___________________________Reading the Data______________________________

```
%matplotlib inline
import pylab
from pycbc.filter import highpass
from pycbc.catalog import Merger
from pycbc.frame import read_frame

merger = Merger("GW170817")
strain, stilde = {}, {}
for ifo in ['H1', 'L1']:
   # We'll download the data and select 256 seconds that includes the
event time
   ts =
read_frame("{}-{}_LOSC_CLN_4_V1-1187007040-2048.gwf".format(ifo[0], ifo),
                    '{}:LOSC-STRAIN'.format(ifo),
                   start_time=merger.time - 224,
                   end_time=merger.time + 32,
                   check_integrity=False)

   # Read the detector data and remove low frequency content
   strain[ifo] = highpass(ts, 15)

   # Remove time corrupted by the high pass filter
   strain[ifo] = strain[ifo].crop(4, 4)

   # Also create a frequency domain version of the data
   stilde[ifo] = strain[ifo].to_frequencyseries()


#print (strain.delta_t)
pylab.plot(strain['H1'].sample_times, strain['H1'])
pylab.xlabel('Time (s)')
pylab.show()
```

#---------------------------Estimate PSD from your data---

```
from pycbc.psd import interpolate, inverse_spectrum_truncation

psds = {}
for ifo in ['L1', 'H1']:
```

```
    # Calculate a psd from the data. We'll use 2s segments in a median -
welch style estimate
    # We then interpolate the PSD to the desired frequency step.
    psds[ifo] = interpolate(strain[ifo].psd(2), stilde[ifo].delta_f)

    # We explicitly control how much data will be corrupted by
overwhitening the data later on
    # In this case we choose 2 seconds.
    psds[ifo] = inverse_spectrum_truncation(psds[ifo], int(2 *
strain[ifo].sample_rate),
                                    low_frequency_cutoff=15.0,
                                    trunc_method='hann')

    pylab.loglog(psds[ifo].sample_frequencies, psds[ifo], label=ifo)
    pylab.xlim(20, 1024)
    pylab.ylim(1e-47, 1e-42)
pylab.legend()



##########_______________Matched filtering_____________________

from pycbc.waveform import get_fd_waveform
from pycbc.filter import matched_filter
from pycbc.conversions import mass1_from_mchirp_q
import numpy

# We will try different component masses and see which gives us the
largest
masses = numpy.arange(1.3, 1.5, .01)

# Variables to store when we've found the max
hmax, smax, tmax, mmax, nsnr = None, {}, {}, 0, 0
snrs = []

for m in masses:
    #Generate a waveform with a given component mass; assumed equal mass,
nonspinning
    hp, hc = get_fd_waveform(approximant="TaylorF2",
                             mass1=m, mass2=m,
                             f_lower=20, delta_f=stilde[ifo].delta_f)
```

```
    hp.resize(len(stilde[ifo]))

    # Matched filter the data and find the peak
    max_snr, max_time = {}, {}
    for ifo in ['L1', 'H1']:
        snr = matched_filter(hp, stilde[ifo], psd=psds[ifo],
low_frequency_cutoff=20.0)

        # The complex SNR at the peak
        snr = snr.time_slice(merger.time - 1, merger.time + 1)
        _, idx = snr.abs_max_loc()
        max_snr[ifo] = snr[idx]

        # The time of the peak
        max_time[ifo] = float(idx) / snr.sample_rate + snr.start_time

    network_snr = (abs(numpy.array(list(max_snr.values()))) ** 2.0).sum()
** 0.5
    snrs.append(max_snr)

    # Keep track of only the loudest peak
    if network_snr > nsnr:
        tmax, hmax, mmax, smax = max_time, hp, m, max_snr
        nsnr = network_snr

# See the SNR as a function of the component mass. Notice where this peaks
as it gives us
# an estimate of what the parameters of the source system are. Note that
masses
# here are in the *detector* frame, so if the source is located far away,
it will in
# fact correspond to a lighter system due to cosmological redshift.
print("We found the best Mass1=Mass2 was %2.2f solar masses (detector
frame)" % mmax)
#______________________________________________________________________
```