```
! pip install gwpy
```

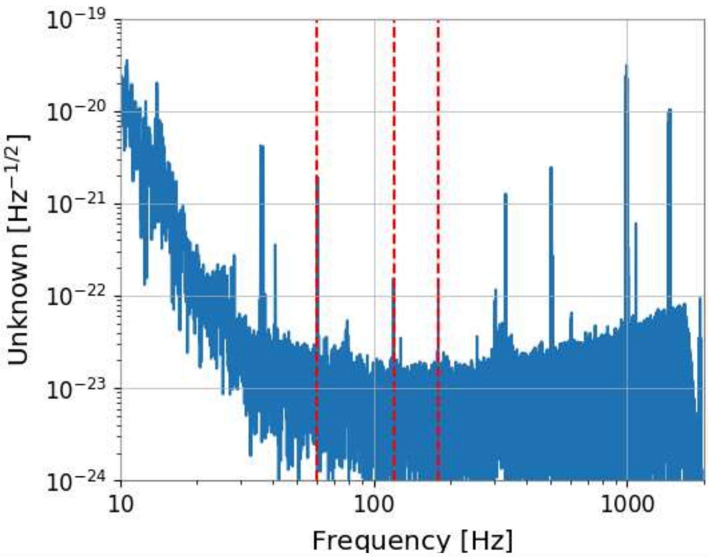
```
                                             51.0/51.0 kB 2.7 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting ligotimegps>=1.2.1 (from gwpy)
  Downloading ligotimegps-2.0.1-py2.py3-none-any.whl.metadata (2.6 kB)
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from gwpy) (3.7.1)
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.10/dist-packages (from gwpy) (1.26.4)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from gwpy) (2.8.2)
Requirement already satisfied: requests>=2.20.0 in /usr/local/lib/python3.10/dist-packages (from gwpy) (2.32.3)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from gwpy) (1.13.1)
Requirement already satisfied: tqdm>=4.10.0 in /usr/local/lib/python3.10/dist-packages (from gwpy) (4.66.5)
Requirement already satisfied: pyerfa>=2.0.1.1 in /usr/local/lib/python3.10/dist-packages (from astropy>=4.3.0->gwpy) (2.0.1.4)
Requirement already satisfied: astropy-iers-data>=0.2024.8.27.10.28.29 in /usr/local/lib/python3.10/dist-packages (from astropy>=4.3.
Requirement already satisfied: PyYAML>=3.13 in /usr/local/lib/python3.10/dist-packages (from astropy>=4.3.0->gwpy) (6.0.2)
Requirement already satisfied: packaging>=19.0 in /usr/local/lib/python3.10/dist-packages (from astropy>=4.3.0->gwpy) (24.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.10/dist-packages (from dateparser>=1.1.4->gwpy) (2024.2)
Requirement already satisfied: regex!=2019.02.19,!=2021.8.27 in /usr/local/lib/python3.10/dist-packages (from dateparser>=1.1.4->gwpy
Requirement already satisfied: tzlocal in /usr/local/lib/python3.10/dist-packages (from dateparser>=1.1.4->gwpy) (5.2)
Collecting igwn-auth-utils>=0.3.1 (from gwdatafind>=1.1.0->gwpy)
  Downloading igwn_auth_utils-1.1.1-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from ligo-segments>=1.0.0->gwpy) (1.16.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (4.54.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (1.4.7)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (10.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=3.3.0->gwpy) (3.2.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20.0->gwpy) (3.4
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20.0->gwpy) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20.0->gwpy) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20.0->gwpy) (2024.8.30
Requirement already satisfied: cryptography>=2.3 in /usr/local/lib/python3.10/dist-packages (from igwn-auth-utils>=0.3.1->gwdatafind>
Collecting safe-netrc>=1.0.0 (from igwn-auth-utils>=0.3.1->gwdatafind>=1.1.0->gwpy)
  Downloading safe_netrc-1.0.1-py3-none-any.whl.metadata (1.9 kB)
Collecting scitokens>=1.7.0 (from igwn-auth-utils>=0.3.1->gwdatafind>=1.1.0->gwpy)
  Downloading scitokens-1.8.1-py3-none-any.whl.metadata (12 kB)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.10/dist-packages (from cryptography>=2.3->igwn-auth-utils>=0.3.1-
Requirement already satisfied: PyJWT>=1.6.1 in /usr/local/lib/python3.10/dist-packages (from scitokens>=1.7.0->igwn-auth-utils>=0.3.1
Requirement already satisfied: pycparser in /usr/local/lib/python3.10/dist-packages (from cffi>=1.12->cryptography>=2.3->igwn-auth-ut
Downloading gwpy-3.0.10-py3-none-any.whl (1.4 MB)
                                         1.4/1.4 MB 22.6 MB/s eta 0:00:00
Downloading dateparser-1.2.0-py2.py3-none-any.whl (294 kB)
                                         295.0/295.0 kB 11.9 MB/s eta 0:00:00
Downloading gwdatafind-1.2.0-py3-none-any.whl (45 kB)
                                         45.5/45.5 kB 2.2 MB/s eta 0:00:00
Downloading gwosc-0.7.1-py3-none-any.whl (27 kB)
Downloading ligotimegps-2.0.1-py2.py3-none-any.whl (19 kB)
Downloading dqsegdb2-1.2.1-py3-none-any.whl (25 kB)
Downloading igwn_auth_utils-1.1.1-py3-none-any.whl (26 kB)
Downloading safe_netrc-1.0.1-py3-none-any.whl (10 kB)
Downloading scitokens-1.8.1-py3-none-any.whl (31 kB)
Building wheels for collected packages: ligo-segments
  Building wheel for ligo-segments (setup.py) ... done
  Created wheel for ligo-segments: filename=ligo_segments-1.4.0-cp310-cp310-linux_x86_64.whl size=99224 sha256=cc1d0cb72c9a0e3835be36
  Stored in directory: /root/.cache/pip/wheels/6d/48/d1/3466977be4e41ba57f92ad0d5619f083df43cf319a151c4e06
Successfully built ligo-segments
Installing collected packages: safe-netrc, ligotimegps, ligo-segments, gwosc, dateparser, scitokens, igwn-auth-utils, gwdatafind, dqs
Successfully installed dateparser-1.2.0 dqsegdb2-1.2.1 gwdatafind-1.2.0 gwosc-0.7.1 gwpy-3.0.10 igwn-auth-utils-1.1.1 ligo-segments-1
```

The following example is from: https://gwpy.github.io/docs/stable/examples/signal/gw150914/

```
# # -- Set a GPS time:
t0 = 1126259462.4    # -- GW150914
# t0 = 1187008882.4    # -- GW170817

from gwpy.timeseries import TimeSeries # Loads in the libraries needed to run the coade
hdata = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478) # Gets the data needed from the file above.


import matplotlib.pyplot as plt # Provides functions for creating plots.

# -- Plot ASD
fig2 = hdata.asd().plot() # Calculates the amplitude spectral density of the hdata and plots it. Amplitude spectral
# density is the measure of the signal's amplitude at different frequencies. Represents the strength of a signal's
# frequency components.

plt.xlim(10,2000) # Sets the lower and upper limit to 10 and 2000, respectively.
ymin = 1e-24 # Sets the y min to 1e-24
```

```
ymax = 1e-19 # Sets y max to 1e-19
plt.ylim(ymin, ymax) # Plots it.
plt.vlines(60, ymin, ymax, linestyle="dashed", color="red") # Draws a vertical dashed line on x=60
plt.vlines(120, ymin, ymax, linestyle="dashed", color="red") # Draws a vertical dashed line on x=120
plt.vlines(180, ymin, ymax, linestyle="dashed", color="red") # Draws a vertical dashed line on x=180
```

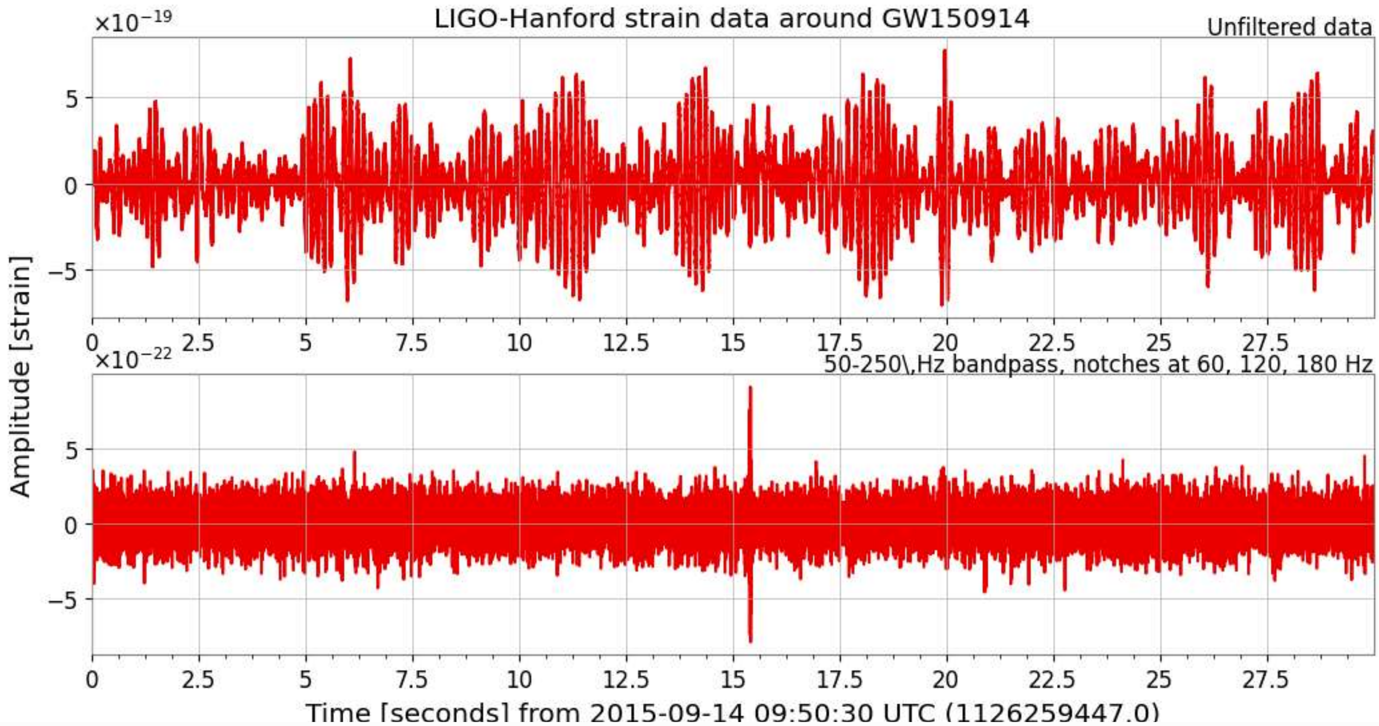
```
<matplotlib.collections.LineCollection at 0x7b9d14008670>
```




```
from gwpy.signal import filter_design # Imports the filter design module from the gwpy.signal package. This package
# designs digital filters.

bp = filter_design.bandpass(50, 250, hdata.sample_rate) # Designs a bandpass filter (bp) using the bandpass function from
# the filter_design module. The FILTER allows frequencies between 50Hz and 250Hz to pass.

notches = [filter_design.notch(line, hdata.sample_rate) for
           line in (60, 120, 180)] # Removes the 60Hz, 120Hz, and 180Hz frequencies that will skew the data if not removed.

zpk = filter_design.concatenate_zpks(bp, *notches) # Combines the bandpass filter and the notch filters into a
# single filter.

hfilt = hdata.filter(zpk, filtfilt=True) # Takes the combined data and applies it to the data forwards and backwards.
# This ensures that the signals timing isnt affected by the filtering process.



hdata = hdata.crop(*hdata.span.contract(1)) # Removes one second from the beginning and end of hdata.
hfilt = hfilt.crop(*hfilt.span.contract(1))# Removes one second from the beginning and end of hfilt.


print(*hdata.span.contract(1)) # Prints the start and end times of the contracted time span of hdata.
```

```
1126259448.0 1126259476.0
```

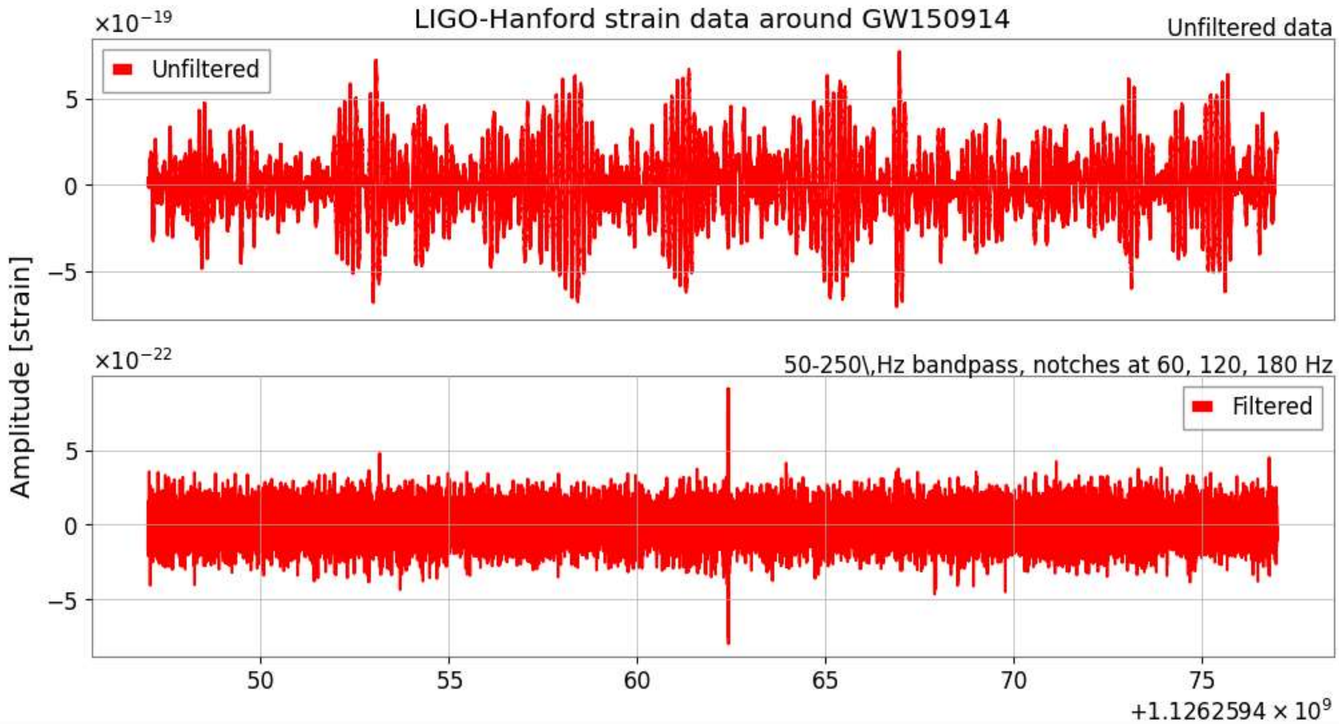
```
from gwpy.plot import Plot # Imports the necessary plotting function.
plot = Plot(hdata, hfilt, figsize=[12, 6], separate=True, sharex=True, # Creates a new plot object given the data,
            # figure size, etc.
            color='gwpy:ligo-hanford') # sets the color of the plot lines to the LIGO Hanford color scheme
ax1, ax2 = plot.axes #  assigns the two subplot axes to variables for further customization.
ax1.set_title('LIGO-Hanford strain data around GW150914') # Sets the title of the graph
ax1.text(1.0, 1.01, 'Unfiltered data', transform=ax1.transAxes, ha='right') # Adds a label to the first subplot
ax1.set_ylabel('Amplitude [strain]', y=-0.2) # Sets the y-axis label for the first subplot.
ax2.set_ylabel('')
ax2.text(1.0, 1.01, r'50-250\,Hz bandpass, notches at 60, 120, 180 Hz',
         transform=ax2.transAxes, ha='right') # Labels the graph below the unflitered data as the one filtered from
         # the above frequencies.
plot.show() # Shows the plot.
```



Now using functions in matlibplot to plot without using gwpy

```
plt.figure(figsize=[12, 6]) # Creates a new figure with a specified size
ax1 = plt.subplot(2, 1, 1) # Creates the first subplot in a 2 by 1 grid.
ax1.plot(hdata.times.value, hdata.value, color="red", label="Unfiltered") # Plots unfiltered data on the first subplot with
# time on the x-axis and strain values on the y-axis.
plt.legend() # Displays a legend
ax1.set_title('LIGO-Hanford strain data around GW150914') # titles the subplot.
ax1.text(1.0, 1.01, 'Unfiltered data', transform=ax1.transAxes, ha='right') #Adds a label to the first subplot.
ax1.xaxis.set_visible(False) # Suppresses the x-axis
ax2 = plt.subplot(2, 1, 2) #
ax2.plot(hfilt.times.value, hfilt.value, color="red", label="Filtered")
ax1.set_ylabel('Amplitude [strain]', y=-0.2) # y=-0.2 Asking the top graph to be shifted down by 20 percent.
ax2.set_ylabel('') # Its the same data just shown differently than the top graph, so there is no need for re-labeling.
ax2.text(1.0, 1.01, r'50-250\,Hz bandpass, notches at 60, 120, 180 Hz',
         transform=ax2.transAxes, ha='right') # Will allign to the right edge of the plot.
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7b9d0fd96dd0>
```

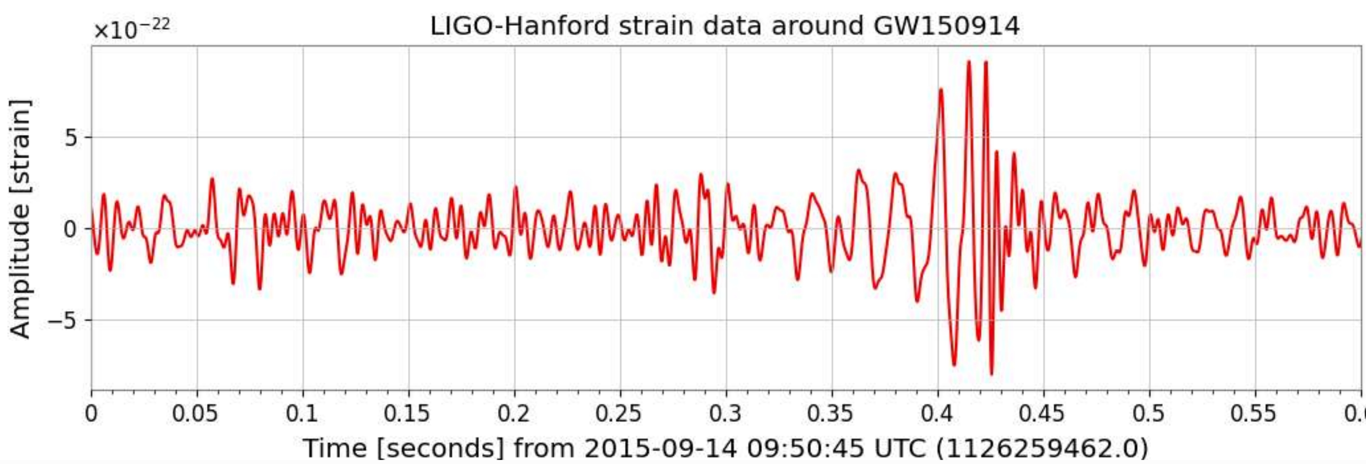


Start coding or generate with AI.

```
hdata.value # extracts the raw strain data from hdata TimeSeries
```

```
array([9.06730891e-21, 2.01178871e-20, 3.71290472e-20, ...,
       2.06988575e-19, 2.00082704e-19, 1.86419171e-19])
```

```
import matplotlib.pyplot as plt # imports the module used for creating plots.

plot = hfilt.plot(color='gwpy:ligo-hanford') # This line creates plot og the filtered strain data
ax = plot.gca() # Gets the current axes of the plot and assigns it to the variable ax
ax.set_title('LIGO-Hanford strain data around GW150914') # Sets the title of the plot.
ax.set_ylabel('Amplitude [strain]') # Sets the y-label
ax.set_xlim(1126259462, 1126259462.6) # The zooming function just changes the low and high limits of the graph listed above to a MUCH smalle
ax.set_xscale('seconds', epoch=1126259462) # sets the x axis scale to seconds, with the specified refrence for the
# time values.
plot.show() # Shows the plot.

x_val = plt.gca().lines[0].get_xdata() # This line gets the x-axis data from the first line of the plot and assigns it to
# the variable x_val.
y_val = plt.gca().lines[0].get_ydata() # Gets the y-axis data from the first line of the plot and assigns it to the
# the variable y_val.
```



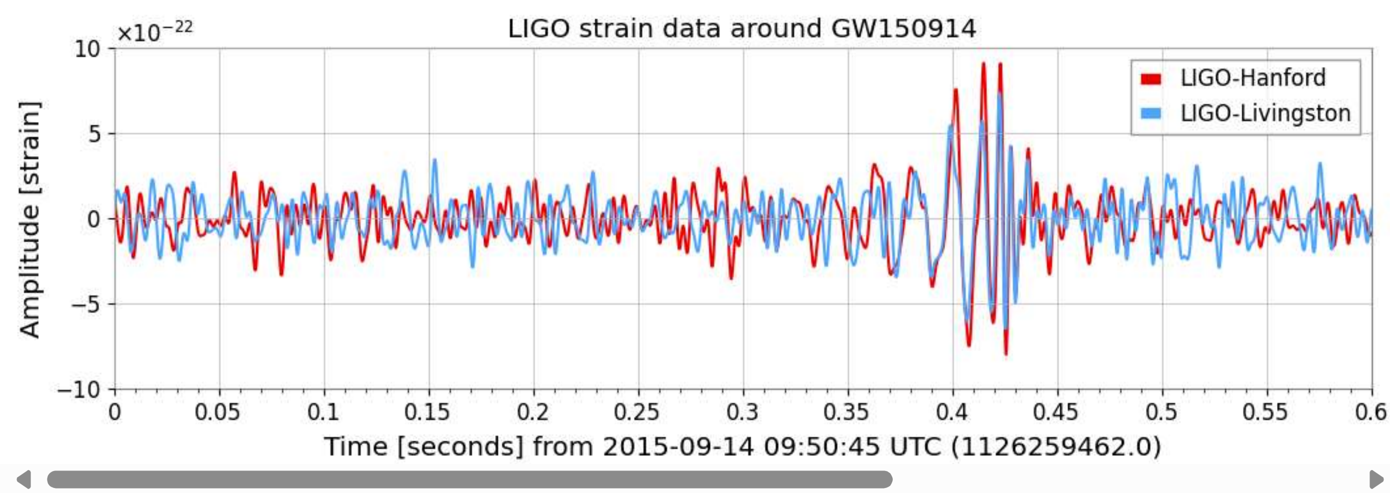
```
ldata = TimeSeries.fetch_open_data('L1', 1126259446, 1126259478) # Retrieves a specific part of the livingston strain data.
lfilt = ldata.filter(zpk, filtfilt=True) # to notch out the unecessary frequencies.
```

```
lfilt.shift('6.9ms') # This is the time delay. Because the two detectors are at different locations, they will receive
# the gravitational wave signal at slightly different times. This time difference compensates for that.
lfilt *= -1 # This flips the sign of all the data values in lfit. Corrects for the relative orientations of the
# two LIGO detectors, and flips the data upside down.
```

```
print(0.0069*3e8)
```

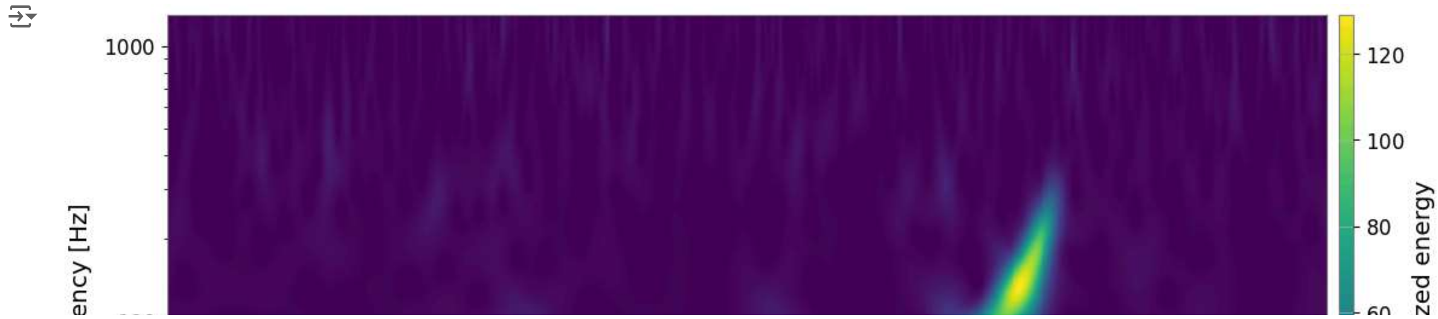
```
2070000.0
```

```
plot = Plot(figsize=[12, 4]) # Creates a new plot object
ax = plot.gca() # Gets the currect axes of the plot and assigns it to "ax".
ax.plot(hfilt, label='LIGO-Hanford', color='gwpy:ligo-hanford') # Plots the Hanford data on the axes,
# with a label and color for the Hanford detector.
ax.plot(lfilt, label='LIGO-Livingston', color='gwpy:ligo-livingston') # Plots the Livingston data on the same axes.
ax.set_title('LIGO strain data around GW150914') # Title of the combined data.
ax.set_xlim(1126259462, 1126259462.6) # Sets the limits of the x-axis, zooming in on the important part of the data.
ax.set_xscale('seconds', epoch=1126259462) # Sets the x-axis scale to seconds
ax.set_ylabel('Amplitude [strain]') # Sets the label for the y-axis of the plot.
ax.set_ylim(-1e-21, 1e-21)  # Sets the limit for the y-axis
ax.legend() # displays the legend defined previously
plot.show() # shows the plot.
```



https://colab.research.google.com/github/losc-tutorial/quickview/blob/master/index.ipynb Also from: https://gwpy.github.io/docs/stable/examples/signal/qscan/

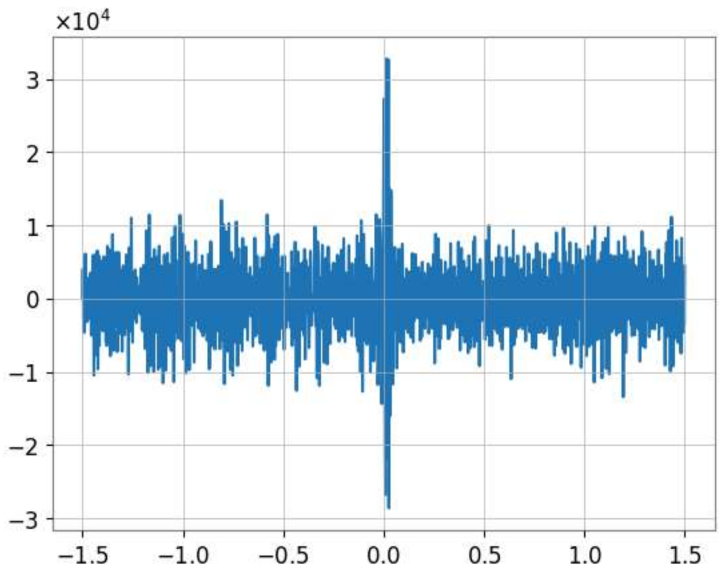
```
dt = 0.2  #-- Set width of q-transform plot, in seconds
hq = hfilt.q_transform(outseg=(t0-dt, t0+0.1))# Performs the q-transform on hfilt data. outseg specifies the time segment.
fig4 = hq.plot() # Creates a plot of the Q-tranform results (hq)
ax = fig4.gca() # Gets the current axes of the plot and assigns it to varaible ax
fig4.colorbar(label="Normalized energy") # adds a color bar to the chart to define the level of normalized energy.
ax.grid(False) # Disables grid lines.
ax.set_yscale('log') # Sets the y axis scal to logaritmic, for better visualization of frequency.
```




```
from scipy.io.wavfile import write # Imports the write function from the cipy.io.wavfile module. Allows array data from NUmPy
# to be written as a audio file.
import numpy as np # Imports numpy module.
```

```
amplitude = np.iinfo(np.int16).max # Gets the maximum value for a 16 bit integer. Used to scale strain data for audio
# representation.

ind = np.where((x_val < (t0+1.5)) & (x_val > (t0-1.5))) # Specifies the range of the event
y = y_val[ind] # Extracts the strain data values given the "ind" in the previous line.
# y = y**3
y = y / np.max(y) # Normalizes the strain data.
plt.plot(x_val[ind] - t0, (np.array(y) * amplitude).astype(np.int16)) # Plots the processed strain data.

# This box prepares the the strain data for an audio conversion.
```

```
[<matplotlib.lines.Line2D at 0x7b9d108f54e0>]
```




```
fs = int(1 / np.median(np.diff(np.array(x_val[ind] - t0)))) #Calculates the sampling frequency (fs) by taking the
# reciprocal of the median time.
print("fs = ", fs) # Prints the caluclated sampling frequency
```