LIGO Open Science Center Tutorial

Chris Messenger[1], Siong Heng[1], Xilong Fan[2], Mervyn Chan[1], Martin Hendry[1]

1. University of Glasgow, 2. Hubei University of Education

LIGO Open Science Center

LIGO is operated by California Institute of Technology and Massachusetts Institute of Technology
and supported by the U.S. National Science Foundation.

All tutorials can be found at https://losc.ligo.org/tutorials/

- **There are multiple experts available to help you. Please ask them for help.**

- **Also try to make use of the internet (e.g. sites like stackoverflow.com) for help with problems.**

- **If you are finding things easy then please use progress ahead at your own speed through this presentation OR follow the online tutorials at** https://losc.ligo.org/tutorials/

- **Remember to save your work because we will be continuing this analysis tomorrow.**

Accessing the data

Online version

Download LIGO data using python

```
import json
import urllib2

#t0 = 933661015.0  # Time of a successful s6 hardware injection in H1 (SNR 140)
t0 = 932422615.0 # Time of a successful s6 hardware injection in H1 (SNR 40)
dataset = 'S6'
detector = 'H1'
version = 'V1'  # V1 is "version 1" of the data release

observatory = detector[0]          # first letter of the detector H or L
hour        = int(t0)&0xFFFFF000   # the filename rounding down to a multiple of 4096
fortnight   = int(t0)&0xFFF00000   # the directory by rounding down to multiple of 4096*256
filename = '{0}-{1}_LOSC_4_{2}-{3}-4096.hdf5'.format(observatory, detector, version, hour)
urlformat = 'https://losc.ligo.org/archive/data/{0}/{1}/{2}'
url = urlformat.format(dataset, fortnight, filename)

# -- Uncomment these 3 lines to run this notebook on GW150914 instead!
#t0 = 1126259462.43 # GW150914
#url = 'https://losc.ligo.org/s/events/GW150914/H-H1_LOSC_4_V1-1126259446-32.hdf5'
#filename = url.split('/')[-1]

print('Downloading ' + url + '  (this might take a few minutes...)')
r = urllib2.urlopen(url).read()
f = open(filename, 'w')   # write it to the right filename
f.write(r)
f.close()
print("File download complete")
```

Exercise: *Try downloading different datasets at different times.*

Accessing the data

Offline version

To begin, get the necessary files, by downloading the zip file and unpacking it into single directory:

- https://losc.ligo.org/s/events/LOSC_Event_tutorial_v1.4.zip

Or obtaining it from the USB stick being passed around the class

This zip file contains:

- this IPython notebook LOSC_Event_tutorial.ipynb, and LOSC_Event_tutorial.py code.
- python code for reading LOSC data files: readligo.py.
- the event data files (32s sampled at 4096 Hz, in hdf5 format, for both LIGO detectors).
- Waveform templates (32s sampled at 4096 Hz, in hdf5 format, for both plus and cross polarizations).
- a parameter file in json format, O1_events.json .

Exercise: *Verify that you can import readligo*

Reading the data

Set the event name to choose event and the plot type

```
#-- SET ME   Tutorial should work with most binary black hole events
#-- Default is no event selection; you MUST select one to proceed.
eventname = ''
# eventname = 'GW150914'
# eventname = 'GW151226'
# eventname = 'LVT151012'

# want plots?
make_plots = 1
plottype = "png"
#plottype = "pdf"
```

Exercise: *Choose an event to analyse.*

Import the required modules and set up IPython

```
# Standard python numerical analysis imports:
import numpy as np
from scipy import signal
from scipy.interpolate import interp1d
from scipy.signal import butter, filtfilt, iirdesign, zpk2tf, freqz
import h5py
import json

# the IPython magic below must be commented out in the .py file,
# since it doesn't work there.
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab

# LIGO-specific readligo.py
import readligo as rl

# you might get a matplotlib warning here; you can ignore it.
```

Open the local json file (download in advance):

```
# Read the event properties from a local json file
fnjson = "O1_events.json"
try:
    events = json.load(open(fnjson,"r"))
except IOError:
    print("Cannot find resource file "+fnjson)
    print("You can download it from https://losc.ligo.org/s/events/"+fnjson)
    print("Quitting.")
    quit()

# did the user select the eventname ?
try:
    events[eventname]
except:
    print('You must select an eventname that is in '+fnjson+'! Quitting.')
    quit()
```

Exercise: *Open the json file in a text editor and see what it looks like.*

Read the event properties from a local json file (download in advance):

```
# Extract the parameters for the desired event:
event = events[eventname]
fn_H1 = event['fn_H1']                # File name for H1 data
fn_L1 = event['fn_L1']                # File name for L1 data
fn_template = event['fn_template']    # File name for template waveform
fs = event['fs']                      # Set sampling rate
tevent = event['tevent']              # Set approximate event GPS time
fband = event['fband']                # frequency band for bandpassing signal
print("Reading in parameters for event " + event["name"])
print(event)
```

Exercise: *Look at the values of the parameters, do they make sense?*

Read in the data

```
#---------------------------------------------------------------
# Load LIGO data from a single file.
# FIRST, define the filenames fn_H1 and fn_L1, above.
#---------------------------------------------------------------
try:
    # read in data from H1 and L1, if available:
    strain_H1, time_H1, chan_dict_H1 = rl.loaddata(fn_H1, 'H1')
    strain_L1, time_L1, chan_dict_L1 = rl.loaddata(fn_L1, 'L1')
except:
    print("Cannot find data files!")
    print("You can download them from https://losc.ligo.org/s/events/"+eventname)
    print("Quitting.")
    quit()

# read in the NR template
NRtime, NR_H1 = np.genfromtxt('GW150914_4_NR_waveform.txt').transpose()
```

Exercise: *What is now stored in strain_H1, time_H1, chan_dict_H1?*

Data Gaps

NOTE that in general, LIGO strain time series data has gaps (filled with NaNs) when the detectors are not taking valid ("science quality") data.

Analyzing these data requires the user to loop over “segments” of valid data stretches.

In this tutorial, for simplicity, we assume there are no data gaps - this will not work for all times!

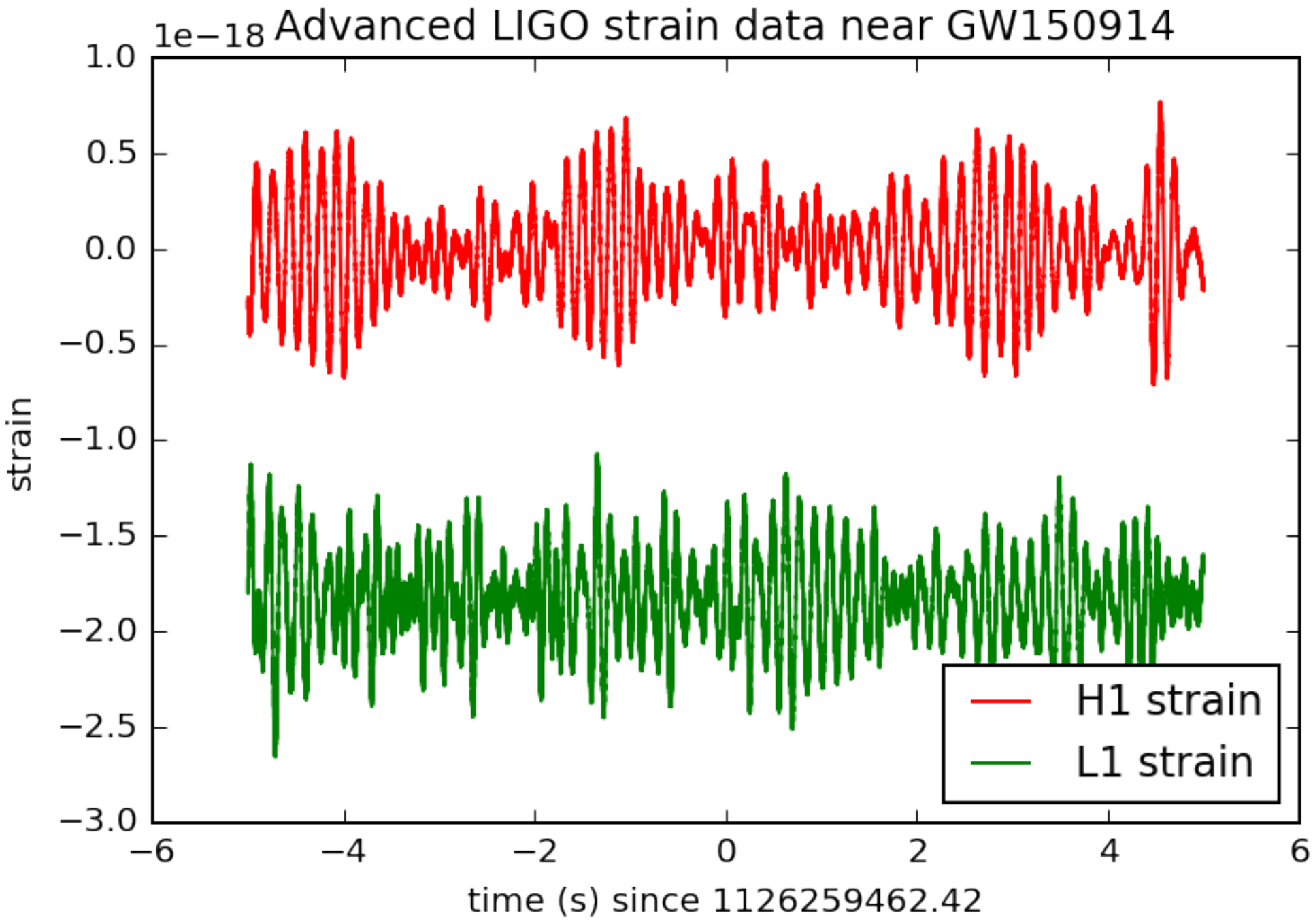
First look at the data from H1 and L1

```
# both H1 and L1 will have the same time vector, so:
time = time_H1
# the time sample interval (uniformly sampled!)
dt = time[1] - time[0]

# Let's look at the data and print out some stuff:

print('time_H1: len, min, mean, max = ', \
    len(time_H1), time_H1.min(), time_H1.mean(), time_H1.max() )
print('strain_H1: len, min, mean, max = ', \
    len(strain_H1), strain_H1.min(),strain_H1.mean(),strain_H1.max())
print( 'strain_L1: len, min, mean, max = ', \
    len(strain_L1), strain_L1.min(),strain_L1.mean(),strain_L1.max())

# What's in chan_dict?  (See also https://losc.ligo.org/tutorials/)
bits = chan_dict_H1['DATA']
print("For H1, {0} out of {1} seconds contain usable DATA”.format(bits.sum(), \
    len(bits)))
bits = chan_dict_L1['DATA']
print("For L1, {0} out of {1} seconds contain usable DATA".format(bits.sum(), \
    len(bits)))
```

Plot the data

```
# plot +- deltat seconds around the event:
# index into the strain time series for this time interval:
deltat = 5
indxt = np.where((time >= tevent-deltat) & (time < tevent+deltat))
print(tevent)

if make_plots:
plt.figure()
plt.plot(time[indxt]-tevent,strain_H1[indxt],'r',label='H1 strain')
plt.plot(time[indxt]-tevent,strain_L1[indxt],'g',label='L1 strain')
plt.xlabel('time (s) since '+str(tevent))
plt.ylabel('strain')
plt.legend(loc='lower right')
plt.title('Advanced LIGO strain data near '+eventname)
plt.savefig(eventname+'_strain.'+plottype)
```

Exercise: *Open the images, can you see the signal?*


Advanced LIGO strain data near GW150914
1e−18
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
strain
−6
−4
−2
0
2
4
6
time (s) since 1126259462.42
H1 strain
L1 strain

Exercise: Look at all 3 of the detected LIGO signals. What are the main differences that you see?

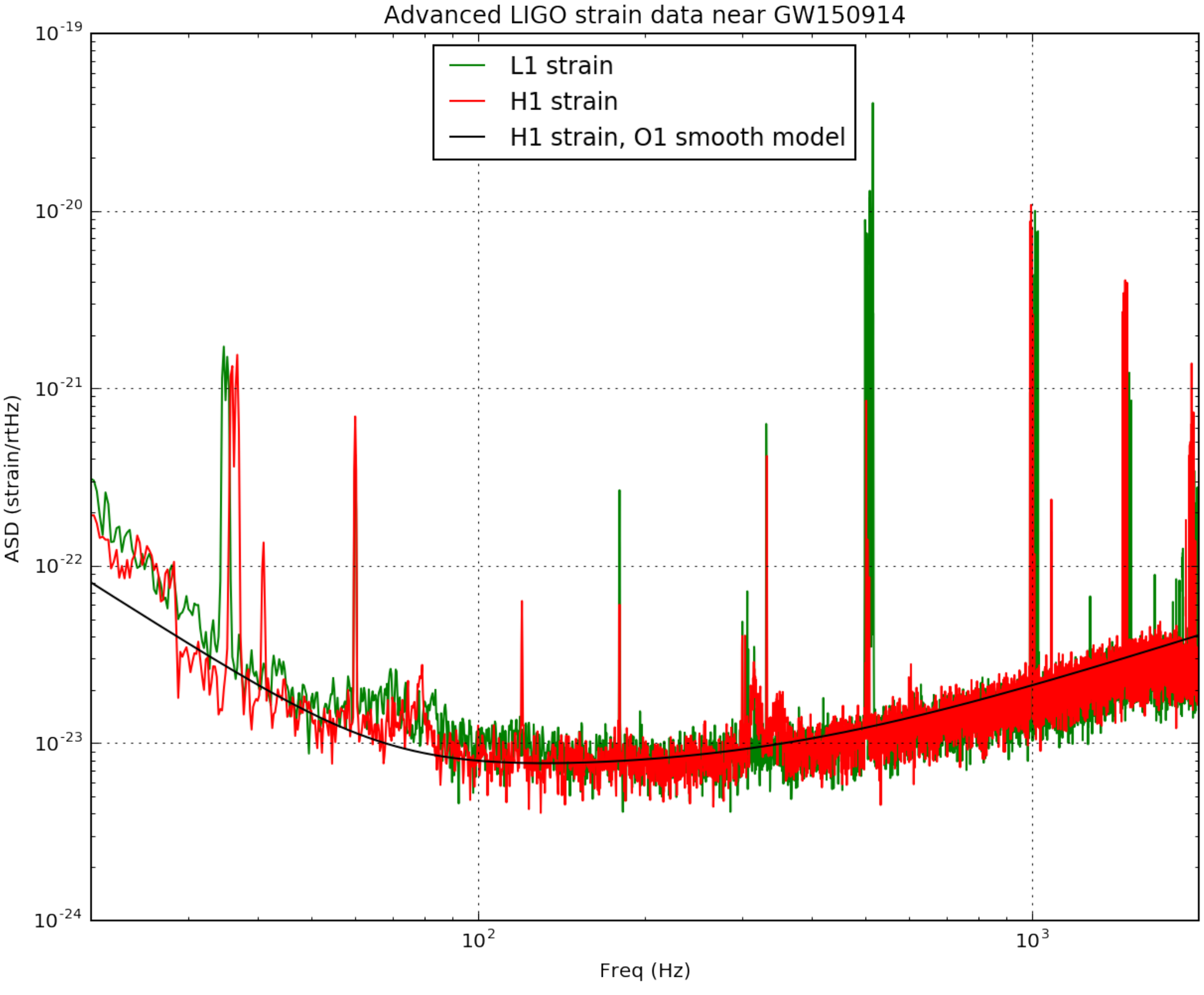
The detector noise computing the amplitude spectral density

Compute the Amplitude Spectral Density (ASD)

```
# number of sample for the fast fourier transform:
NFFT = 4*fs
Pxx_H1, freqs = mlab.psd(strain_H1, Fs = fs, NFFT = NFFT)
Pxx_L1, freqs = mlab.psd(strain_L1, Fs = fs, NFFT = NFFT)

# We will use interpolations of the ASDs computed above for whitening:
psd_H1 = interp1d(freqs, Pxx_H1)
psd_L1 = interp1d(freqs, Pxx_L1)

# Here is an approximate, smoothed PSD for H1 during O1, with no lines.
# We'll use it later.
Pxx = (1.e-22*(18./(0.1+freqs))**2)**2+0.7e-23**2+((freqs/2000.)*4.e-23)**2
psd_smooth = interp1d(freqs, Pxx)
```

Plot the Amplitude Spectral Density (ASD)

```
# plot the ASDs, with the template overlaid:
f_min = 20.
f_max = 2000.
plt.figure(figsize=(10,8))
plt.loglog(freqs, np.sqrt(Pxx_L1),'g',label='L1 strain')
plt.loglog(freqs, np.sqrt(Pxx_H1),'r',label='H1 strain')
plt.loglog(freqs, np.sqrt(Pxx),'k',label='H1 strain, O1 smooth model')
plt.axis([f_min, f_max, 1e-24, 1e-19])
plt.grid('on')
plt.ylabel('ASD (strain/rtHz)')
plt.xlabel('Freq (Hz)')
plt.legend(loc='upper center')
plt.title('Advanced LIGO strain data near '+eventname)
plt.savefig(eventname+'_ASDs.'+plottype)
```


Advanced LIGO strain data near GW150914
L1 strain
H1 strain
H1 strain, O1 smooth model
ASD (strain/rtHz)
Freq (Hz)
10^-19
10^-20
10^-21
10^-22
10^-23
10^-24
10^2
10^3

Whiten the data

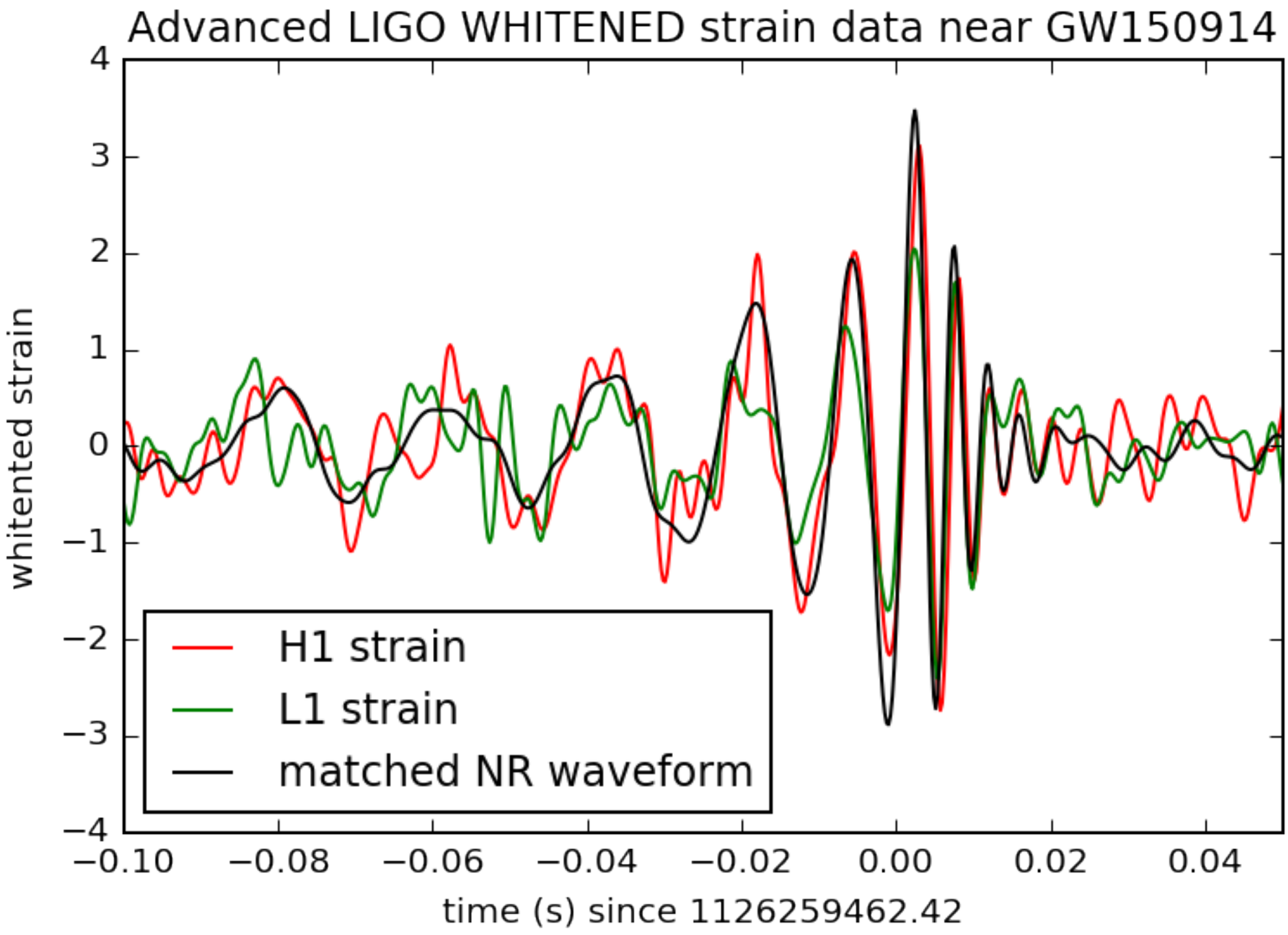
```
# function to whiten data
def whiten(strain, interp_psd, dt):
    Nt = len(strain)
    freqs = np.fft.rfftfreq(Nt, dt)

    # whitening: transform to freq domain, divide by asd, then
    # transform back, taking care to get normalization right.
    hf = np.fft.rfft(strain)
    white_hf = hf / (np.sqrt(interp_psd(freqs) /dt/2.))
    white_ht = np.fft.irfft(white_hf, n=Nt)
    return white_ht


# now whiten the data from H1 and L1:
strain_H1_whiten = whiten(strain_H1,psd_H1,dt)
strain_L1_whiten = whiten(strain_L1,psd_L1,dt)
```

Exercise: Plot the data again, *can you see the signal now?*

Bandpass the data

```
# We need to suppress the high frequency noise (no signal!)
# with some bandpassing:
bb, ab = butter(4, [fband[0]*2./fs, fband[1]*2./fs], btype='band')
strain_H1_whitenbp = filtfilt(bb, ab, strain_H1_whiten)
strain_L1_whitenbp = filtfilt(bb, ab, strain_L1_whiten)
```

Exercise: Plot the data again, *can you see the signal now?*


Advanced LIGO WHITENED strain data near GW150914
whitented strain
4
3
2
1
0
−1
−2
−3
−4
−0.10
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
time (s) since 1126259462.42
H1 strain
L1 strain
matched NR waveform

How sensitive?

Working our how far we can detect signals

Binary Neutron Star (BNS) detection range

```
#-- compute the binary neutron star (BNS) detectability range

#-- choose a detector noise power spectrum:
f = freqs.copy()
# get frequency step size
df = f[2]-f[1]
#-- constants
# speed of light:
clight = 2.99792458e8                    # m/s
# Newton's gravitational constant
G = 6.67259e-11                          # m^3/kg/s^2
# one parsec, popular unit of astronomical distance (around 3.26 light
years)
parsec = 3.08568025e16                   # m
# solar mass
MSol = 1.989e30                          # kg
# solar mass in seconds (isn't relativity fun?):
tSol = MSol*G/np.power(clight,3)         # s
# Single-detector SNR for detection above noise background:
SNRdet = 8.
# conversion from maximum range (horizon) to average range:
Favg = 2.2648
# mass of a typical neutron star, in solar masses:
mNS = 1.4
```

```
# Masses in solar masses
m1 = m2 = mNS
mtot = m1+m2  # the total mass
eta = (m1*m2)/mtot**2  # the symmetric mass ratio
mchirp = mtot*eta**(3./5.)  # the chirp mass (FINDCHIRP, following Eqn 3.1b)

# distance to a fiducial BNS source:
dist = 1.0                                  # in Mpc
Dist =  dist * 1.0e6 * parsec /clight # from Mpc to seconds

# We integrate the signal up to the frequency of
# the "Innermost stable circular orbit (ISCO)"

# Orbital separation at ISCO, in geometric units.
# 6M for PN ISCO; 2.8M for EOB
R_isco = 6.
# frequency at ISCO (end the chirp here; the merger and ringdown follow)
f_isco = 1./(np.power(R_isco,1.5)*np.pi*tSol*mtot)
# minimum frequency (below which, detector noise
# is too high to register any signal):
f_min = 20. # Hz
# select the range of frequencies between f_min and isco
fr = np.nonzero(np.logical_and(f > f_min , f < f_isco))
# get the frequency and spectrum in that range:
ffr = f[fr]
```

```
# In stationary phase approx, this is htilde(f):
# See FINDCHIRP Eqns 3.4, or 8.4-8.5
htilde = (2.*tSol/Dist)*np.power(mchirp,5./6.)*np.sqrt(5./96./np.pi)*(np.pi*tSol)
htilde *= np.power(np.pi*tSol*ffr,-7./6.)
htilda2 = htilde**2

# loop over the detectors
dets = ['H1', 'L1']
for det in dets:
    if det is 'L1': sspec = Pxx_L1.copy()
    else:           sspec = Pxx_H1.copy()
    sspecfr = sspec[fr]
    # compute "inspiral horizon distance" for optimally
    # oriented binary; FINDCHIRP Eqn D2:
    D_BNS = np.sqrt(4.*np.sum(htilda2/sspecfr)*df)/SNRdet
    # and the "inspiral range", averaged over source direction
    # and orientation:
    R_BNS = D_BNS/Favg
    print(det+' BNS inspiral horizon = {0:.1f} Mpc, BNS \
        inspiral range   = {1:.1f} Mpc'.format(D_BNS,R_BNS))
```

Exercise: What are the ranges for a binary black hole system?

Spectrograms

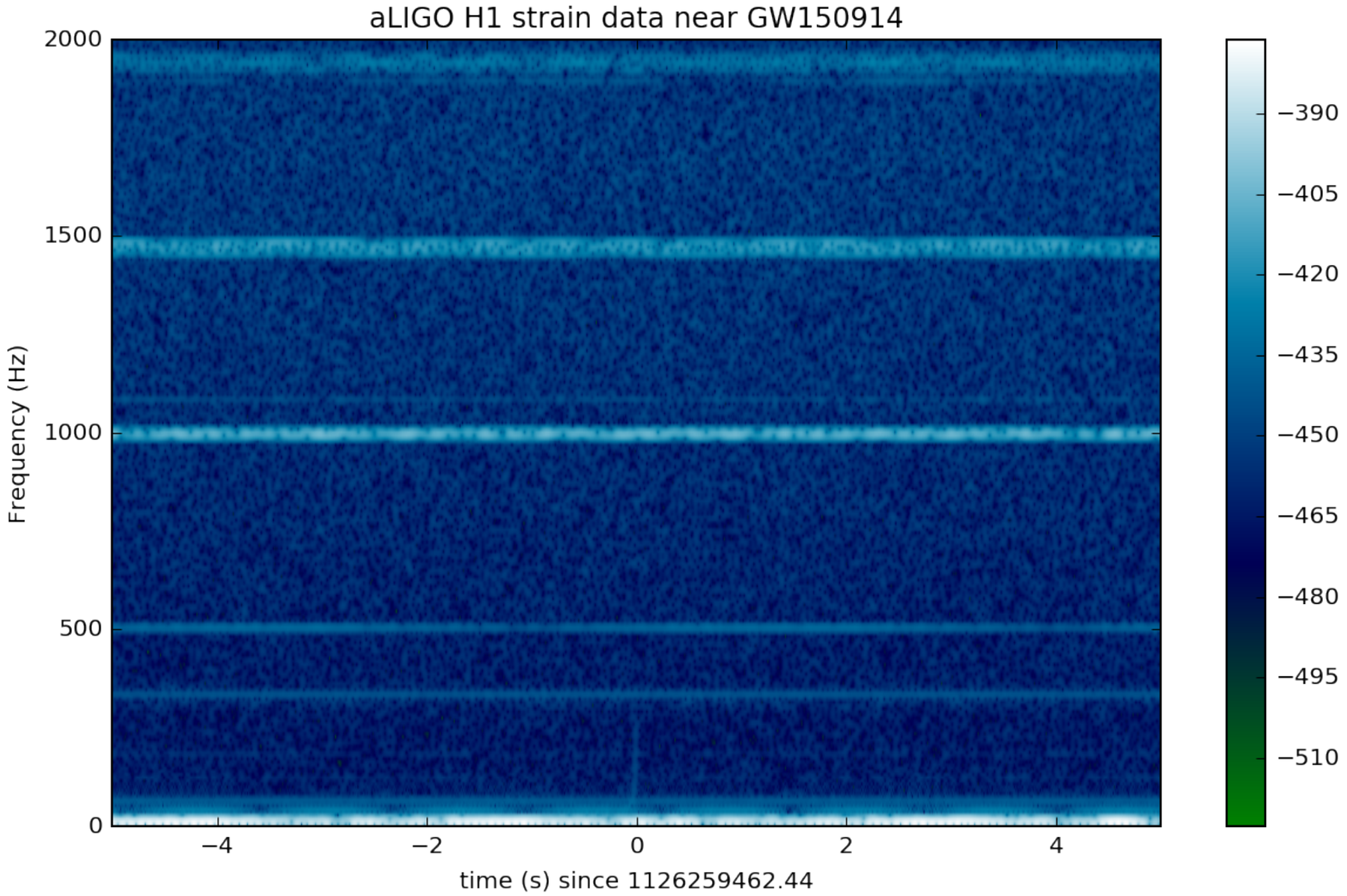
Or time-frequency plots

Spectrograms

```
# index into the strain time series for this time interval:
indxt = np.where((time >= tevent-deltat) & (time < tevent+deltat))

# pick a shorter FTT time interval, like 1/8 of a second:
NFFT = int(fs/8)
# and with a lot of overlap, to resolve short-time features:
NOVL = int(NFFT*15./16)
# and choose a window that minimizes "spectral leakage"
# (https://en.wikipedia.org/wiki/Spectral_leakage)
window = np.blackman(NFFT)

# the right colormap is all-important! See:
# http://matplotlib.org/examples/color/colormaps_reference.html
# viridis seems to be the best for our purposes, but it's new;
# if you don't have it, you can settle for ocean.

#spec_cmap='viridis'
spec_cmap='ocean'
```

```
# Plot the H1 spectrogram:
plt.figure(figsize=(10,6))
spec_H1, freqs, bins, im = plt.specgram(strain_H1[indxt], NFFT=NFFT, Fs=fs, window=window,
        noverlap=NOVL, cmap=spec_cmap, xextent=[-deltat,deltat])
plt.xlabel('time (s) since ‘+str(tevent))
plt.ylabel('Frequency (Hz)’)
plt.colorbar()
plt.axis([-deltat, deltat, 0, 2000])
plt.title('aLIGO H1 strain data near ‘+eventname)
plt.savefig(eventname+’_H1_spectrogram.'+plottype)
```

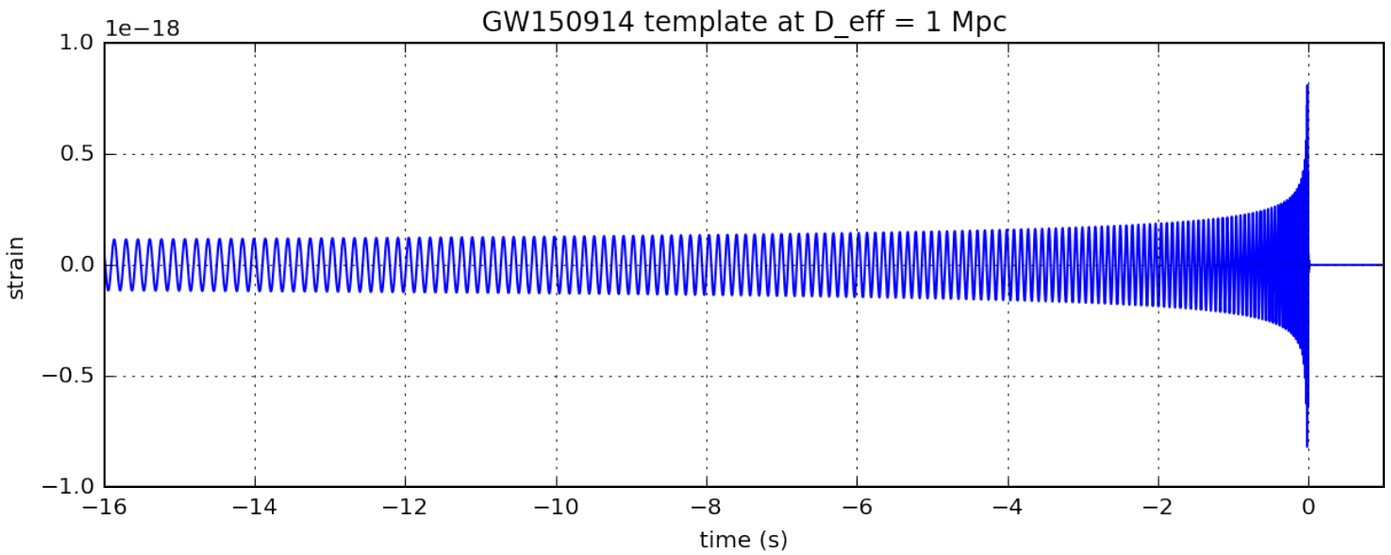
```
# Plot the L1 spectrogram:
plt.figure(figsize=(10,6))
spec_H1, freqs, bins, im = plt.specgram(strain_L1[indxt], NFFT=NFFT, Fs=fs, window=window,
        noverlap=NOVL, cmap=spec_cmap, xextent=[-deltat,deltat])
plt.xlabel('time (s) since ‘+str(tevent))
plt.ylabel('Frequency (Hz)’)
plt.colorbar()
plt.axis([-deltat, deltat, 0, 2000])
plt.title('aLIGO L1 strain data near ‘+eventname)
plt.savefig(eventname+'_L1_spectrogram.'+plottype)
```



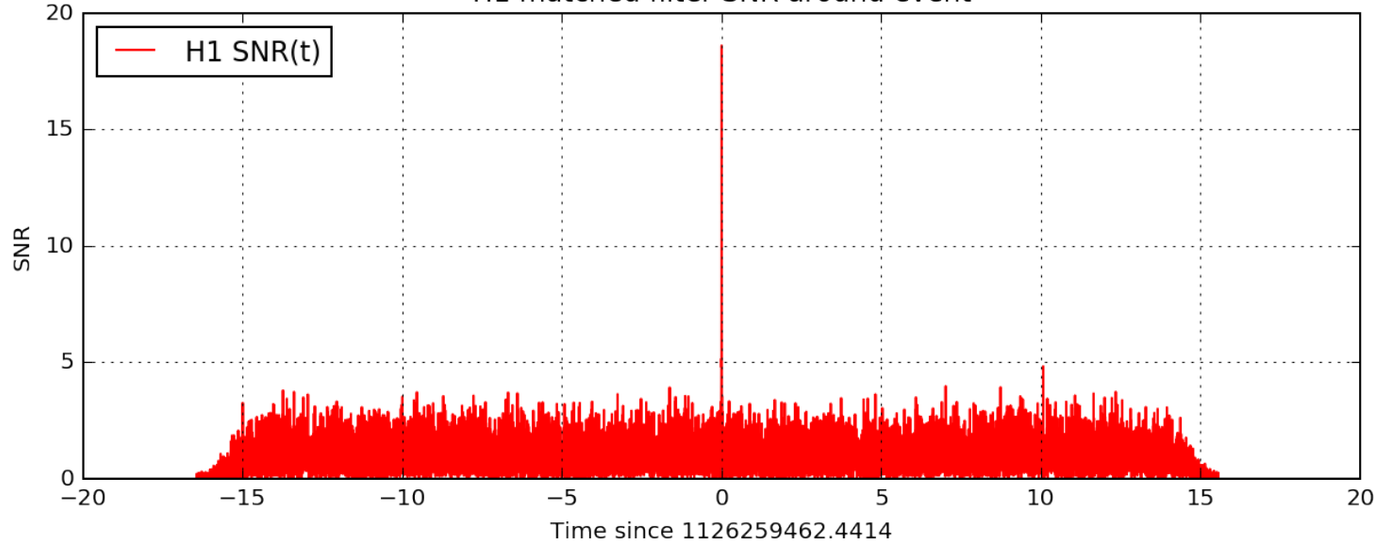
Exercise: Plot the whitened data and zoom in on the signal

Advanced steps

1. Waveform template
2. Matched filtering
3. Make sound Files



H1 matched filter SNR around event



Thanks for your attention