from pycbc.frame import read\_frame

import urllib.request

import pylab

import pycbc

from pycbc import frame

from pycbc.catalog import Merger

from pycbc.filter import resample\_to\_delta\_t, highpass

from pycbc.filter import matched\_filter

from pycbc.waveform import get\_td\_waveform

from pycbc.filter import sigma

from pycbc.psd import interpolate, inverse\_spectrum\_truncation

from pycbc.types import TimeSeries

import numpy

%matplotlib inline

# -- download data

! wget https://www.gw-openscience.org/s/workshop3/challenge/challenge3.gwf

# -- for pycbc

urllib.request.urlretrieve("https://www.gw-openscience.org/s/workshop3/challenge/challenge3.gwf", "challenge3.gwf")

pycbc\_strain = read\_frame('challenge3.gwf', 'H1:CHALLENGE3')

print(pycbc\_strain)

pycbc\_strain = highpass(pycbc\_strain, 20)

pycbc\_strain = resample\_to\_delta\_t(pycbc\_strain, 1.0/2048)

#Plot the data in the time-domain.

conditioned = pycbc\_strain.crop(2, 2)

psd = conditioned.psd(4)

# Now that we have the psd we need to interpolate it to match our data

# and then limit the filter length of 1 / PSD. After this, we can

# directly use this PSD to filter the data in a controlled manner

psd = interpolate(psd, conditioned.delta\_f)

# 1/PSD will now act as a filter with an effective length of 4 seconds

# Since the data has been highpassed above 20 Hz, and will have low values

# below this we need to informat the function to not include frequencies

# below this frequency.

psd = inverse\_spectrum\_truncation(psd, int(4 \* conditioned.sample\_rate),

                                  low\_frequency\_cutoff=20)

print(psd.sample\_frequencies)

pylab.plot(psd.sample\_frequencies, psd, label="Spectral Data")

pylab.yscale('log')

pylab.xscale('log')

pylab.ylim(1e-47, 1e-41)

pylab.xlim(20, 1024)

pylab.ylabel('$Strain^2 / Hz$')

pylab.xlabel('Frequency (Hz)')

pylab.grid()

pylab.legend()

pylab.show()

#PSD Plotting ends

#plot the strain data

pylab.plot(conditioned.sample\_times , conditioned)

pylab.xlabel('Time (s)')

pylab.show()

# Strain data plotting ends

#Generate waveform with mass m = 30 and spin = 0

m = 10 # Solar masses

hp, hc = get\_td\_waveform(approximant="SEOBNRv4\_opt",

                     mass1=m,

                     mass2=m,

                     delta\_t=conditioned.delta\_t,

                     f\_lower=20)

# Resize the vector to match our data

hp.resize(len(conditioned))

pylab.figure()

pylab.title('Before shifting')

pylab.plot(hp.sample\_times, hp)

pylab.xlabel('Time (s)')

pylab.ylabel('Strain')

template = hp.cyclic\_time\_shift(hp.start\_time)

pylab.figure()

pylab.title('After shifting')

pylab.plot(template.sample\_times, template)

pylab.xlabel('Time (s)')

pylab.ylabel('Strain')

#Wave form generation ends

#Create Matched filter

snr = matched\_filter(template, conditioned,

                     psd=psd, low\_frequency\_cutoff=20)

# Remove time corrupted by the template filter and the psd filter

# We remove 4 seconds at the beginning and end for the PSD filtering

# And we remove 4 additional seconds at the beginning to account for

# the template length (this is somewhat generous for

# so short a template). A longer signal such as from a BNS, would

# require much more padding at the beginning of the vector.

snr = snr.crop(4 + 4, 4)

# Why are we taking an abs() here?

# The `matched\_filter` function actually returns a 'complex' SNR.

# What that means is that the real portion correponds to the SNR

# associated with directly filtering the template with the data.

# The imaginary portion corresponds to filtering with a template that

# is 90 degrees out of phase. Since the phase of a signal may be

# anything, we choose to maximize over the phase of the signal.

pylab.figure(figsize=[10, 4])

pylab.plot(snr.sample\_times, abs(snr))

pylab.ylabel('Signal-to-noise')

pylab.xlabel('Time (s)')

pylab.show()

#Calculate SnR Peak

peak = abs(snr).numpy().argmax()

snrp = snr[peak]

time = snr.sample\_times[peak]

print("We found a signal at {}s with SNR {}".format(time, abs(snrp)))

# The time, amplitude, and phase of the SNR peak tell us how to align

# our proposed signal with the data.

# Shift the template to the peak time

dt = time - conditioned.start\_time

aligned = template.cyclic\_time\_shift(dt)

# scale the template so that it would have SNR 1 in this data

aligned /= sigma(aligned, psd=psd, low\_frequency\_cutoff=20.0)

# Scale the template amplitude and phase to the peak value

aligned = (aligned.to\_frequencyseries() \* snrp).to\_timeseries()

aligned.start\_time = conditioned.start\_time

#Whiten the data

# We do it this way so that we can whiten both the template and the data

white\_data = (conditioned.to\_frequencyseries() / psd\*\*0.5).to\_timeseries()

white\_template = (aligned.to\_frequencyseries() / psd\*\*0.5).to\_timeseries()

white\_data = white\_data.highpass\_fir(30., 512).lowpass\_fir(300, 512)

white\_template = white\_template.highpass\_fir(30, 512).lowpass\_fir(300, 512)

# Select the time around the merger

mergertime = time

white\_data = white\_data.time\_slice(mergertime-.2, mergertime+.1)

white\_template = white\_template.time\_slice(mergertime-.2, mergertime+.1)

pylab.figure(figsize=[15, 3])

pylab.plot(white\_data.sample\_times, white\_data, label="Data")

pylab.plot(white\_template.sample\_times, white\_template, label="Template")

pylab.legend()

pylab.show()

subtracted = conditioned - aligned

# Plot the original data and the subtracted signal data

#Qtransform white data and plot the graph

#Qtransform white data and plot the graph[

# Plot a spectrogram (or q-transform) of the data, and try to identify the signal

for data, title in [(conditioned, 'Original H1 Data'),

                    (subtracted, 'Signal Subtracted from H1 Data')]:

    t, f, p = data.whiten(4, 4).qtransform(.001, logfsteps=100, qrange=(8, 8), frange=(20, 512))

    pylab.figure(figsize=[15, 3])

    pylab.title(title)

    pylab.pcolormesh(t, f, p\*\*0.5, vmin=1, vmax=6, shading='auto')

    pylab.yscale('log')

    pylab.xlabel('Time (s)')

    pylab.ylabel('Frequency (Hz)')

    #pylab.xlim(mergertime - 2, mergertime + 1)

    pylab.show()

[-3.04517369e-19 -3.35465069e-19 -3.35210566e-19 ... 4.46471052e-19

5.07624678e-19 4.98623016e-19]

[0.00000000e+00 2.44379277e-04 4.88758553e-04 ... 1.02399951e+03

1.02399976e+03 1.02400000e+03]

Chart, histogram

Description automatically generated

Chart, histogram

Description automatically generated

Shape, rectangle

Description automatically generated

Rectangle

Description automatically generated with medium confidence

Chart, histogram

Description automatically generated

We found a signal at 2483.96728515625s with SNR 28.875814057421188

A picture containing arrow

Description automatically generated