# Effect of Low Frequency Noise in Detection of Gravitational Waves

Baisakhi Mitra

Punjabi University, Patiala, India

(Dated: March 2019)

## Abstract

Gravitational wave was first detected on September 14, 2015. In this GW150914 event, frequency at peak Gravitational wave strain was 150 Hz. In this research, 200 datasets were chosen from LIGO's O1 data release. By doing software injection and recovery in these datasets, using GW150914 template, the effect of low frequency (<100 Hz) noise power on Signal to Noise Ratio of detected Gravitational wave in GW150914 event, has been studied in this research.

## I. BACKGROUND

In 1916, Albert Einstein first predicted the existence of Gravitational Waves. Contradicting Newton's theory of Gravity, which manifests Gravity as a force between two masses, Einstein suggested that the Universe is made of stretched fabric of space and time. The curvature in this space time fabric is caused by presence of mass. Disturbances in this space time fabric, caused by various ways, propagates outwards with the speed of light in vacuum, is known as Gravitational waves. There are various sources of Gravitational Wave: Coalescing binary systems (Black Holes, Neutron Stars), Transient Burst Sources like Supernova, Continuous sources like spinning Neutron Stars and Stochastic noise background like residue of Big Bang. In September 2015, LIGO detected Gravitational wave from the first kind of source, i.e. merging of two Black Holes. This event is named as GW150914.

The two polarizations of the Gravitational wave, produced by compact binary inspiral, exhibit a monotonically increasing frequency and amplitude as the orbital motion radiate away energy and decays. The waveform called a chirp waveform. There are three phases in Compact Binary Coalesce (CBC) waveform: Inspiral, merger and ringdown. As the process proceeds from inspiral to merger, the amplitude and frequency of Gravitational wave increases and reaches maximum value at end of merger phase. For GW150914, the peak Gravitational wave strain was $10^{-21}$ and frequency at peak Gravitational wave strain was 150 Hz.

In Laser Interferometer Gravitational Wave Observatory, when a signal from a distant Gravitational wave source reaches, then Gravitational wave signal superimposed on existing noise in interferometer channel. Since, noise in interferometer arm is not of a particular frequency; therefore noise spectrum covers a wide range of frequency. Among that, low frequency noise is mainly contributed by seismic activities. Therefore, Signal to Noise Ratio (SNR) of detected Gravitational wave signal decreases, if noise power increases. Even though, Gravitational wave peak strain corresponds to 150 Hz, but due to wide

noise spectrum, SNR decreases with increase in noise power in different frequency ranges.

In this paper, nature of change of SNR with change in noise power in low frequency domain (<100 Hz) has been studied.

## II. ANALYSIS

The object of this research is to study, the effect of low frequency noise in the detection of Gravitational Wave.

The first step is to find time segments [5] in O1 data release [1] of LIGO, which passes data category check. Nearly 200 data sets are downloaded, which satisfy the above criteria, from both Hanford and Livingston datasets.

In the next step, hardware injection check was done, and datasets which have such injections were removed. Next, software injection was done using GW150914 template and then recovery is done, using matched filtering technique [3].

The formula used for matched filter recovery of s(t) is:[4]

$$x(t_0) = 2\int_{-\infty}^{\infty} \frac{\tilde{s}(f)\tilde{h}^*_{\text{template}}(f)}{S_n(f)}\,df$$

Using the above formula Signal to Noise Ratio is calculated for matched filter recovery. Then, the above mentioned steps are repeated for different amplifications of injected signal.

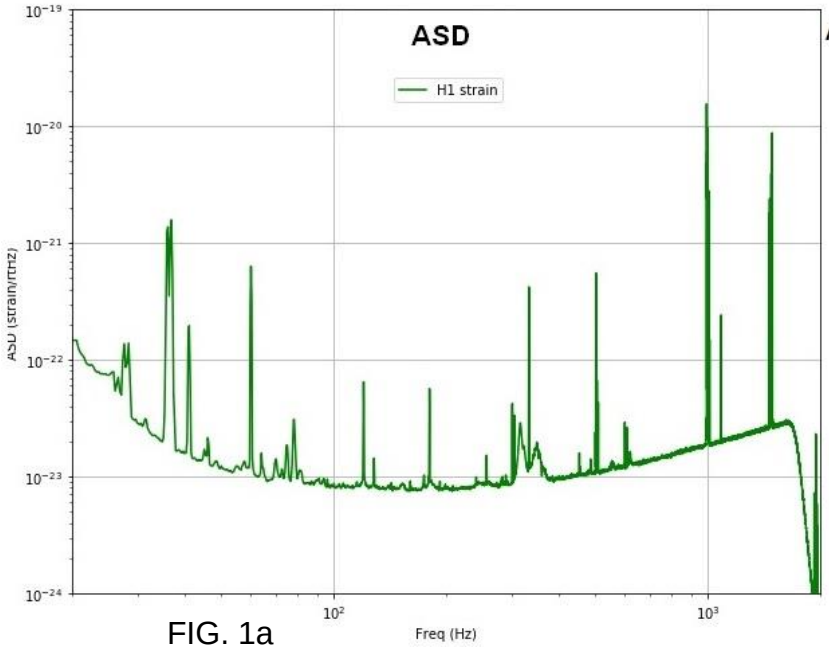
After that, Amplitude spectral density of all those 200 datasets was plotted [2], using code mentioned in Appendix.

In the following step, a band pass filter (20 Hz< f <100 Hz) is used followed by Fast Fourier Transform, to calculate noise power in those 200 dataset. This low frequency noise power is plotted against SNR of the respective dataset, calculated in previous step.

## III. RESULT

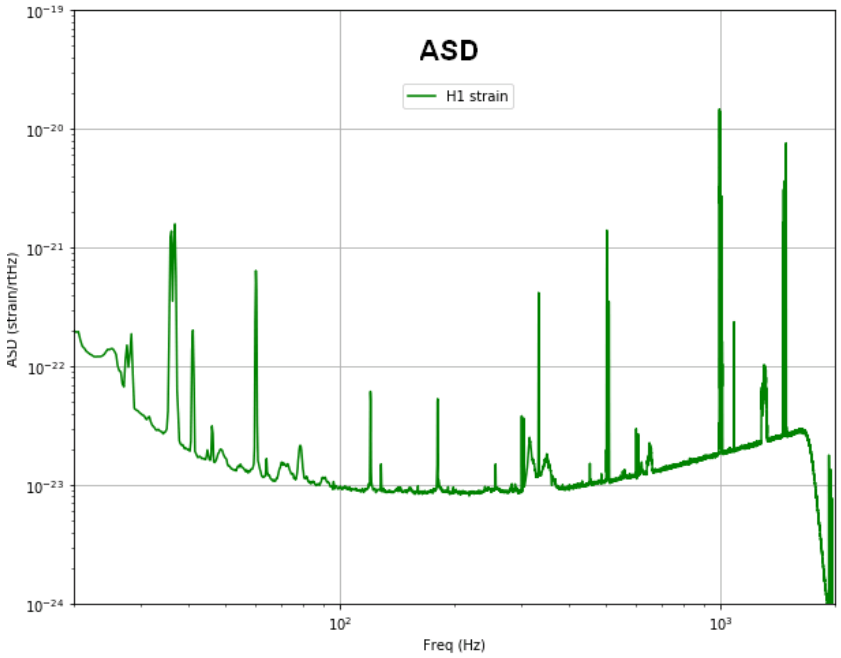
The comparison of Amplitude spectral density plot of a dataset in O1 run and SNR obtained by software injection of GW150914 followed by matched filter recovery, in the same dataset reveals below mentioned results:

A. If noise in low frequency domain increases, then SNR decreases. SNR corresponding to zero amplification of injected signal is less affected by low frequency noise, whereas SNR at higher amplification is more affected by low frequency noise.




| Amplification | SNR |
|---|---|
| 0 | 4.802675889502484 |
| 1 | 22.925850029152176 |
| 2 | 46.01429203216445 |
| 4 | 92.22812049228199 |
| 6 | 138.44446573766078 |
| 8 | 184.6614473767156 |
| 10 | 230.87868476369658 |
| 12 | 277.09605035977756 |
| 14 | 323.313489342244 |
| 16 | 369.5309742457363 |
| 18 | 415.74848979027047 |
| 20 | 461.9660267981647 |
| 22 | 508.18357942425683 |
| 24 | 554.4011437691885 |
| 26 | 600.6187171319342 |
| 28 | 646.8362975823104 |
| 30 | 693.0538837043026 |

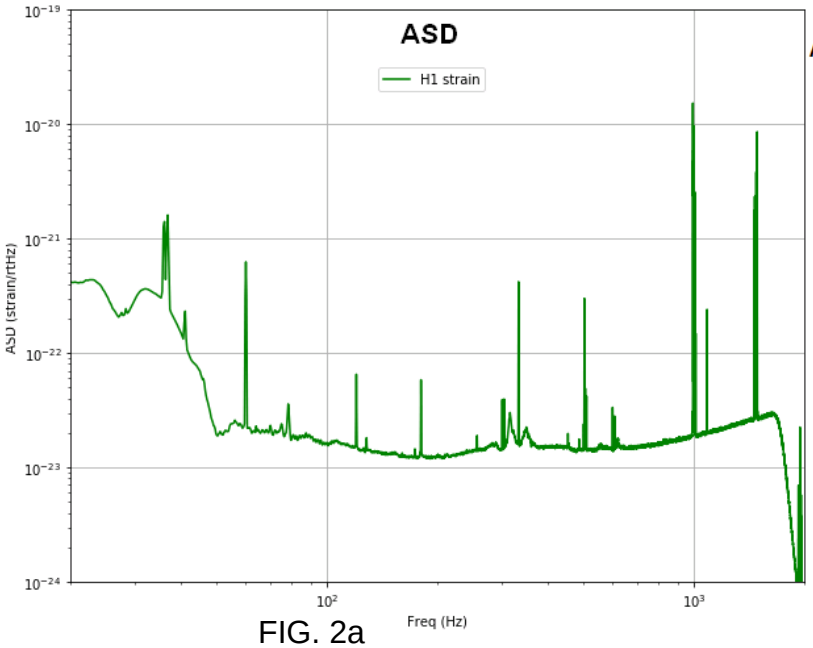
FIG. 1a




| Amplification | SNR |
|---|---|
| 0 | 4.333533661045784 |
| 1 | 21.08492700531476 |
| 2 | 41.99862103804403 |
| 4 | 83.86943028846515 |
| 6 | 125.7506338085306 |
| 8 | 167.64456024555483 |
| 10 | 209.53891490722066 |
| 12 | 251.43348417870942 |
| 14 | 293.3281762675153 |
| 16 | 335.22294519737153 |
| 18 | 377.11776539425045 |
| 20 | 419.01262149947445 |
| 22 | 460.907503732262 |
| 24 | 502.80240556830586 |
| 26 | 544.6973224886251 |
| 28 | 586.5922512640916 |
| 30 | 628.4871895258782 |

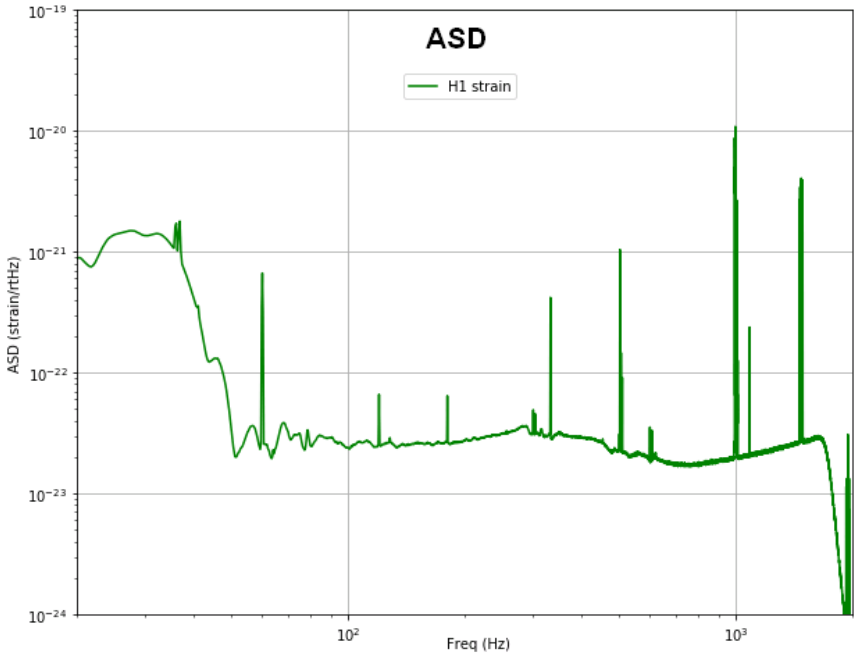
FIG. 1b

FIG. 1a and 1b: ASD plot and SNR of a dataset, where noise power is less.




| Amplification | SNR |
|---|---|
| 0 | 2.713694108486748 |
| 1 | 12.973964320766848 |
| 2 | 26.329273720685926 |
| 4 | 53.062193416873924 |
| 6 | 79.80792120848895 |
| 8 | 106.5538333181461 |
| 10 | 133.29981874992677 |
| 12 | 160.04584073120773 |
| 14 | 186.79188355713615 |
| 16 | 213.53793939318462 |
| 18 | 240.28400389388426 |
| 20 | 267.03007445511594 |
| 22 | 293.7761494212783 |
| 24 | 320.52222768947274 |
| 26 | 347.2683084966282 |
| 28 | 374.0143912979764 |
| 30 | 400.7604756941968 |

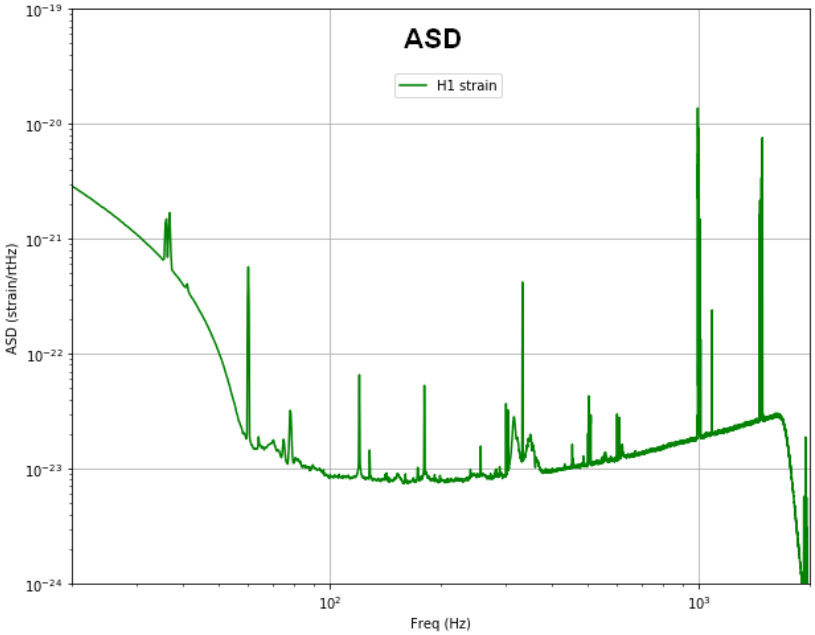
FIG. 2a




| Amplification | SNR |
|---|---|
| 0 | 2.3661389805639566 |
| 1 | 9.86729545982236 |
| 2 | 19.53965294589272 |
| 4 | 38.91138036630654 |
| 6 | 58.284925937377736 |
| 8 | 77.6588616148687 |
| 10 | 97.03295330702773 |
| 12 | 116.40712299926872 |
| 14 | 135.78133726042307 |
| 16 | 155.1555793760566 |
| 18 | 174.52984006081374 |
| 20 | 193.90411374367582 |
| 22 | 213.27839687954398 |
| 24 | 232.65268710506908 |
| 26 | 252.02698278411464 |
| 28 | 271.4012827480283 |
| 30 | 290.77558613980864 |

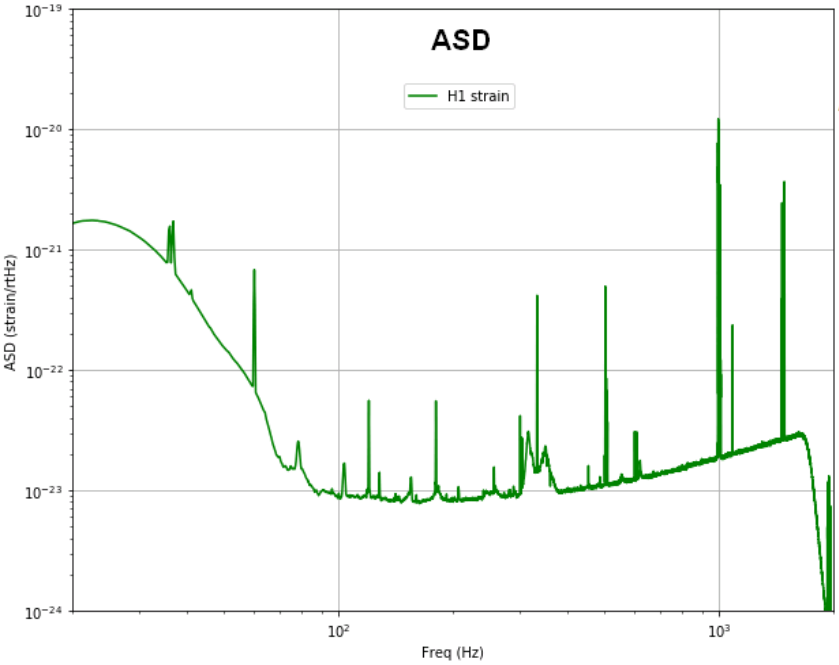
FIG. 2b

FIG. 2a and 2b:. High Noise level is high as indicated in ASD plot, therefore SNR for both higher and lower amplification is low.




| Amplification | SNR |
|---|---|
| 0 | 4.461259371017929 |
| 1 | 18.661008687797555 |
| 2 | 36.62027175278105 |
| 4 | 72.57792537611104 |
| 6 | 108.54948882632011 |
| 8 | 144.52182546999748 |
| 10 | 180.49447211390003 |
| 12 | 216.46727395892094 |
| 14 | 252.44016456418026 |
| 16 | 288.4131106769263 |
| 18 | 324.3860938106372 |
| 20 | 360.3591028676515 |
| 22 | 396.33213078296933 |
| 24 | 432.30517284506624 |
| 26 | 468.27822579125075 |
| 28 | 504.2512872905119 |
| 30 | 540.2243556331199 |

FIG. 3a

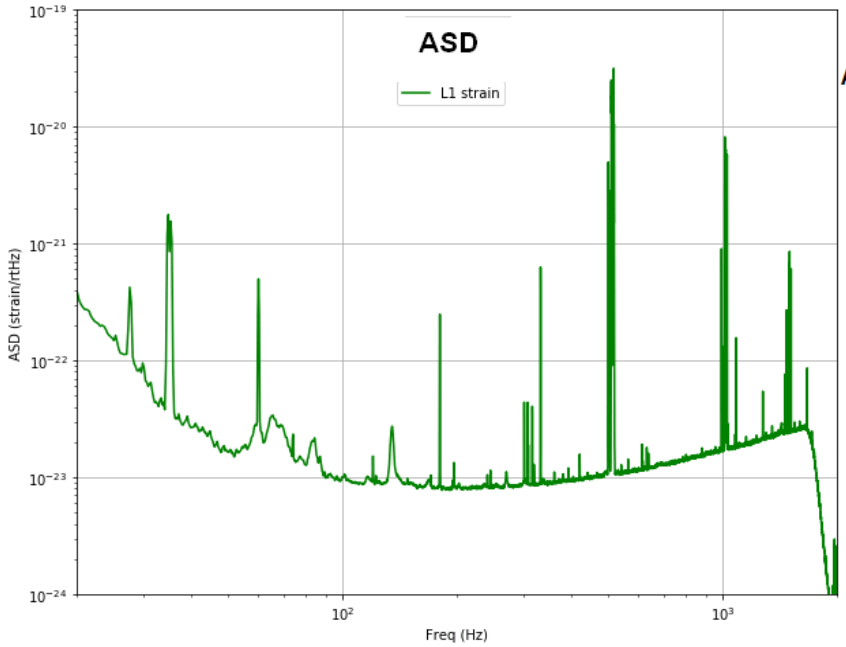



| Amplification | SNR |
|---|---|
| 0 | 3.5145708855968167 |
| 1 | 14.905008929201895 |
| 2 | 29.55216825972355 |
| 4 | 58.86981207488518 |
| 6 | 88.1899435747517 |
| 8 | 117.51069831795628 |
| 10 | 146.8317025453868 |
| 12 | 176.152831563995 |
| 14 | 205.4740319092739 |
| 16 | 234.7952768411746 |
| 18 | 264.11655150111 |
| 20 | 293.43784697260014 |
| 22 | 322.75915758087234 |
| 24 | 352.0804795424016 |
| 26 | 381.401810237631 |
| 28 | 410.7231477953343 |
| 30 | 440.04449084320777 |

FIG. 3b

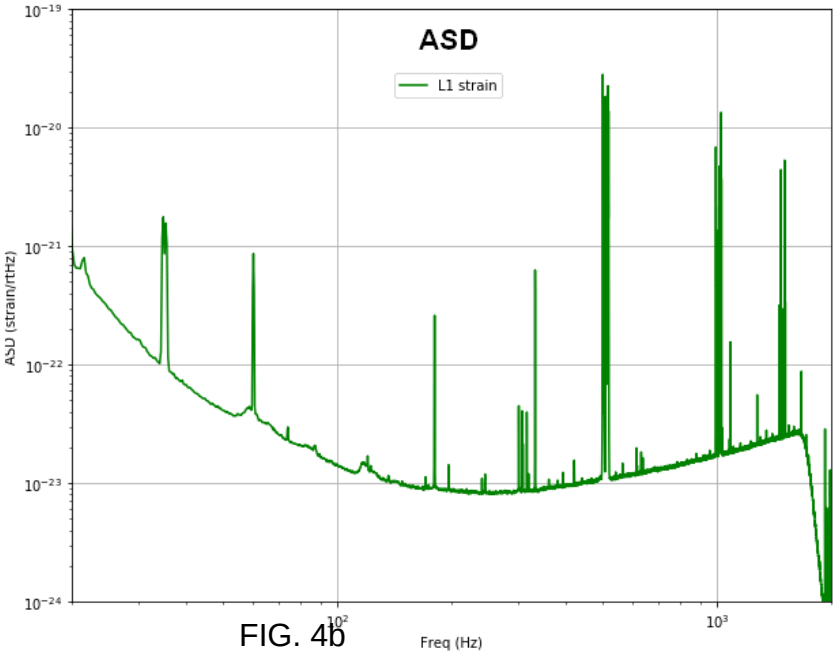
FIG. 3a and 3b: Noise at low frequency is high, but at frequency between 100 to 1000 Hz is less. Hence, SNR at zero amplification is less affected. But, at higher amplification is more affected.

B. Noise near 150 Hz is more in Livingston data than Hanford data, therefore for similar scenarios, SNR for Livingston data is less than Hanford. This change is more visible at higher amplification.



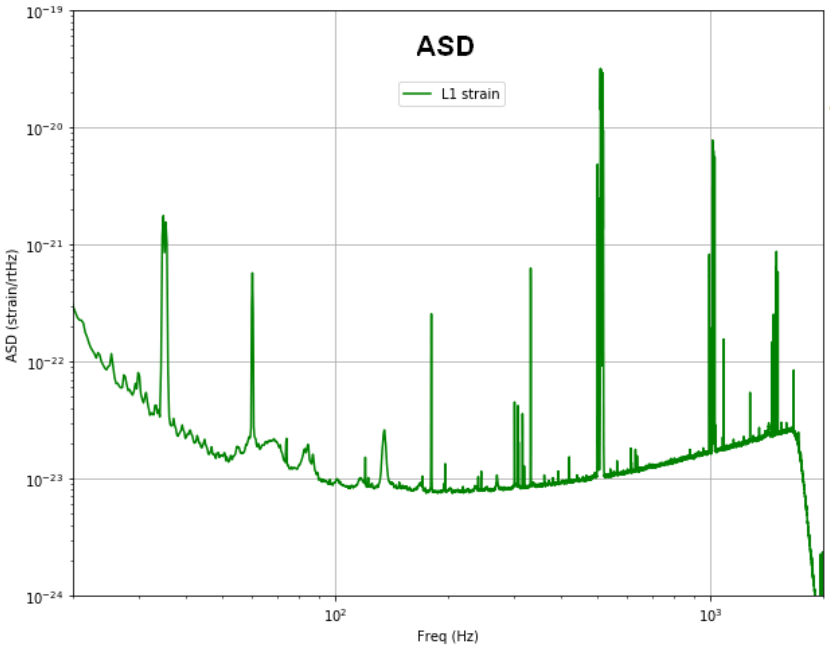

| Amplification | SNR |
|---|---|
| 0 | 4.974009992973795 |
| 1 | 18.123620637916584 |
| 2 | 36.263556222281565 |
| 4 | 72.56368983455324 |
| 6 | 108.86658502502019 |
| 8 | 145.18463019232385 |
| 10 | 181.50607260413562 |
| 12 | 217.82764551333247 |
| 14 | 254.14929304184707 |
| 16 | 290.47098722899625 |
| 18 | 326.7927125325321 |
| 20 | 363.1144596232725 |
| 22 | 399.4362225625665 |
| 24 | 435.75799739030197 |
| 26 | 472.07978136428176 |
| 28 | 508.4015725254567 |
| 30 | 544.7233694369712 |

FIG. 4a



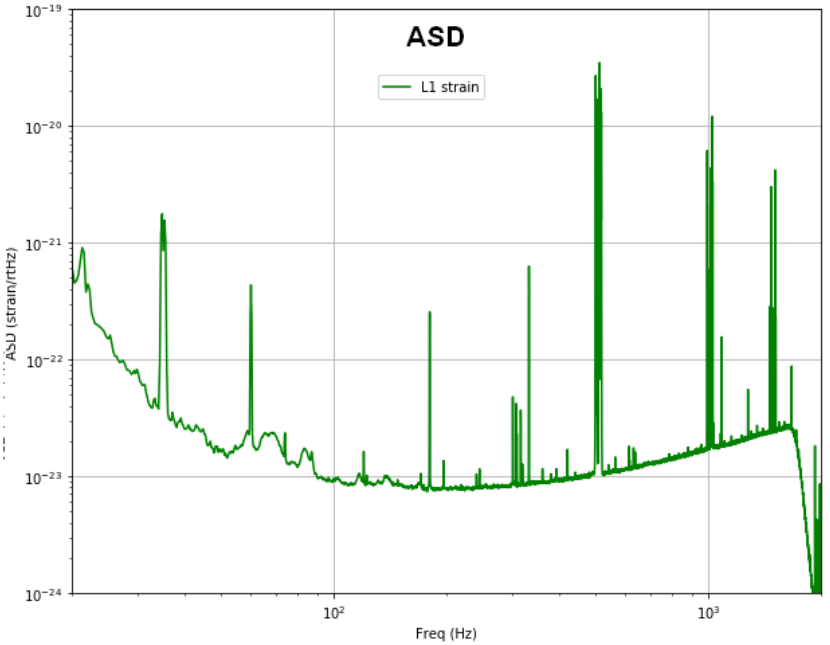

| Amplification | SNR |
|---|---|
| 0 | 4.042144946564852 |
| 1 | 16.711476395913184 |
| 2 | 32.924440429304475 |
| 4 | 65.36358745057387 |
| 6 | 97.80567555930922 |
| 8 | 130.2486887864544 |
| 10 | 162.69209126445048 |
| 12 | 195.13564143582462 |
| 14 | 227.5792760346539 |
| 16 | 260.02296341482577 |
| 18 | 292.4666859897482 |
| 20 | 324.9104332049534 |
| 22 | 357.354198342687 |
| 24 | 389.79797692375234 |
| 26 | 422.24176584676945 |
| 28 | 454.6855628962254 |
| 30 | 487.1293664472597 |

FIG. 4b




| Amplification | SNR |
|---|---|
| 0 | 4.515073349205083 |
| 1 | 19.57828748131291 |
| 2 | 38.98990272932833 |
| 4 | 77.8620694958857 |
| 6 | 116.73889254950146 |
| 8 | 155.6168857170733 |
| 10 | 194.49534791052275 |
| 12 | 233.37404489072816 |
| 14 | 272.2528761356936 |
| 16 | 311.13179134036454 |
| 18 | 350.01076254004823 |
| 20 | 388.8897729480645 |
| 22 | 427.76881187807766 |
| 24 | 466.6478722037755 |
| 26 | 505.5269489903697 |
| 28 | 544.4060387123011 |
| 30 | 583.2851387837184 |

FIG. 4c




| Amplification | SNR |
|---|---|
| 0 | 4.28271077196275 |
| 1 | 18.76930119152874 |
| 2 | 37.5803811479089 |
| 4 | 75.21770131022384 |
| 6 | 112.85726896616261 |
| 8 | 150.49739739914486 |
| 10 | 188.13787322494304 |
| 12 | 225.77900134321027 |
| 14 | 263.42019642140474 |
| 16 | 301.0614333379799 |
| 18 | 338.7026981411193 |
| 20 | 376.3439824617935 |
| 22 | 413.9852809752727 |
| 24 | 451.6265901322778 |
| 26 | 489.2679074759245 |
| 28 | 526.909231251478 |
| 30 | 564.5505601722459 |

FIG. 4d

Fig 4a,b,c,d:. ASD plot of L1 dataset shows noise around 150 Hz is more than H1. Therefore SNR is less than H1, and this is more visible in higher amplification.

C. The integrated plot for few selected datasets among 200 datasets shows that, SNR decreases with increase in noise power in low frequency regime.

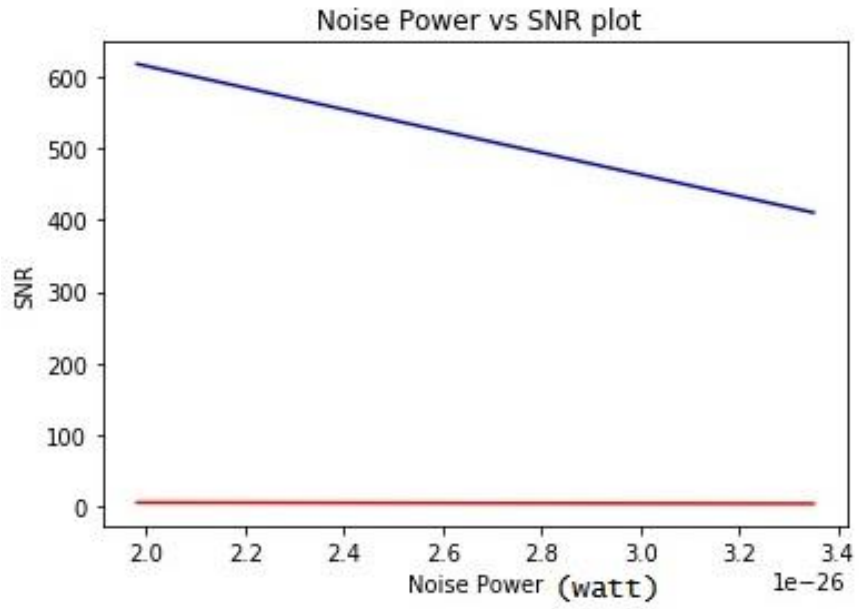



FIG. 5: Low frequency noise power vs. SNR plot (Red: 0 amplification, Blue: 30 amplification)

D. ASD plot and SNR for 200 datasets is present in [6].

## IV. CONCLUSION

In this project, software injection has been done, using GW150914 template, on 200 dataset of LIGO O1 run. Then by matched filtering SNR has been calculated. Comparison of this SNR with Amplitude spectral density of noise in those 200 dataset shows that, SNR decrease with increase in noise power in low frequency regime, i.e below 100 Hz.

## V. ACKNOWLEDGEMENT

This research has made use of data, software and/or web tools obtained from the Gravitational Wave Open Science Center (https://www.gw-openscience.org), a service of LIGO Laboratory, the LIGO Scientific Collaboration and the Virgo Collaboration. LIGO is funded by the U.S. National Science Foundation. Virgo is funded by the French Centre National de Recherche Scientifique (CNRS), the Italian Istituto Nazionale della Fisica Nucleare (INFN) and the Dutch Nikhef, with contributions by Polish and Hungarian institutes.

## VI. REFERENCES

[1] https://www.gw-openscience.org/O1/

[2] https://www.gwopenscience.org/s/events/GW150914/

LOSC_Event_tutorial_GW150914.html#Plot-the-Amplitude-Spectral-Density-(ASD)

[3] https://www.gw-openscience.org/tutorials/

[4] B. P. Abbott et al. (2016), arXiv:1606.04856

[5] LIGO CalTech, *Ligo timeline*, URL https://www.ligo.caltech.edu/page/timeline.

[6] https://jumpshare.com/b/AUAmy5E1GxcbqM4oLyKW

[7] https://www.gw-openscience.org/static/papers/Tony_paper4.pdf

[8] M Vallisneri et al. "The LIGO Open Science Center", proceedings of the 10th LISA Symposium, University of Florida, Gainesville, May 18-23, 2014

[9] arxiv:1410.4839

## VII. APPENDIX

1. Python script to plot amplitude spectral density:

```
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import scipy.signal as sig
```

```
import readligo as rl
import numpy as np
from random import randint
from scipy import signal
from scipy.interpolate import interp1d
from scipy.signal import butter, filtfilt, iirdesign, zpk2tf, freqz
import h5py
import json
fs = 4096
fileName = 'L-L1_LOSC_4_V1-1126354944-4096'
f = open(fileName + '.hdf5', 'r')
dataFile = h5py.File(fileName + '.hdf5', 'r')
strain = dataFile['strain/Strain']
dqInj = dataFile['quality/injections/Injmask']
make_psds = 1
f.close()
if make_psds:
# number of sample for the fast fourier transform:
    NFFT = 4*fs

    Pxx_L1, freqs = mlab.psd(strain, Fs = fs, NFFT = NFFT)
    psd_L1 = interp1d(freqs, Pxx_L1)
    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 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.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)
dataFile.close()
```

2. Python script to do software injection in O1 datasets and recovery using matched filtering and hence calculation of SNR[7] :

```
import h5py
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import scipy.signal as sig
import readligo as rl
from random import randint
#-- Sampling rate equals to 4096 Hz
fs = 4096
#-- Read in the waveform/
temp_time, temp_strain =
np.genfromtxt('GW150914_4_NR_waveform.txt').transpose()
#-- Read in the template
f_template = h5py.File('GW150914_4_template.hdf5', 'r')
template_p, template_c = f_template['template'].value
template = template_p + template_c * 1.j
# to remove effects at the beginning and end of the data stretch, window the data
# https://en.wikipedia.org/wiki/Window_function#Tukey_window
try: dwindow2 = sig.tukey(template.size, alpha=1./8) # Tukey window preferred,
but requires
#recent scipy version
except: dwindow2 = sig.blackman(template.size) # Blackman window OK if
Tukey is not
#available
# the length and sampling rate of the template MUST match that of the data.
datafreq = np.fft.fftfreq(template.size)*fs
df = np.abs(datafreq[1] - datafreq[0])
# prepare the template fft.
template_fft = np.fft.fft(template*dwindow2) / fs
# -- To calculate the PSD of the data, choose an overlap and a window (common
to all detectors)
# that minimizes \enquote{spectral leakage}
https://en.wikipedia.org/wiki/Spectral_leakage
NFFT = 4*fs
psd_window = np.blackman(NFFT)
# and a 50% overlap:
NOVL = NFFT/2
#-- Set the amplifications of signal to be injected
A=[]
A = [0,1,2] + list(range(4,31,2))
#A = [0,1,2]
#-- Define the function to segment the segments into suitable segments
def slice_filter(dq, hw):
#-- Cut the whole segment in to small ones of 60s, with spacing of 1s
```

```
    i = 0
    while i < len(dq):
        if dq[i].stop - dq[i].start < 32*fs:
            del dq[i]
            continue
        dq.insert(i, slice(dq[i].start, dq[i].start + 32*fs))
        dq[i+1] = slice(dq[i].stop + 1*fs, dq[i+1].stop)
        i += 1
#-- Delete the segments with injection activatedz
    i = 0
    j = 0
    if len(hw) == 0:
        return dq
    while i < len(dq) and j < len(hw):
#-- dq is 100% before hw
        if dq[i].stop < hw[j].start:
            i += 1
#-- dq is 100% after hw
        elif dq[i].start > hw[j].stop:
            j += 1
#-- dq and hw overlap
        else:
            del dq[i]
    return dq
#-- Prepare the files to work on
fileName = []
fileName.append('L-L1_LOSC_4_V1-1126801408-4096')
fileName.append('L-L1_LOSC_4_V1-1126354944-4096')
fileName.append('L-L1_LOSC_4_V1-1127002112-4096')
fileName.append('H-H1_LOSC_4_V1-1127260160-4096')
fileName.append('H-H1_LOSC_4_V1-1127530496-4096')
fileName.append('H-H1_LOSC_4_V1-1126780928-4096')
fileName.append('H-H1_LOSC_4_V1-1126105088-4096')
fileName.append('H-H1_LOSC_4_V1-1126088704-4096')
….

#============================#
#========Main Part==========#
#============================#
for name in fileName:
#-- Prepare the document to record the table (SNR vs Ampli.)
    f = open(name + '.txt', 'w')
#-- Load data from the original file
    rawFile = h5py.File(name + '.hdf5', 'r')
```

```
    strain_raw = rawFile['strain/Strain'].value
    CBC_CAT3 = rawFile['quality/simple/DQmask'].value
    CBC_CAT3 = (rawFile['quality/simple/DQmask'].value >> 4) & 1
    if all(v == 0 for v in (rawFile['quality/simple/DQmask'].value)):
        NO_CBC_HW_INJ = (rawFile['quality/injections/Injmask'].value) or 1
    else:
        NO_CBC_HW_INJ = (rawFile['quality/injections/Injmask'].value ) & 0


    GPSstart = rawFile['meta/GPSstart'].value
    rawFile.close()
#-- Load the file again which will be injected
    dataFile = h5py.File(name + '.hdf5', 'r+')
    strain = dataFile['strain/Strain']
    dqInj = dataFile['quality/injections/Injmask']
#-- Getting the suitable segement lists
    segList = rl.dq_channel_to_seglist(CBC_CAT3)
    segList_HW = rl.dq_channel_to_seglist(NO_CBC_HW_INJ)
    segList = slice_filter(segList, segList_HW)

#-- Pick 10 (or if not more than 10, all) segments randomly
    inj_num = []
    if len(segList) <= 5:
        inj_num = range(0, len(segList))
#-- If less than 5, throw a caution
        print ('Caution: there are less than five suitable spots in %s!' % name)
        print ('{0} only!'.format(len(segList)))
    else:
        while len(inj_num) < 5:
            i = randint(0, len(segList)-1)
            if i in inj_num:
                continue
            inj_num.append(i)
    inj_num.sort()
    List = []
    for i in inj_num:
        List.append(segList[i])
#-- Make and recover injections at each amplitude
    for k in range(0, len(A)):
        f.write('{0} '.format(A[k]))
#-- Amplify the template with the difference to next amplification
        if k == 0:
            temp = temp_strain * (A[k])
        else:
```

```
        print(A[k],A[k-1])
        temp = temp_strain * (A[k] - A[k-1])
#-- Make the injection to every piece
      SNRsum = 0
      for seg in List:
#-- Injection starts at the middle
        inj_sample = seg.start + 16*fs
#-- Superpose the waveform to the signal
        for j in range(0, temp.size):
          strain[inj_sample + j] += temp[j]
#-- Injection Recovery --#
#using the segments in the above section
        data = strain[seg]
# to remove effects at the beginning and end of the data stretch, window the data
# https://en.wikipedia.org/wiki/Window_function#Tukey_window
        try: dwindow = sig.tukey(data.size, alpha=1./8) # Tukey window preferred,
but
#requires recent scipy version
        except: dwindow = sig.blackman(data.size) # Blackman window OK if
Tukey is
#not available
#-- Calculate the PSD of the data. Also use an overlap, and window:
# This is where the only change is made, I replaced the PSD of the 32-s segment
with
# that of the whole file, because when the file is small the PSD will be greatly
#affected by the
# signal in the file, whereas we only want the PSD of the background noise. So
taking
#the PSD
# of a much larger interval can help eliminating the effect.
        data_psd, freqs = mlab.psd(strain_raw, Fs = fs, NFFT = NFFT,
                        window=psd_window,noverlap=NOVL)
# Take the Fourier Transform (FFT) of the data and the template (with dwindow)
        data_fft = np.fft.fft(data*dwindow) / fs
#-- Interpolate to get the PSD values at the needed frequencies
        power_vec = np.interp(np.abs(datafreq), freqs, data_psd)
#-- Calculate the matched filter output in the time domain:
# Multiply the Fourier Space template and data, and divide by the noise power in
each
#frequency bin.
# Taking the Inverse Fourier Transform (IFFT) of the filter output puts it back in
#the time domain,
# so the result will be plotted as a function of time off-set between the template
#and the data:
```

```
            optimal = data_fft * template_fft.conjugate() / power_vec
            optimal_time = 2*np.fft.ifft(optimal)*fs
    #-- Normalize the matched filter output:
    # Normalize the matched filter output so that we expect a value of 1 at times of
    just
    #noise.
    # Then, the peak of the matched filter output will tell us the signal-to-noise ratio
    #(SNR) of the signal.
            sigmasq = 1*(template_fft * template_fft.conjugate() /
                    power_vec).sum() * df
            sigma = np.sqrt(np.abs(sigmasq))
            SNR_complex = optimal_time/sigma
    # shift the SNR vector by the template length so that the peak is at the END of
    the
    #template
            peaksample = int(data.size / 2) # location of peak in the template
            SNR_complex = np.roll(SNR_complex,peaksample)
            SNR = abs(SNR_complex)
    #-- Find the time and SNR value at maximum:
            indmax = np.argmax(SNR)
            SNRmax = SNR[indmax]
            SNRsum += SNRmax
        SNR_avr = SNRsum / len(inj_num)
        f.write('{0}\n'.format(SNR_avr))
        dataFile.flush()
        print ('%s at amplification = {0} finished.'.format(A[k]) % name)
    dataFile.close()
    f.close()
```

3. Python script to calculate power in low frequency region of selected dataset:

```
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import scipy.signal as sig
import readligo as rl
from random import randint
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
import re
```

```
# 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
amp = []
npow = []
fileName = []
fileName.append('H-H1_LOSC_4_V1-1126076416-4096')
fileName.append('H-H1_LOSC_4_V1-1126080512-4096')
fileName.append('H-H1_LOSC_4_V1-1126084608-4096')


for name in fileName:
    rawFile = h5py.File(name + '.hdf5', 'r')
    strain_raw = rawFile['strain/Strain'].value
    (B,D) = sig.butter(4, [20/(4096/2.0), 100/(4096/2.0)], btype='pass')
    strain_p= sig.lfilter(B, D, strain_raw)
    strain_fft = np.fft.fft(strain_raw)
    strain_pow = np.square(np.absolute(strain_fft))
    power = sum(strain_pow)
    rawFile.close()
    f = open(name + '-lowpow' + '.txt', 'w')
    f.write('{0}\n'.format(power))
    f.close()
```

4. Python script to plot Low frequency noise power vs SNR:

```
import scipy.signal as sig
import readligo as rl
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
from random import randint
from scipy import signal
from scipy.interpolate import interp1d
from scipy.signal import butter, filtfilt, iirdesign, zpk2tf, freqz
import h5py
import json
```

```
import re
# 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
amp = []

fileName = []
fileName.append('H-H1_LOSC_4_V1-1126080512-4096')
fileName.append('H-H1_LOSC_4_V1-1126256640-4096')
fileName.append('H-H1_LOSC_4_V1-1126309888-4096')
fileName.append('H-H1_LOSC_4_V1-1126318080-4096')
fileName.append('H-H1_LOSC_4_V1-1126404096-4096')
fileName.append('H-H1_LOSC_4_V1-1126432768-4096')



snrav0 = []
snrav1 = []
snrav2 = []
snrav4 = []
snrav6 = []
snrav8 = []
snrav10 = []
snrav12= []
snrav14= []
snrav16= []
snrav18 = []
snrav20 = []
snrav22 = []
snrav24 = []
snrav26 = []
snrav28 = []
snrav30= []
npow = []

#===========================#
#========Main Part==========#
#===========================#
for name in fileName:
```

```
#-- Prepare the document to record the table (SNR vs Ampli.)
    f = open(name + '.txt', 'r')
    pow = open(name + '-lowpow' + '.txt', 'r')
    power = pow.readline()
    npow.append(float(power))
    print(npow)
    line = f.readlines()
    for lines in line:
        i = lines.split()
        if i[0] == '0':
            snrav0.append(float(i[1]))
        if i[0] == '1':
            snrav1.append(float(i[1]))
        if i[0] == '2':
            snrav2.append(float(i[1]))
        if i[0] == '4':
            snrav4.append(float(i[1]))
        if i[0] == '6':
            snrav6.append(float(i[1]))
        if i[0] == '8':
            snrav8.append(float(i[1]))
        if i[0] == '10':
            snrav10.append(float(i[1]))
        if i[0] == '12':
            snrav12.append(float(i[1]))
        if i[0] == '14':
            snrav14.append(float(i[1]))
        if i[0] == '16':
            snrav16.append(float(i[1]))
        if i[0] == '18':
            snrav18.append(float(i[1]))
        if i[0] == '20':
            snrav20.append(float(i[1]))
        if i[0] == '22':
            snrav22.append(float(i[1]))
        if i[0] == '24':
            snrav24.append(float(i[1]))
        if i[0] == '26':
            snrav26.append(float(i[1]))
        if i[0] == '28':
            snrav28.append(float(i[1]))
        if i[0] == '30':
            snrav30.append(float(i[1]))
#-- Load data from the original file
```

```
    f.close()
    pow.close()

plt.plot(npow,snrav0,'r')
#plt.plot(npow,snrav1,'b')
#plt.plot(npow,snrav2,'o')
#plt.plot(npow,snrav4,'y')
#plt.plot(npow,snrav6,'g')
#plt.plot(npow,snrav8,'p')
#plt.plot(npow,snrav10,'w')
#plt.plot(npow,snrav12,'a')
#plt.plot(npow,snrav14,'c')
#plt.plot(npow,snrav16,'d')
#plt.plot(npow,snrav18,'e')
#plt.plot(npow,snrav20,'f')
#plt.plot(npow,snrav22,'h')
#plt.plot(npow,snrav24,'i')
#plt.plot(npow,snrav26,'j')
#plt.plot(npow,snrav28,'l')
plt.plot(npow,snrav30,'b')

plt.ylabel("SNR")
plt.xlabel("Noise Power")
plt.title(" Noise Power vs SNR plot")
```