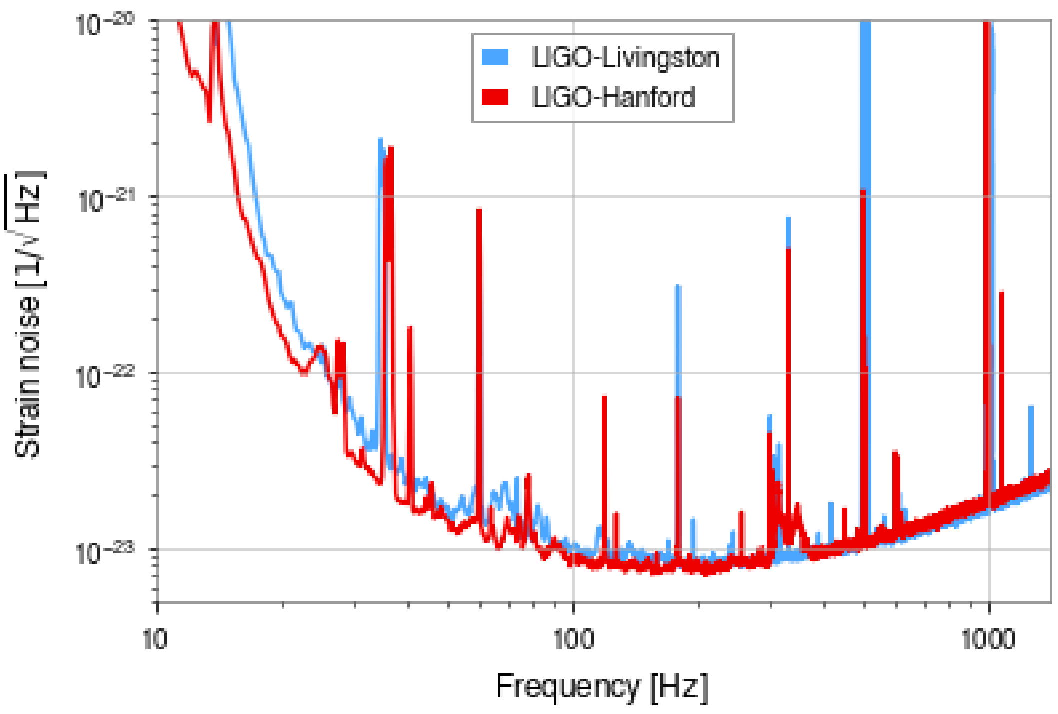


Now we can see some more features, including sets of lines around ~30 Hz and ~65 Hz, and some more isolate lines through the more sensitive region.

For comparison, we can load the LIGO-Hanford data and plot that as well:

[13]:
```
# get Hanford data
hdata2 = TimeSeries.fetch_open_data('H1', int(gps)-512, int(gps)+512,␣
 ↪cache=True)
hasd2 = hdata2.asd(fftlength=4, method="median")

# and plot using standard colours
ax.plot(hasd2, label='LIGO-Hanford', color='gwpy:ligo-hanford')

# update the Livingston line to use standard colour, and have a label
lline = ax.lines[0]
lline.set_color('gwpy:ligo-livingston')  # change colour of Livingston data
lline.set_label('LIGO-Livingston')

ax.set_ylabel(r'Strain noise [$1/\sqrt{\mathrm{Hz}}$]')
ax.legend()
plot
```

[13]:



Now we can see clearly the relative sensitivity of each LIGO instrument, the common features between both, and those unique to each observatory.

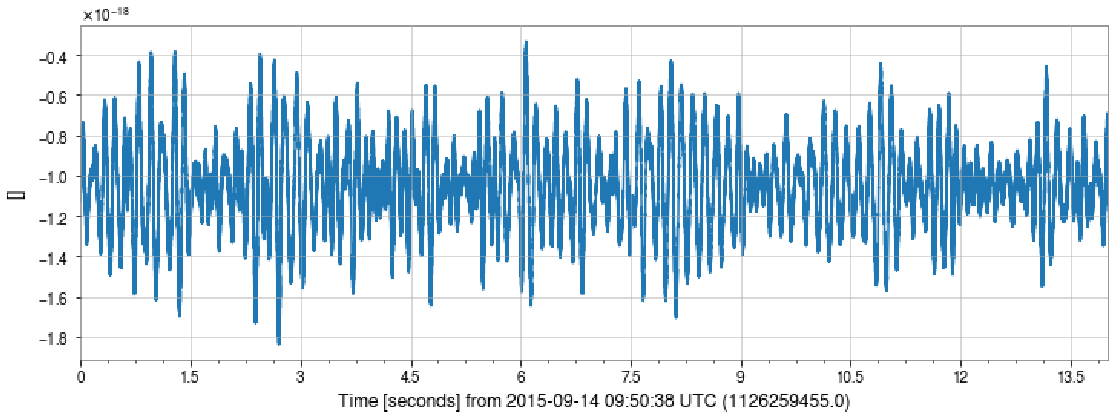
2 Challenges:

Quiz Question 1: The peak amplitude in the LIGO-Livingston data occurs at approximately 5 seconds into the plot above and is undetectable above the background noise by the eye. Plot the data for the LIGO-Hanford detector around GW150914. Looking at your new LIGO-Handford plot, can your eye identify a signal peak?

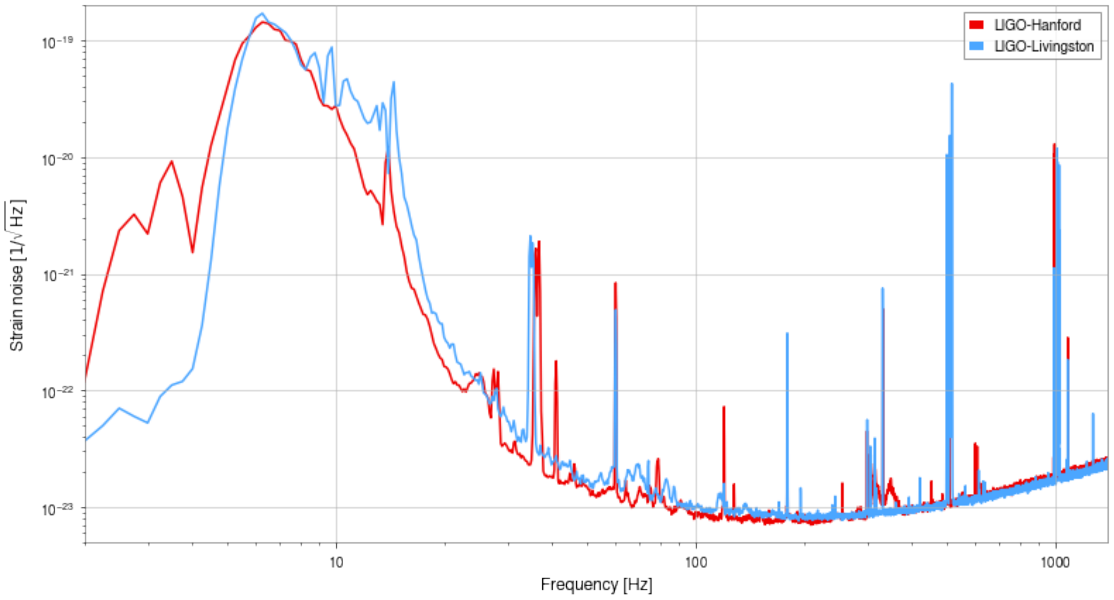
[14]:
```
from gwosc.datasets import event_gps
gps1 = event_gps('GW150914')
print(gps1)
segment1 = (int(gps)-7, int(gps)+7)
print(segment1)

from gwpy.timeseries import TimeSeries
ldata1 = TimeSeries.fetch_open_data('L1', *segment1, verbose=True)
print(ldata1)

%matplotlib inline
```

```
plot1 = ldata1.plot()
```

```
1126259462.4
(1126259455, 1126259469)
Fetched 1 URLs from www.gw-openscience.org for [1126259455 .. 1126259469))
Reading data… [Done]
TimeSeries([-1.12341911e-18, -1.12524861e-18, -1.12212345e-18,
            …, -6.90028651e-19, -7.17681003e-19,
            -7.36341751e-19]
           unit: dimensionless,
           t0: 1126259455.0 s,
           dt: 0.000244140625 s,
           name: Strain,
           channel: None)
```



```
[24]: import matplotlib.pyplot as plt

      ldata2 = TimeSeries.fetch_open_data('L1', int(gps)-512, int(gps)+512,␣
       ↪cache=True)
      lasd2 = ldata2.asd(fftlength=4, method="median")

      # get Hanford data
      hdata2 = TimeSeries.fetch_open_data('H1', int(gps)-512, int(gps)+512,␣
       ↪cache=True)
      hasd2 = hdata2.asd(fftlength=4, method="median")

      fig, ax = plt.subplots(figsize=(13,7))
      # and plot using standard colours
      ax.plot(hasd2, label='LIGO-Hanford', color='gwpy:ligo-hanford')
      ax.plot(lasd2, label='LIGO-Livingston', color='gwpy:ligo-livingston')

      ax.set_xlim(2, 1400)
      ax.set_ylim(5e-24, 2e-19)
      # update the Livingston line to use standard colour, and have a label

      ax.set_yscale('log')
      ax.set_xscale('log')
      ax.set_ylabel(r'Strain noise [$1/\sqrt{\mathrm{Hz}}$]')
      ax.set_xlabel(r'Frequency [Hz]')

      ax.legend()
      plt.show()
```



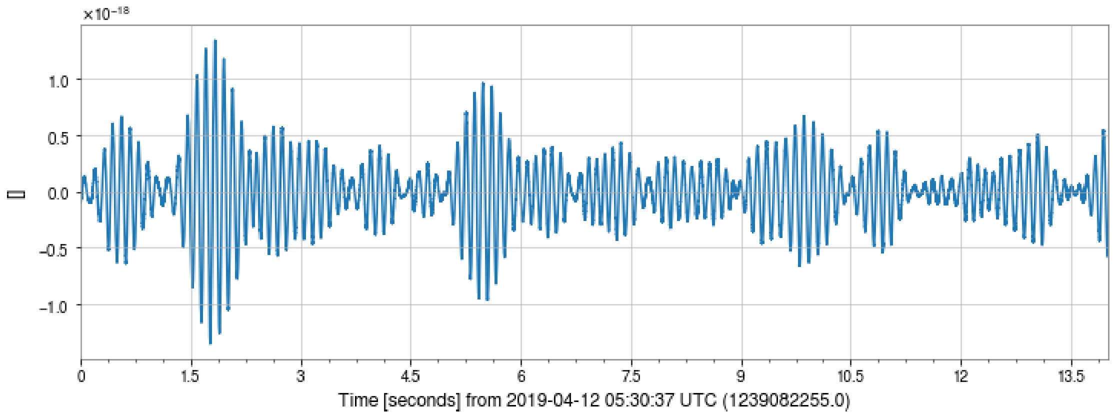
3 Quiz Question 2 :

Make an ASD around the time of an O3 event, GW190412 for L1 detector . Compare this with the ASDs around GW150914 for L1 detector. Which data have lower noise - and so are more sensitive - around 100 Hz?

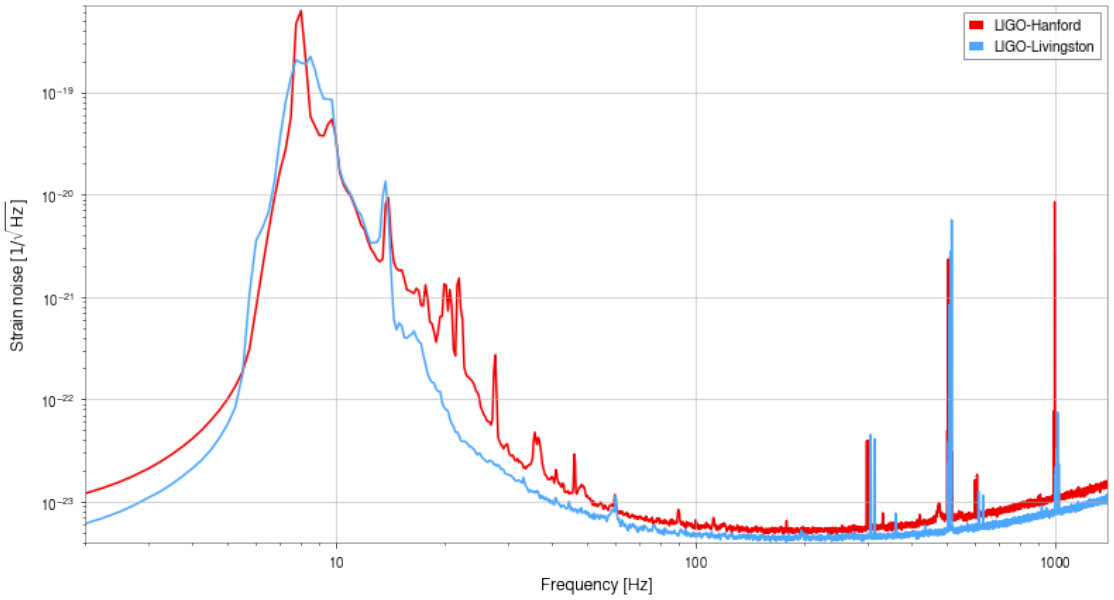
```
[25]: from gwosc.datasets import event_gps
      gps_GW190412 = event_gps('GW190412')
      print(gps_GW190412)
      segment_GW190412 = (int(gps_GW190412)-7, int(gps_GW190412)+7)
      print(segment_GW190412)

      from gwpy.timeseries import TimeSeries
      ldata_GW190412 = TimeSeries.fetch_open_data('L1', *segment_GW190412,␣
       ↪verbose=True)
```

```
print(ldata_GW190412)

%matplotlib inline
plot_GW190412 = ldata_GW190412.plot()
```

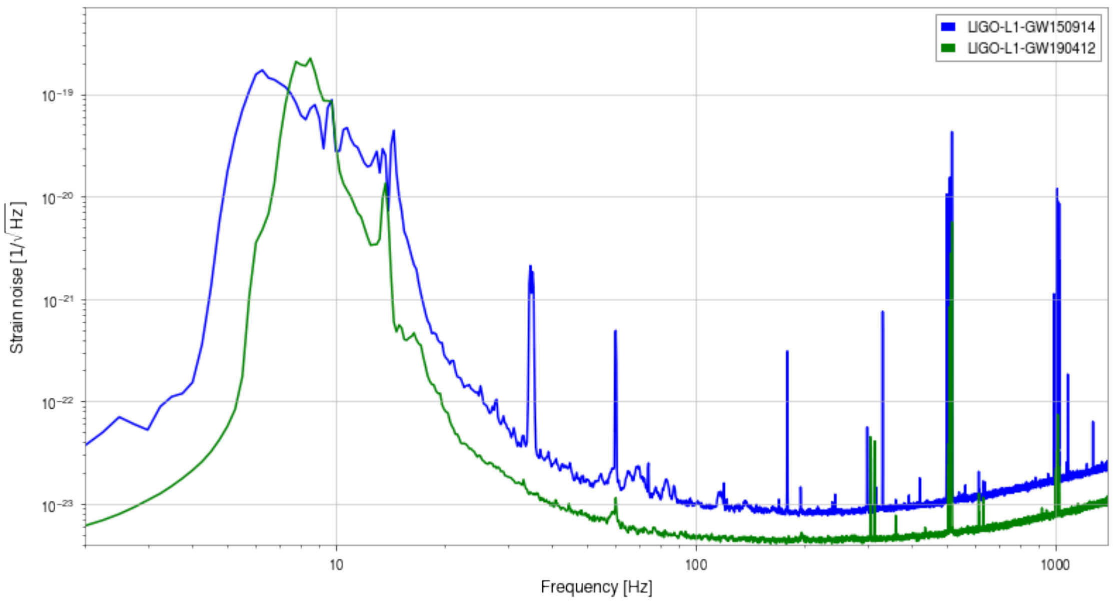
```
1239082262.2
(1239082255, 1239082269)
Fetched 1 URLs from www.gw-openscience.org for [1239082255 .. 1239082269))
Reading data… [Done]
TimeSeries([-9.53073391e-20, -9.87124228e-20, -1.01169057e-19,
            …, -5.63350989e-19, -5.69504640e-19,
            -5.74637581e-19]
           unit: dimensionless,
           t0: 1239082255.0 s,
           dt: 0.000244140625 s,
           name: Strain,
           channel: None)
```



[32]:
```
import matplotlib.pyplot as plt

ldata_GW190412 = TimeSeries.fetch_open_data('L1', int(gps_GW190412)-512,␣
 ↪int(gps_GW190412)+512, cache=True)
lasd_GW190412 = ldata_GW190412.asd(fftlength=4, method="median")

# get Hanford data
hdata_GW190412 = TimeSeries.fetch_open_data('H1', int(gps_GW190412)-512,␣
 ↪int(gps_GW190412)+512, cache=True)
hasd_GW190412 = hdata_GW190412.asd(fftlength=4, method="median")

fig, ax = plt.subplots(figsize=(13,7))
# and plot using standard colours
ax.plot(hasd_GW190412, label='LIGO-Hanford', color='gwpy:ligo-hanford')
ax.plot(lasd_GW190412, label='LIGO-Livingston', color='gwpy:ligo-livingston')

ax.set_xlim(2, 1400)
ax.set_ylim(4e-24, 7e-19)
# update the Livingston line to use standard colour, and have a label

ax.set_yscale('log')
ax.set_xscale('log')
ax.set_ylabel(r'Strain noise [$1/\sqrt{\mathrm{Hz}}$]')
ax.set_xlabel(r'Frequency [Hz]')

ax.legend()
plt.show()
```



[33]:
```
import matplotlib.pyplot as plt

ldata_GW190412 = TimeSeries.fetch_open_data('L1', int(gps_GW190412)-512,␣
 ↪int(gps_GW190412)+512, cache=True)
lasd_GW190412 = ldata_GW190412.asd(fftlength=4, method="median")

ldata2 = TimeSeries.fetch_open_data('L1', int(gps)-512, int(gps)+512,␣
 ↪cache=True)
lasd2 = ldata2.asd(fftlength=4, method="median")

fig, ax = plt.subplots(figsize=(13,7))
# and plot using standard colours
ax.plot(lasd2, label='LIGO-L1-GW150914', color='blue')
ax.plot(lasd_GW190412, label='LIGO-L1-GW190412', color='green')

ax.set_xlim(2, 1400)
ax.set_ylim(4e-24, 7e-19)
# update the Livingston line to use standard colour, and have a label

ax.set_yscale('log')
ax.set_xscale('log')
ax.set_ylabel(r'Strain noise [$1/\sqrt{\mathrm{Hz}}$]')
ax.set_xlabel(r'Frequency [Hz]')

ax.legend()
plt.show()
```



4 The GW190412 data has lesser noise around 100 Hz

```
[ ]:
```