계산천체물리경진대회 문제2:

중력파 데이터 분석

2023 GWNRWinterSchool @UNIST 2023.01.30 – 2023.02.03

JeongCho Kim (SNU)

Data Set

- GWNR_Prob2A_H1.gwf
  - channel name : H1:GWNR2023

Q2-1, Q2-2, Q2-3, Q2-4

- GWNR_Prob2B_H1.gwf
  - channel name : H1:GWNR2023
- GWNR_Prob2B_L1.gwf
  - channel name : L1:GWNR2023

Q2-5

How to read data

Read local file :

frame.read_frame(file, channel name)

PyCBC

Read local file :
TimeSeries.read(file, channel name)

Download file :
TimeSeries.fetch_open_data(IFO, start, end)
TimeSeries.get(channel name, start, end, host=”url")

GWpy

How to read data

Read local file :

```
ifo_list = bilby.gw.detector.InterferometerList([])
ifo = bilby.gw.detector.get_empty_interferometer(IFO)
ifo.strain_data.set_from_frame_file(file, channel name, sampling_frequency,
                                    start_time, duration)
ifo_list.append(ifo)
```

Bilby

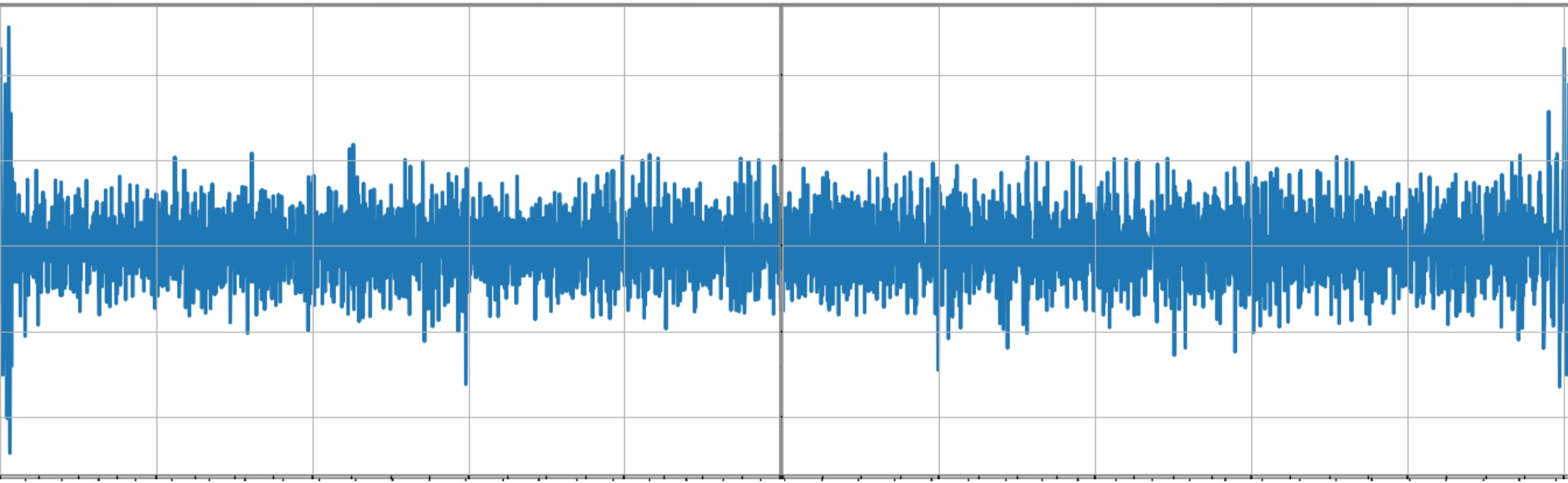
Q2-1. [10점] 정보추출

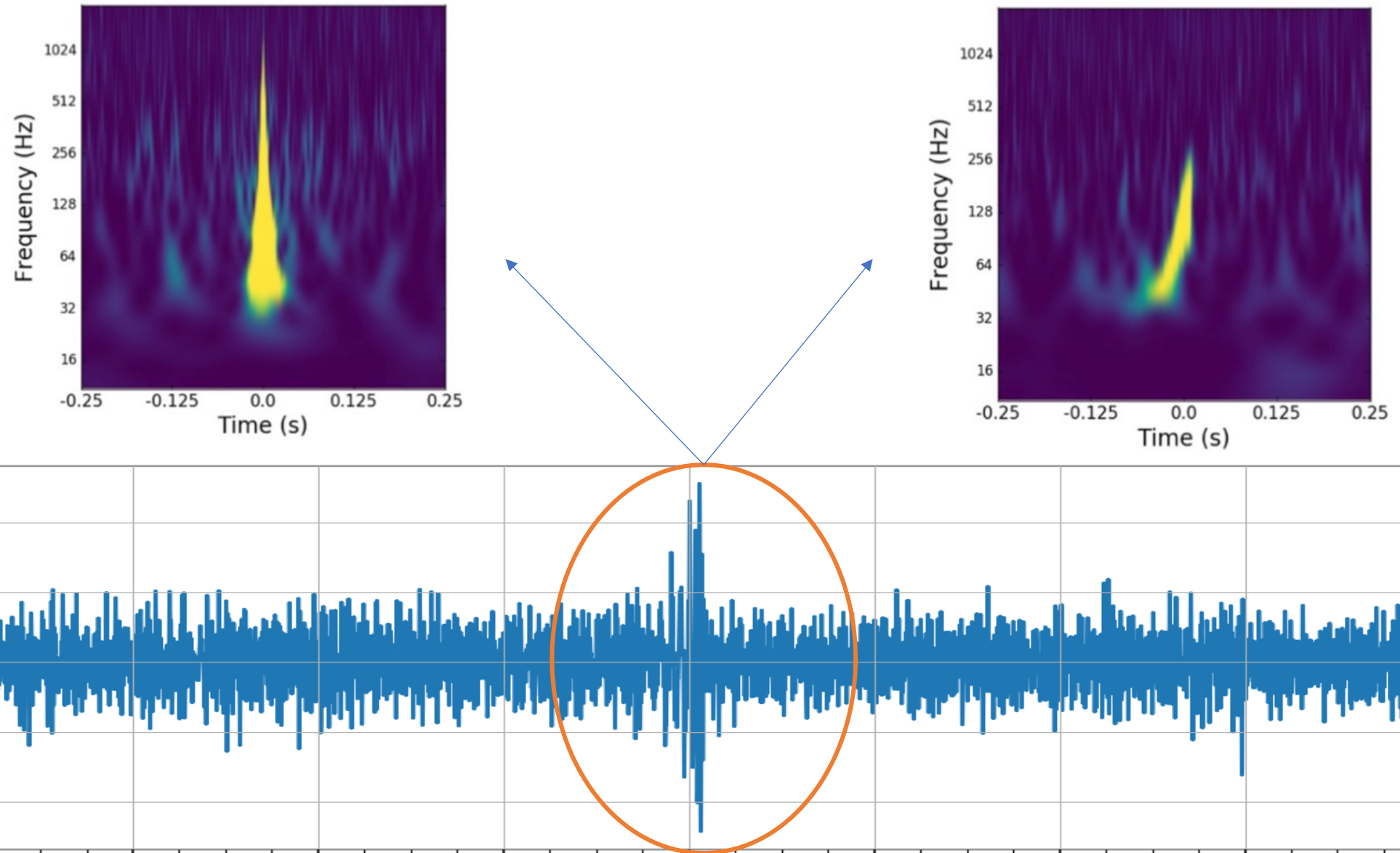
문제 1. 주어진 중력파 데이터 (GWNR_Prob2A_H1.gwf)의 시작시간(gps time), 종료시간(gps time), sampling rate, 데이터의 길이를 출력하는 코드를 작성한다.

- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ1_A_NN.txt

출력변수의 순서는 시작시간, 종료시간, sampling rate, 데이터의 길이 순서이다.

```
f= open("GWQ1_A_NN.txt","w")
f.write("{:.3f} {:.3f} {:.3f} {:.3f}".format(float(start_time), float(end_time), sample_rate, duration))
f.close()
```

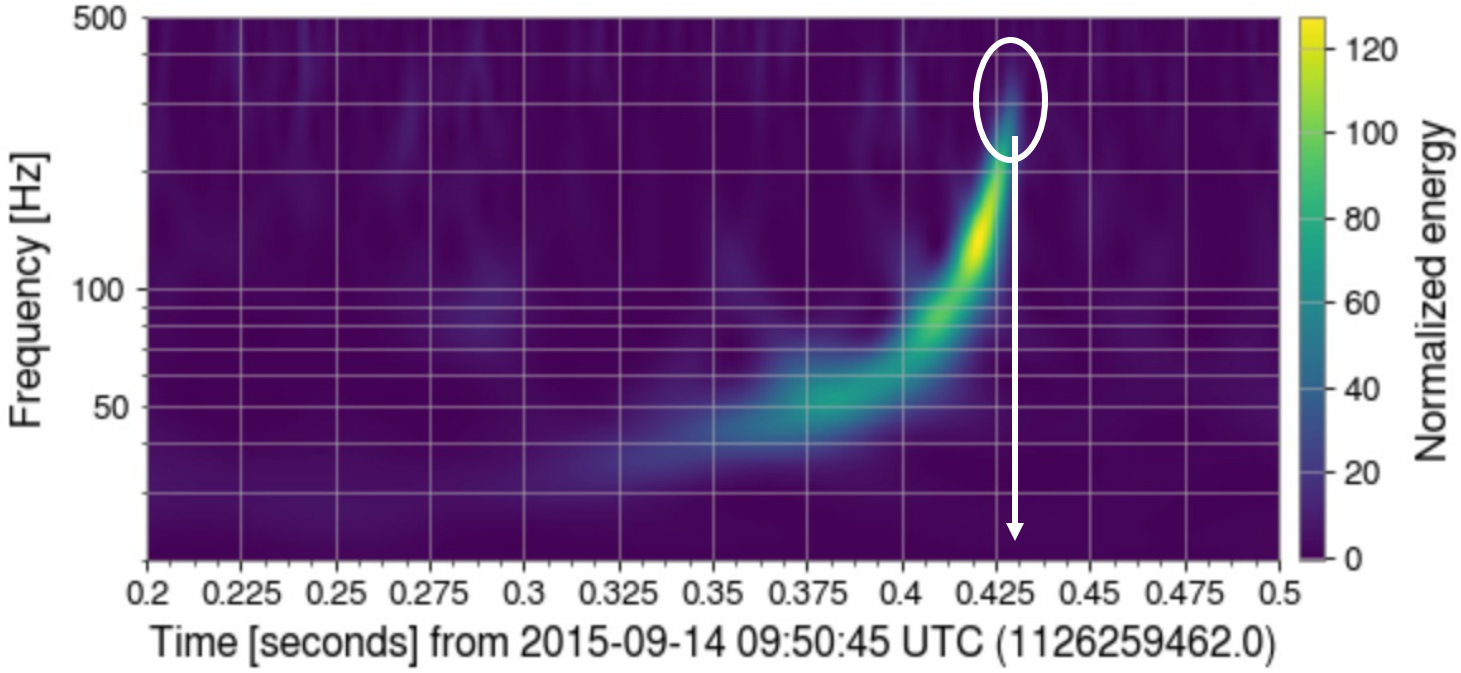



1024
512
256
128
64
32
16
Frequency (Hz)
-0.25
-0.125
0.0
0.125
0.25
Time (s)
1024
512
256
128
64
32
16
Frequency (Hz)
-0.25
-0.125
0.0
0.125
0.25
Time (s)

Q2-2. [20점] Q-transform

문제 2. 주어진 중력파 데이터 (GWNR_Prob2A_H1.gwf)로 부터 Q-transform을 생성하여, Q-transform으로 부터 병합 시간을 확인하여 출력코드에 입력한다.

- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ2_A_NN.txt



Q2-2. [20점] Q-transform

문제 2. 주어진 중력파 데이터 (GWNR_Prob2A_H1.gwf)로 부터 Q-transform을 생성하여, Q-transform으로 부터 병합 시간을 확인하여 출력코드에 입력한다.

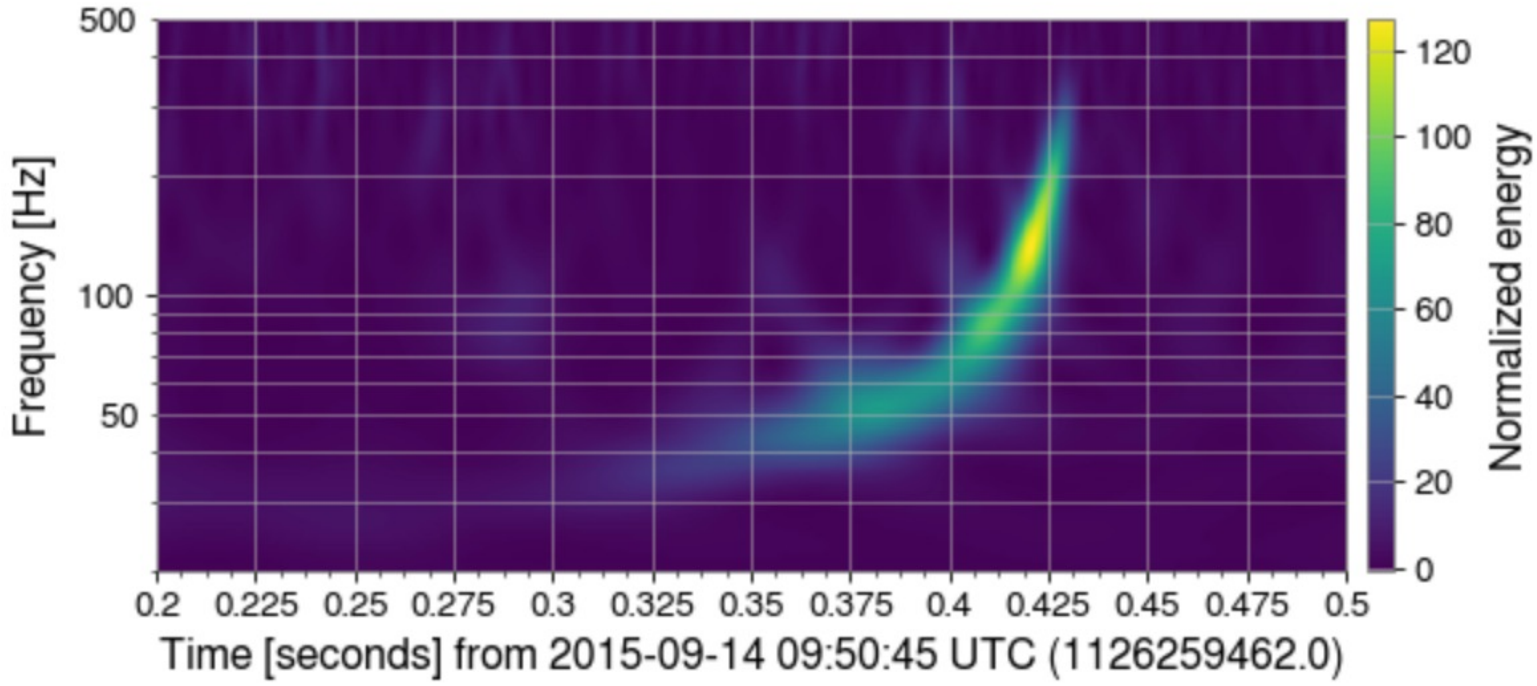
- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ2_A_NN.txt

```
# GW 문제 2번 출력 코드 (수정금지)

f= open("GWQ2_A_NN.txt","w")
mergeTime = input("Q-trasform에서 병합시간을 확인하여 입력 하세요 (소수점 둘째짜리까지 입력하세요.) : ")
f.write("{:.2f}" .format(float(mergeTime)))

f.close()
```

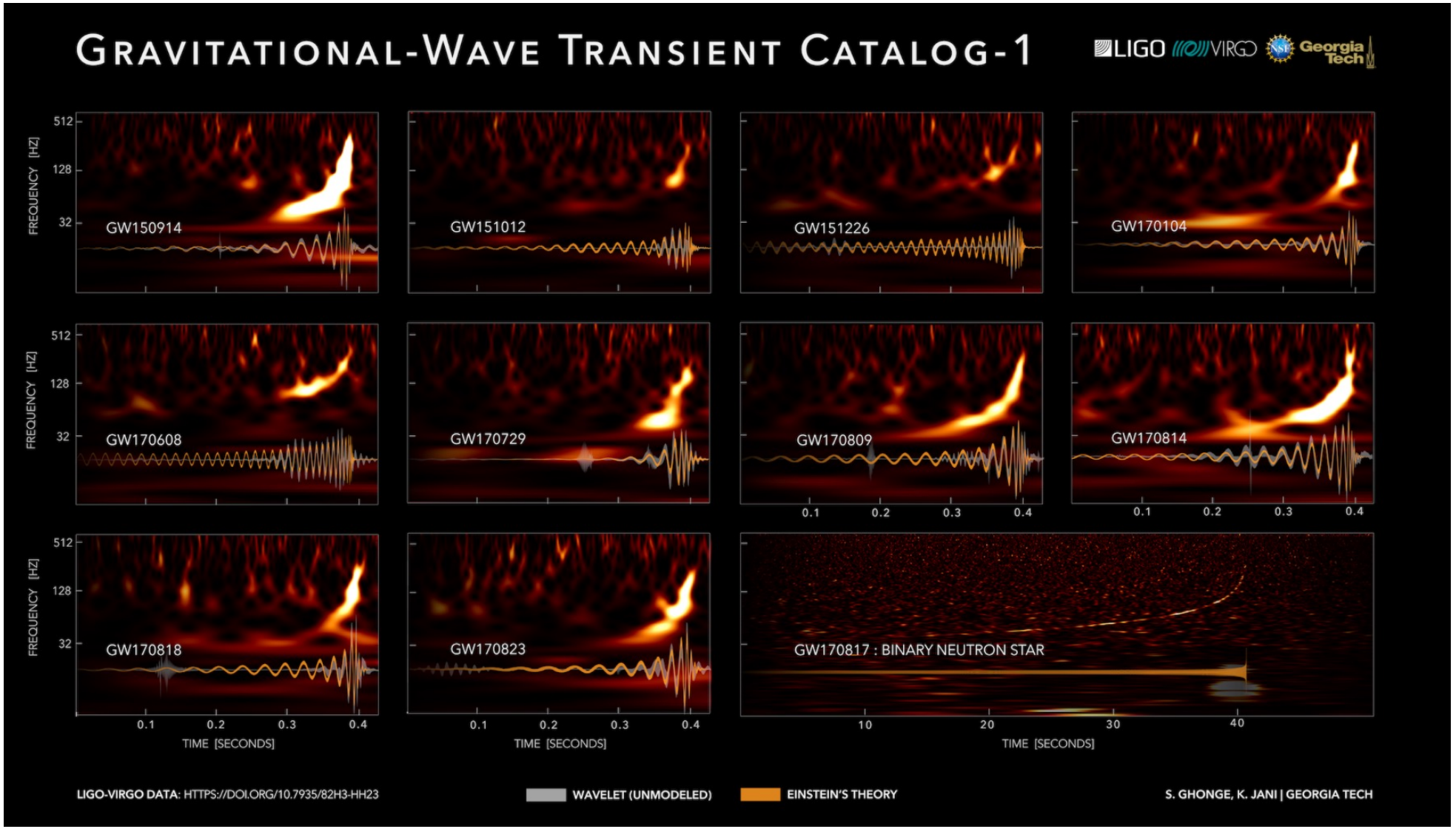
```
Q-trasform에서 병합시간을 확인하여 입력 하세요 (소수점 둘째짜리까지 입력하세요.) :
```

Q-transform for GW150914

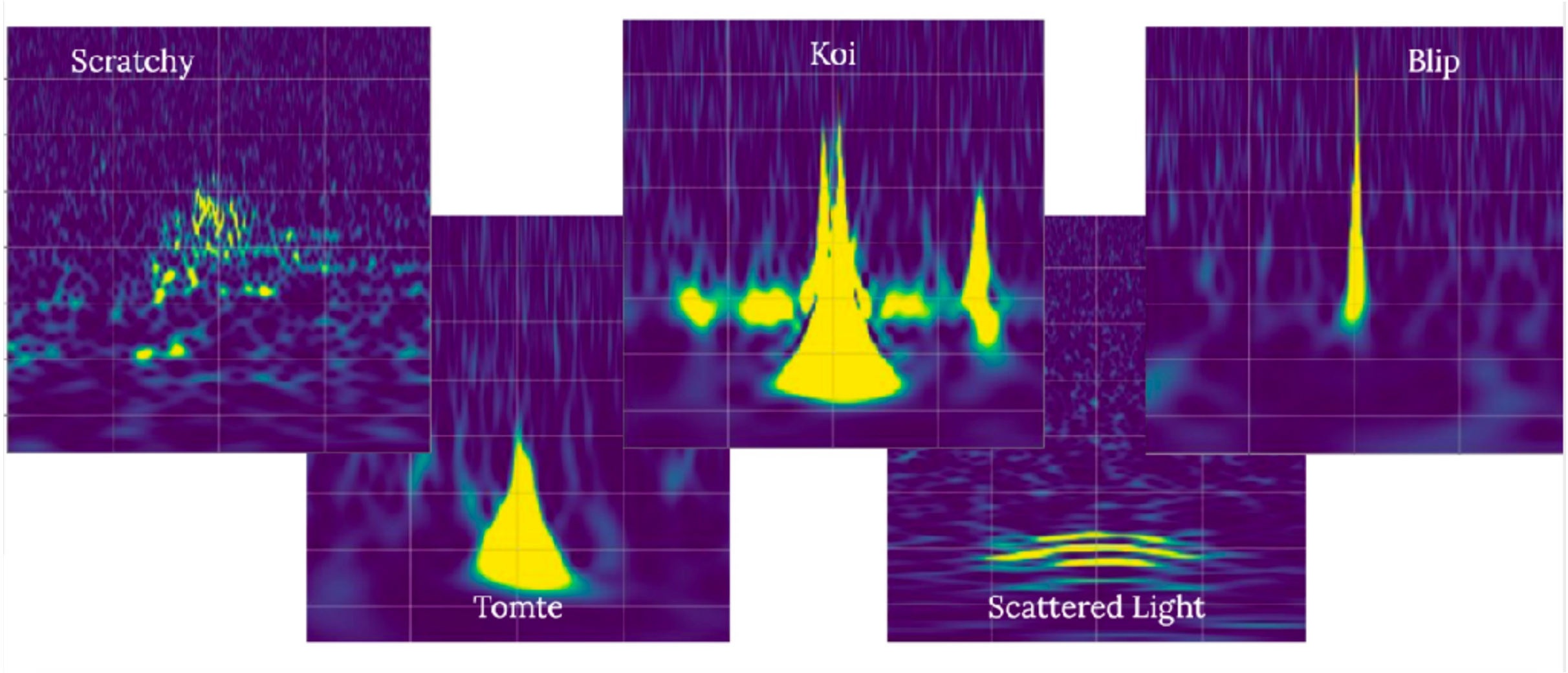
```
from gwpy.timeseries import TimeSeries
data = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478)
qspecgram = data.q_transform(outseg=(1126259462.2, 1126259462.5))
plot = qspecgram.plot(figsize=[8, 4])
ax = plot.gca()
ax.set_xscale('seconds')
ax.set_yscale('log')
ax.set_ylim(20, 500)
ax.set_ylabel('Frequency [Hz]')
ax.grid(True, axis='y', which='bot
ax.colorbar(cmap='viridis', label=
plot.show()
```



Q-transform (GW signal)



Q-transform (Non-Gaussian Glitches)



지난 겨울학교 (2022WinterSchool) 김영민박사님 자료.

Q2-3. [20점] SNR

문제 3. 주어진 중력파 데이터 (GWNR_Prob2A_H1.gwf)로 부터 PSD (Power Spectrum Density)와 주어진 조건의 template을 이용하여 SNR을 계산하는 코드를 작성하고 SNR값을 출력한다. SNR이 값이 높게 나올때의 정확한 신호의 시간도 함께 출력한다.

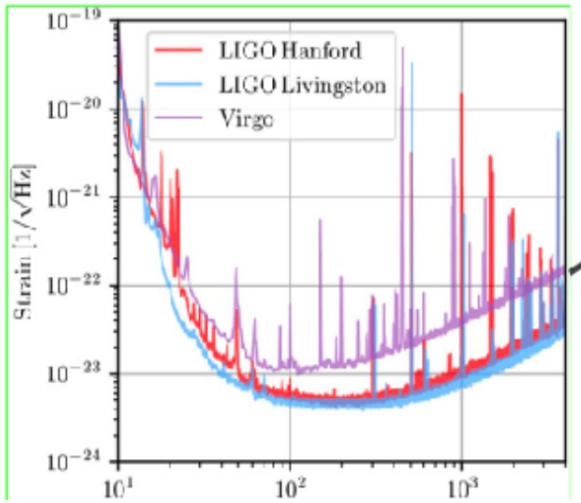
(Hint : waveform 생성, PSD 계산 후 matched filetering을 통하여 SNR값을 계산한다.)

- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ3_A_NN.txt

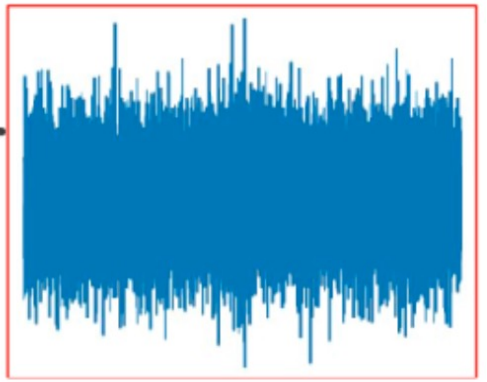
```
f= open("GWQ3_A_NN.txt","w")
f.write("{:.3f} {:.3f}" .format(snr_peak, time))
f.close()
```

mathched filtering

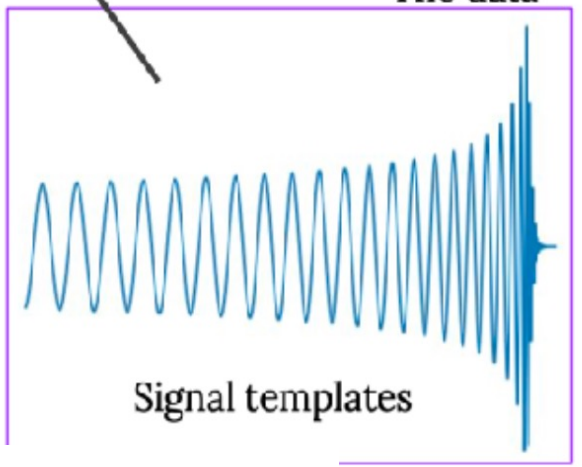
$$(s|h) = 4\Re \int_0^\infty \frac{\tilde{s}(f)\tilde{h}^\star(f)}{S_h(f)} \mathrm{d}f$$



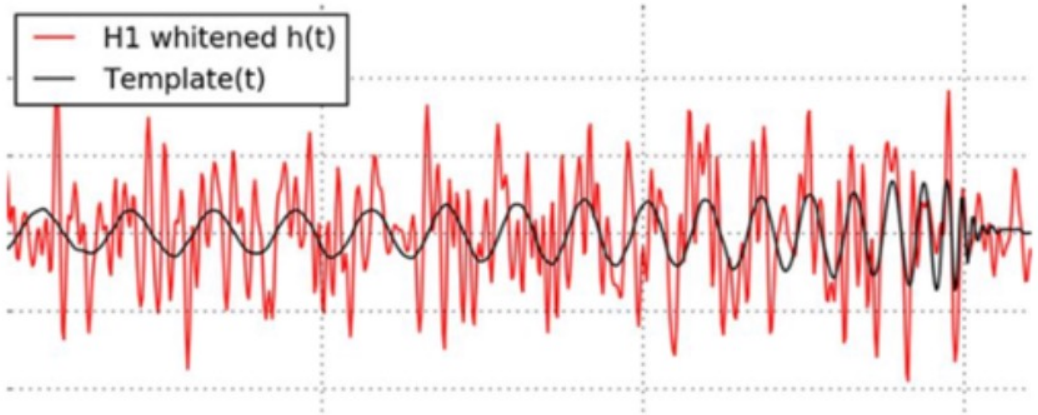
PSD

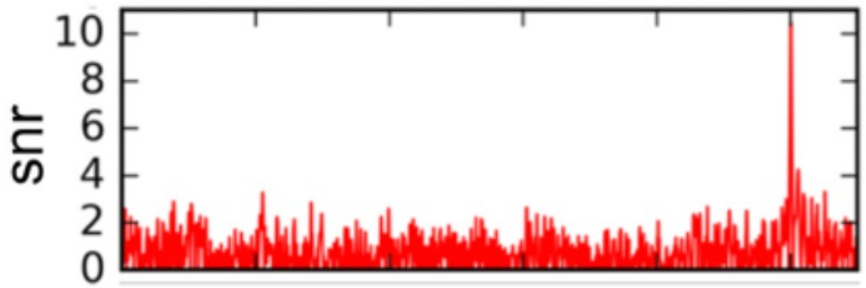


The data



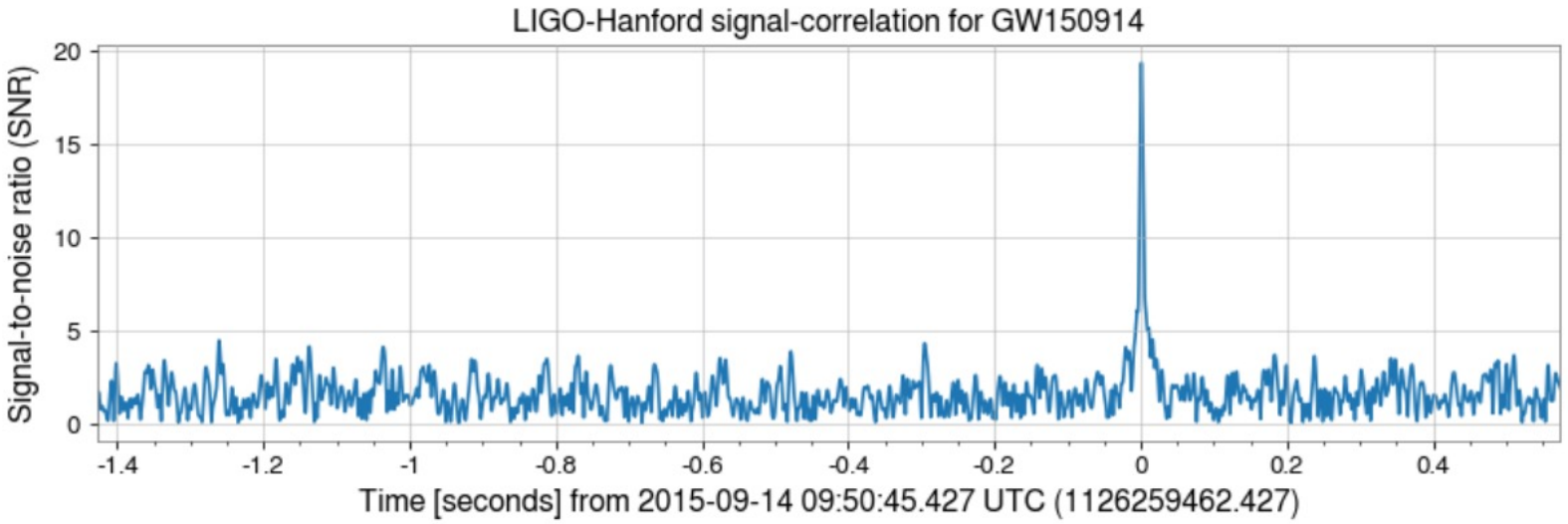
Signal templates





Matched filtering for GW150914

```
from gwpy.timeseries import TimeSeries
data = TimeSeries.fetch_open_data('H1', 1126259446, 1126259478)
high = data.highpass(15)
psd = high.psd(4, 2)
zoom = high.crop(1126259460, 1126259464)
from pycbc.waveform import get_fd_waveform
hp, _ = get_fd_waveform(approximant="IMRPhenomD", mass1=40, mass2=32,
                        f_lower=20, f_final=2048, delta_f=psd.df.value)
from pycbc.filter import matched_filter
snr = matched_filter(hp, zoom.to_pycbc(
                     low_frequency_cutoff=15)
snrts = TimeSeries.from_pycbc(snr).abs(
plot = snrts.plot()
ax = plot.gca()
ax.set_xlim(1126259461, 1126259463)
ax.set_epoch(1126259462.427)
ax.set_ylabel('Signal-to-noise ratio (SNR)
ax.set_title('LIGO-Hanford signal-correla
plot.show()
```



mathched filtering

Signal template

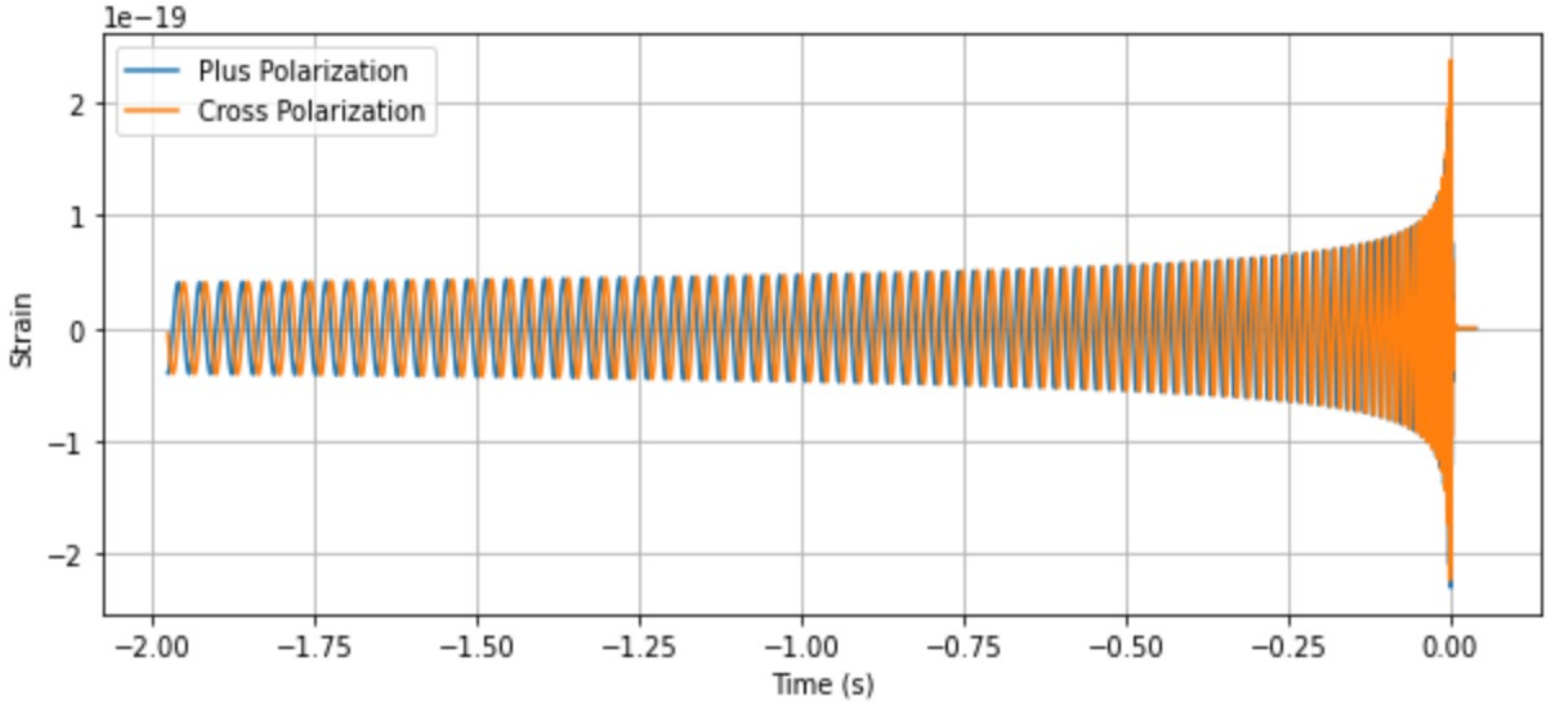
psd

```
matched_filter(hp, zoom.to_pycbc(), psd=psd.to_pycbc(), low_frequency_cutoff=15)
```

The data

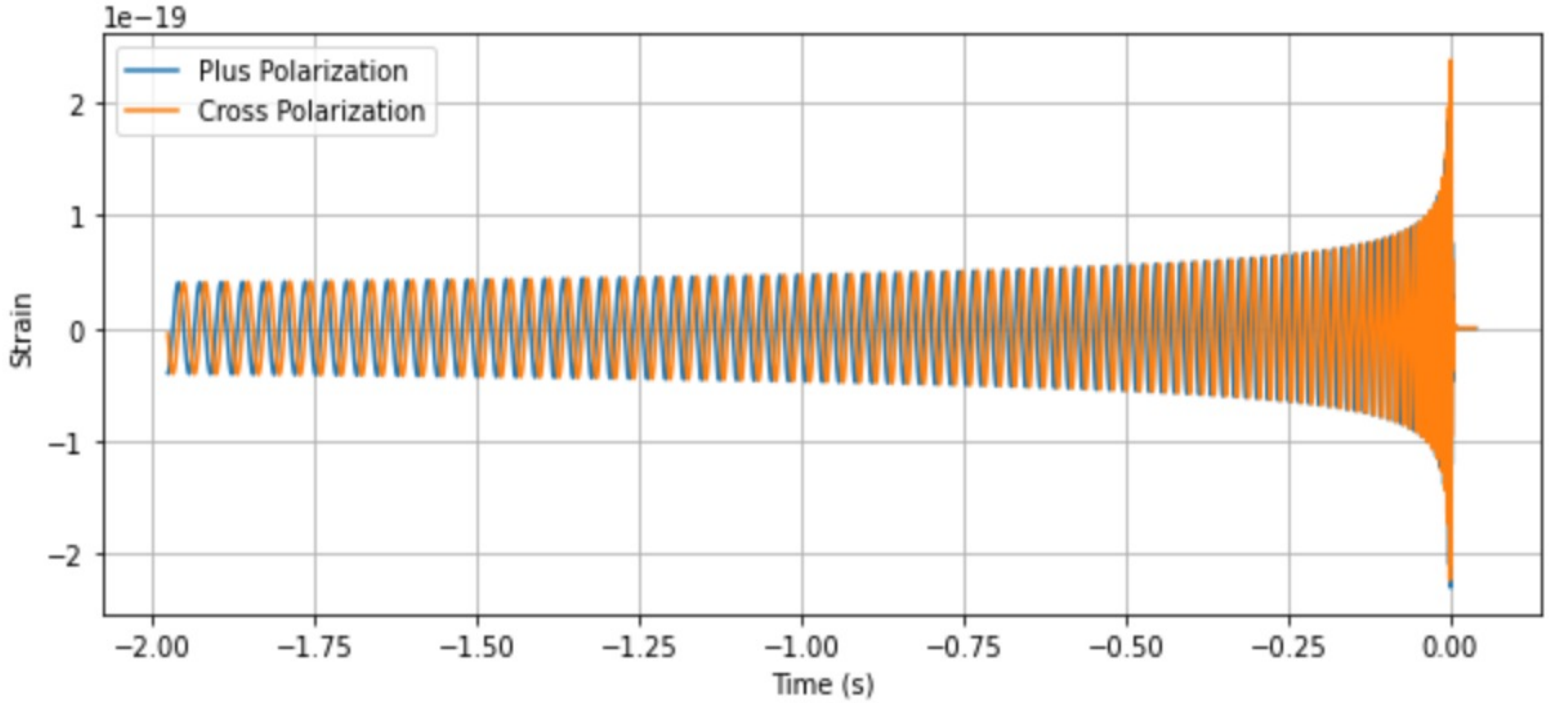
waveform

```
hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                         mass1=10,
                         mass2=10,
                         delta_t=1.0/16384,
                         f_lower=30)

pylab.figure(figsize=pylab.figaspect(0.4))
pylab.plot(hp.sample_times, hp, label='Plus Polarization')
pylab.plot(hp.sample_times, hc, label='Cross Polarization')
pylab.xlabel('Time (s)')
pylab.ylabel('Strain')
pylab.legend()
pylab.grid()
pylab.show()
```



waveform

```
hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                         mass1=10,
                         mass2=10,
                         delta_t=1.0/16384,
                         f_lower=30)

pylab.figure(figsize=pylab.figaspect(0.4))
pylab.plot(hp.sample_times, hp, label='Plus Polarization')
pylab.plot(hp.sample_times, hc, label='Cross Polarization')
pylab.xlabel('Time (s)')
pylab.ylabel('Strain')
pylab.legend()
pylab.grid()
pylab.show()
```



waveform

```
from pycbc.waveform import td_approximants, fd_approximants

# List of td approximants that are available
print(td_approximants())

# List of fd approximants that are currently available
print(fd_approximants())
```

```
['TaylorT1', 'TaylorT2', 'TaylorT3', 'SpinTaylorT1', 'SpinTaylorT4', 'SpinTaylorT5', 'PhenSpinTaylor', 'PhenSpinTay
lorRD', 'EOBNRv2', 'EOBNRv2HM', 'TEOBResum_ROM', 'SEOBNRv1', 'SEOBNRv2', 'SEOBNRv2_opt', 'SEOBNRv3', 'SEOBNRv3_pert
', 'SEOBNRv3_opt', 'SEOBNRv3_opt_rk4', 'SEOBNRv4', 'SEOBNRv4_opt', 'SEOBNRv4P', 'SEOBNRv4PHM', 'SEOBNRv2T', 'SEOBNR
v4T', 'SEOBNRv4_ROM_NRTidalv2', 'SEOBNRv4_ROM_NRTidalv2_NSBH', 'HGimri', 'IMRPhenomA', 'IMRPhenomB', 'IMRPhenomC',
'IMRPhenomD', 'IMRPhenomD_NRTidalv2', 'IMRPhenomNSBH', 'IMRPhenomHM', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal', 'IMRP
henomPv2_NRTidalv2', 'TaylorEt', 'TaylorT4', 'EccentricTD', 'SpinDominatedWf', 'NR_hdf5', 'NRSur7dq2', 'NRSur7dq4',
'SEOBNRv4HM', 'NRHybSur3dq8', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM', 'IMRPhenomXP', 'IMR
PhenomXPHM', 'TEOBResumS', 'IMRPhenomT', 'IMRPhenomTHM', 'IMRPhenomTP', 'IMRPhenomTPHM', 'TaylorF2', 'SEOBNRv1_ROM_
EffectiveSpin', 'SEOBNRv1_ROM_DoubleSpin', 'SEOBNRv2_ROM_EffectiveSpin', 'SEOBNRv2_ROM_DoubleSpin', 'EOBNRv2_ROM',
'EOBNRv2HM_ROM', 'SEOBNRv2_ROM_DoubleSpin_HI', 'SEOBNRv4_ROM', 'SEOBNRv4HM_ROM', 'IMRPhenomD_NRTidal', 'SpinTaylorF
2', 'TaylorF2NL', 'PreTaylorF2', 'SpinTaylorF2_SWAPPER', 'GWS-NRHybSur3dq8', 'GWS-NRHybSur3dq8Tidal', 'GWS-NRSur7dq
4']
['EccentricFD', 'TaylorF2', 'TaylorF2Ecc', 'TaylorF2NLTides', 'TaylorF2RedSpin', 'TaylorF2RedSpinTidal', 'SpinTaylo
rF2', 'EOBNRv2_ROM', 'EOBNRv2HM_ROM', 'SEOBNRv1_ROM_EffectiveSpin', 'SEOBNRv1_ROM_DoubleSpin', 'SEOBNRv2_ROM_Effect
iveSpin', 'SEOBNRv2_ROM_DoubleSpin', 'SEOBNRv2_ROM_DoubleSpin_HI', 'Lackey_Tidal_2013_SEOBNRv2_ROM', 'SEOBNRv4_ROM'
, 'SEOBNRv4HM_ROM', 'SEOBNRv4_ROM_NRTidal', 'SEOBNRv4_ROM_NRTidalv2', 'SEOBNRv4_ROM_NRTidalv2_NSBH', 'SEOBNRv4T_sur
rogate', 'IMRPhenomA', 'IMRPhenomB', 'IMRPhenomC', 'IMRPhenomD', 'IMRPhenomD_NRTidal', 'IMRPhenomD_NRTidalv2', 'IMR
PhenomNSBH', 'IMRPhenomHM', 'IMRPhenomP', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal', 'IMRPhenomPv2_NRTidalv2', 'SpinTa
ylorT4Fourier', 'SpinTaylorT5Fourier', 'NRSur4d2s', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM
', 'IMRPhenomXP', 'IMRPhenomXPHM', 'SpinTaylorF2_SWAPPER', 'TaylorF2NL', 'PreTaylorF2', 'multiband', 'TaylorF2_INTE
RP', 'SpinTaylorT5', 'SEOBNRv1_ROM_EffectiveSpin_INTERP', 'SEOBNRv1_ROM_DoubleSpin_INTERP', 'SEOBNRv2_ROM_Effective
Spin_INTERP', 'SEOBNRv2_ROM_DoubleSpin_INTERP', 'EOBNRv2_ROM_INTERP', 'EOBNRv2HM_ROM_INTERP', 'SEOBNRv2_ROM_DoubleS
pin_HI_INTERP', 'SEOBNRv4_ROM_INTERP', 'SEOBNRv4HM_ROM_INTERP', 'SEOBNRv4', 'SEOBNRv4P', 'IMRPhenomC_INTERP', 'IMRP
henomD_INTERP', 'IMRPhenomPv2_INTERP', 'IMRPhenomD_NRTidal_INTERP', 'IMRPhenomPv2_NRTidal_INTERP', 'IMRPhenomHM_INT
ERP', 'IMRPhenomPv3HM_INTERP', 'IMRPhenomXHM_INTERP', 'IMRPhenomXPHM_INTERP', 'SpinTaylorF2_INTERP', 'TaylorF2NL_IN
TERP', 'PreTaylorF2_INTERP', 'SpinTaylorF2_SWAPPER_INTERP']
```

waveform

Parameters:: **template** (*object*) – An object that has attached properties. This can be used to subsitute for keyword arguments.

mass1 (*{None, float}*) – The mass of the first component object in the binary (in solar masses).

mass2 (*{None, float}*) – The mass of the second component object in the binary (in solar masses).

spin1x (*{0.0, float}*) – The x component of the first binary component's dimensionless spin.

spin1y (*{0.0, float}*) – The y component of the first binary component's dimensionless spin.

spin1z (*{0.0, float}*) – The z component of the first binary component's dimensionless spin.

spin2x (*{0.0, float}*) – The x component of the second binary component's dimensionless spin.

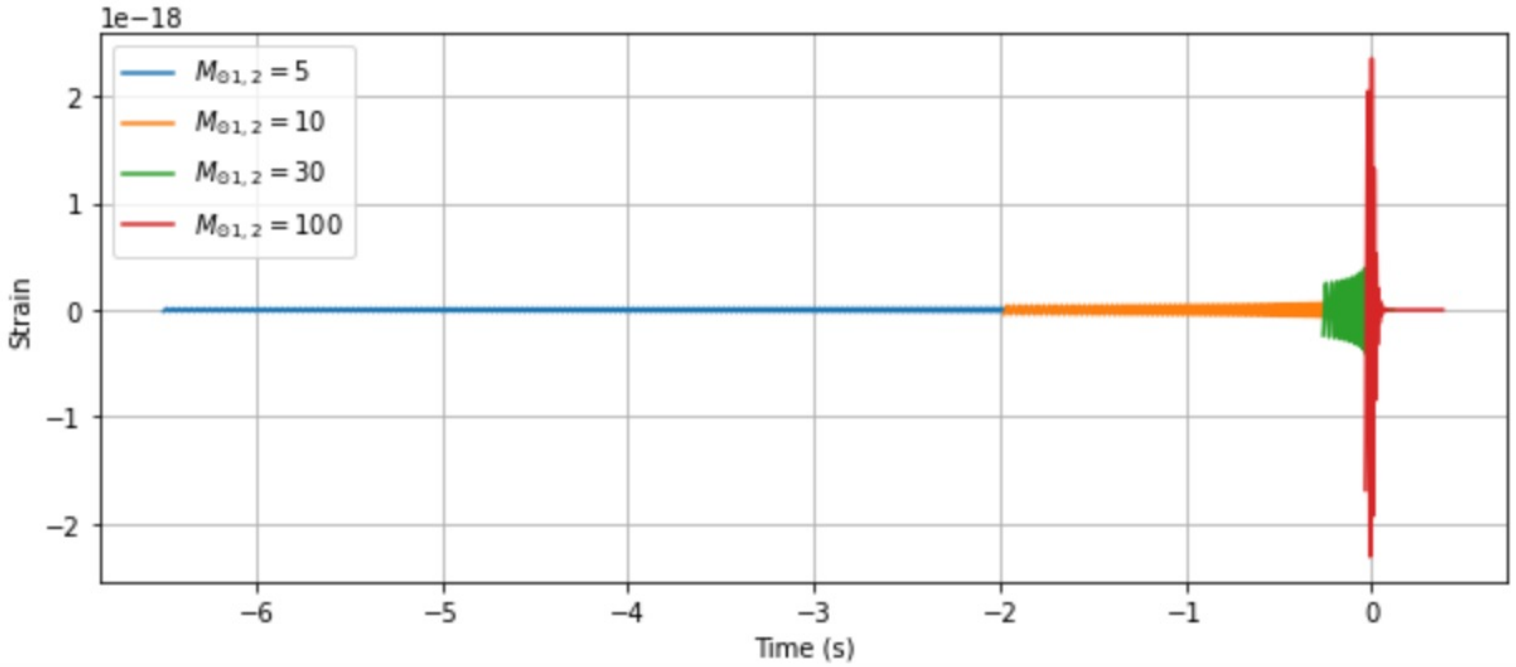
spin2y (*{0.0, float}*) – The y component of the second binary component's dimensionless spin.

spin2z (*{0.0, float}*) – The z component of the second binary component's dimensionless spin.

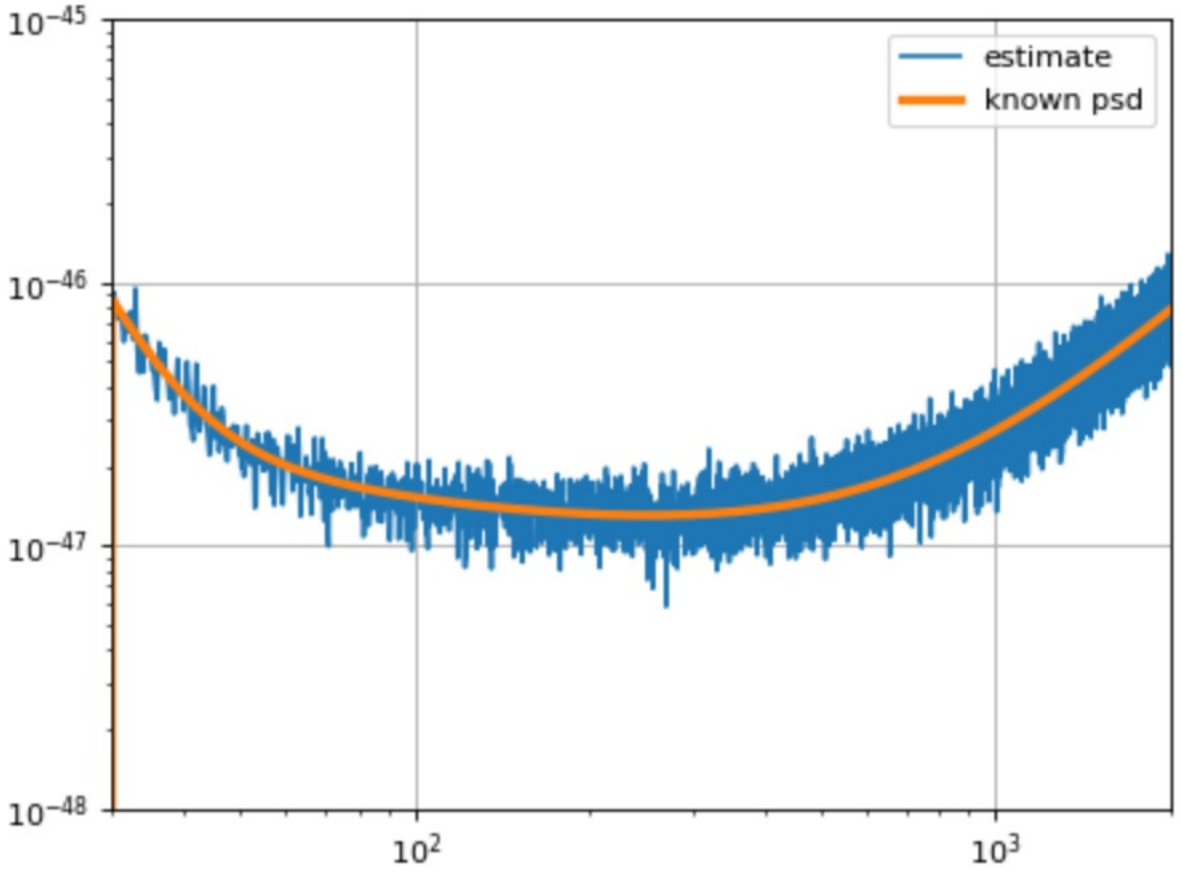
eccentricity (*{0.0, float}*) – Eccentricity.

waveform

```
# Component mass of each binary component. We'll simplify here and assume that each
# component of the binary has the same mass. Again, units are in solar masses.
pylab.figure(figsize=pylab.figaspect(0.4))
for m in [5, 10, 30, 100]:
    hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                             mass1=m,
                             mass2=m,
                             delta_t=1.0/4096,
                             f_lower=30)

    pylab.plot(hp.sample_times, hp, label='$M_{\odot 1,2}=%s$' % m)
pylab.legend()
pylab.grid()
pylab.xlabel('Time (s)')
pylab.ylabel('Strain')
pylab.show()
```

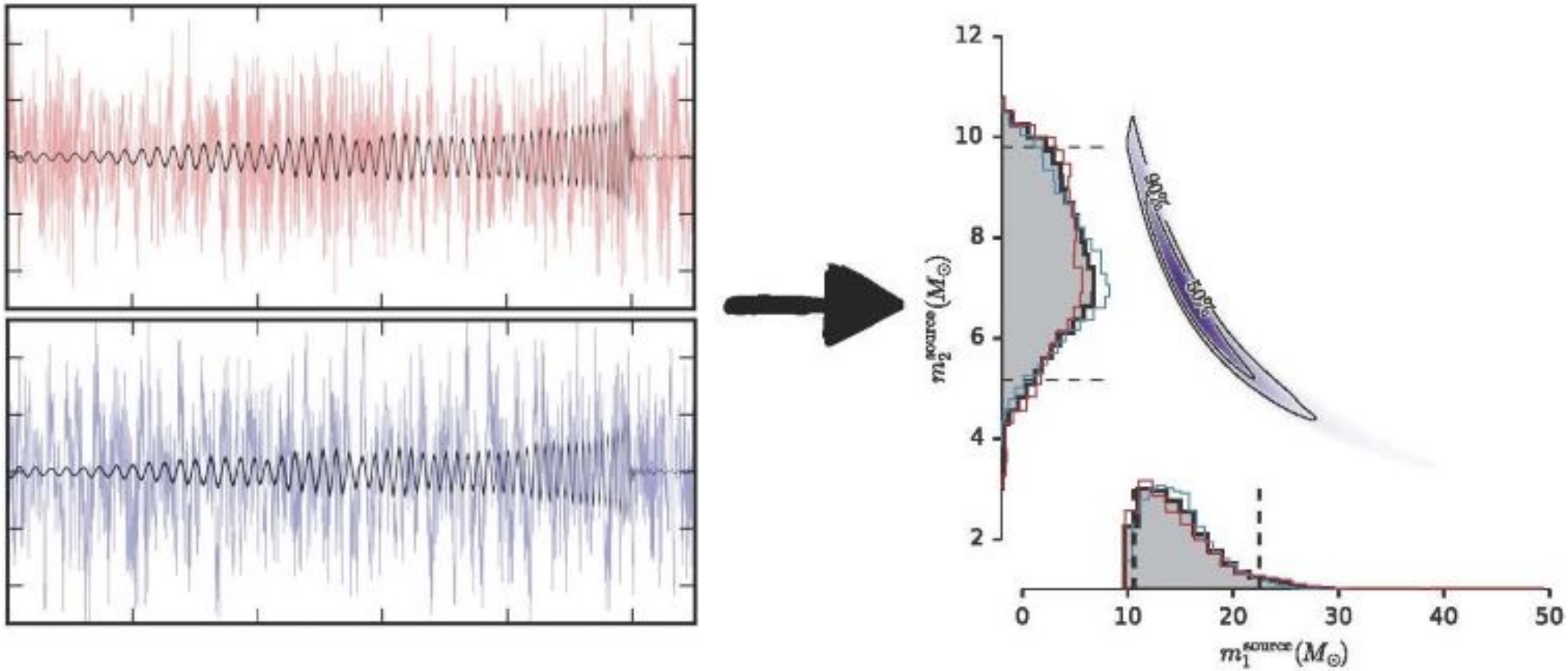


psd




```
import matplotlib.pyplot as pp
import pycbc.noise
import pycbc.psd
# generate some colored gaussian noise
flow = 30.0
delta_f = 1.0 / 16
flen = int(2048 / delta_f) + 1
                                    DZeroDetHighPower(flen, delta_f, flow)
                                    conds of noise at 4096 Hz


                                    delta_t)
                                    e_from_psd(tsamples, delta_t, psd, seed=127)


                                    nds PSD samples that are overlapped 50 %
                                    a_t)
                                    en / 2)
                                    oc.psd.welch(ts, seg_len=seg_len, seg_stride=seg_stride)
                                    psd.sample_frequencies, estimated_psd, label='estimate')
                                    _frequencies, psd, linewidth=3, label='known psd')
                                    nax=2000)
pp.grid()
pp.show()
```

중력파 모수추정


12
10
8
6
4
2
0
10
20
30
40
50
90%
50%
$m_2^{\rm source}(M_\odot)$
$m_1^{\rm source}(M_\odot)$

Bayes theorem

θ : 중력파원의 매개변수, d : 관측 데이터, M : 중력파 모델.

$$\mathrm{P}(\theta|d,\mathrm{M}) = \frac{P(d|\theta,\mathrm{M})P(\theta|\mathrm{M})}{P(d|\mathrm{M})} \rightarrow \mathrm{P}(\vec{\theta}|\vec{d}) = \frac{\mathcal{L}(\vec{d}|\vec{\theta})\mathcal{P}(\vec{\theta})}{P(d)}$$

Posterior : 관측한 데이터를 주는 중력파원 매개변수의 확률분포

Likelihood : 알고있는 중력파원의 매개변수로 부터 예측한 데이터와 실제 관측한 데이터의 유사도

Prior : 중력파원 매개변수의 분포 확률

Evidence : 사후분포에 대한 규격화 상수.

Q2-4. [20점] 중력파 모수추정 1

문제 4. 주어진 중력파 데이터 (GWNR_Prob2A_H1.gwf)로부터 Bilby를 활용한 중력파 모수추정을 수행하여 여러매개 변수중 chirp mass, luminosity distance의 posterior의 Median 값을 출력하는 코드를 작성하고, chirpmass와 distance의 median값을 순서대로 출력한다. 또한 Bilby의 결과를 통해 얻은 Bayes factor를 입력한다.

- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ4_A_NN.txt

```
f= open("GWQ4_A_NN.txt","w")
mc_median = float(input("chirpmass의 median값을 입력하세요 : "))
distance_median = float(input("distance의 median값을 입력하세요 : "))
bayes_factor = float(input("Bilby결과에서 bayes_factor 입력하세요 : "))
f.write("{:.3f} {:.3f} {:.3f}" .format(mc_median, distance_median,bayes_factor))
f.close()
```

File structure

```
print(posterior['Overall_posterior'].dtype.names)
```

```
('costheta_jn', 'luminosity_distance_Mpc', 'right_ascension', 'declination', 'm1_detector_frame_Msun', 'm2_detector_frame_Msun', 'spin1', 'spin2', 'costilt1', 'costilt2')
```

Here are some brief descriptions of these parameters and their uses:

- `luminosity_distance_Mpc` : luminosity distance [Mpc]
- `m1_detector_frame_Msun` : primary (larger) black hole mass (detector frame) [solar mass]
- `m2_detector_frame_Msun` : secondary (smaller) black hole mass (detector frame) [solar mass]
- `right_ascension` , `declination` : right ascension and declination of the source [rad].
- `costheta_jn` : cosine of the angle between line of sight and total angular momentum vector of system.
- `spin1` , `costilt1` : primary (larger) black hole spin magnitude (dimensionless) and cosine of the zenith angle between the spin and the orbital angular momentum vector of system.
- `spin2` , `costilt2` : secondary (smaller) black hole spin magnitude (dimensionless) and cosine of the zenith angle between the spin and the orbital angular momentum vector of system.

Posterior samples from PE

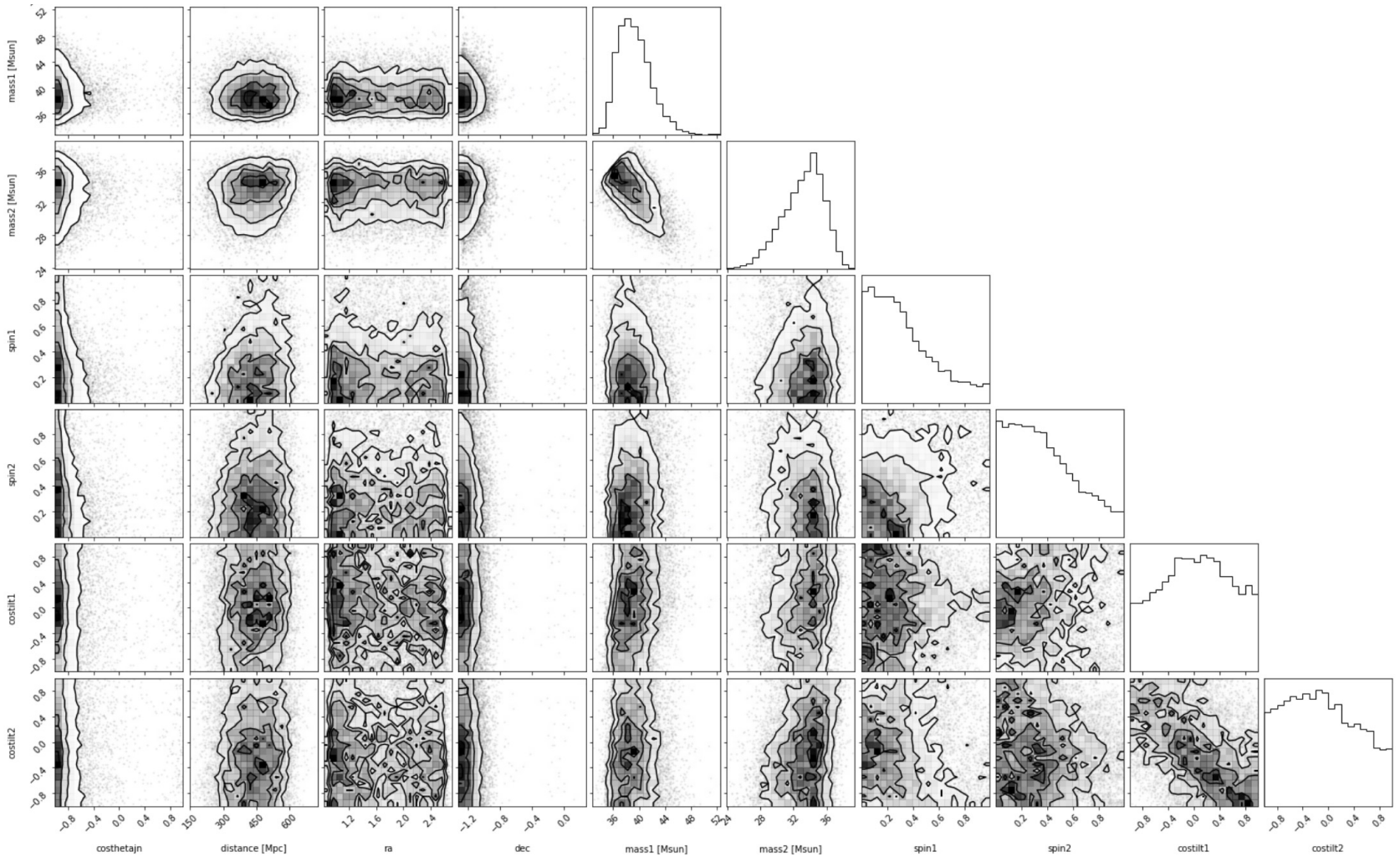
```
samples=pd.DataFrame.from_records(np.array(posterior['Overall_posterior']))
```

```
samples
```

	costheta_jn	luminosity_distance_Mpc	right_ascension	declination	m1_detector_frame_Msun	m2_detector_frame_Msun	spin1	spin2	costilt1	cost
0	-0.976633	517.176717	1.456176	-1.257815	39.037380	37.044563	0.417147	0.867740	-0.280624	0.4038
1	-0.700404	401.626864	2.658802	-0.874661	34.620096	34.184416	0.125709	0.260679	-0.757349	-0.3122
2	-0.840752	369.579071	1.106548	-1.136396	37.894343	33.970520	0.581047	0.926893	0.649781	-0.5108
3	-0.583657	386.935268	2.077180	-1.246351	36.412973	35.684463	0.235808	0.094391	0.116578	-0.7205
4	-0.928271	345.104345	0.993604	-1.069243	39.477251	31.645008	0.511521	0.868009	-0.438237	0.2693
...	...	...	...	...	...	...	...	...	...	
8345	-0.691637	306.985025	1.485646	-1.269228	37.561962	33.355792	0.484003	0.627191	0.194507	-0.4083
8346	-0.834615	462.649414	2.065362	-1.265618	37.824298	36.674075	0.589654	0.650758	-0.737792	0.8753
8347	-0.911463	448.930876	1.536913	-1.257956	38.063291	35.757913	0.708407	0.714805	0.852085	-0.7974
8348	-0.856914	561.020036	2.367289	-1.211824	44.884396	31.592433	0.389284	0.521304	-0.251461	0.8305
8349	-0.919556	519.641782	1.916675	-1.250801	37.275183	35.445032	0.391824	0.516908	-0.705305	0.6007

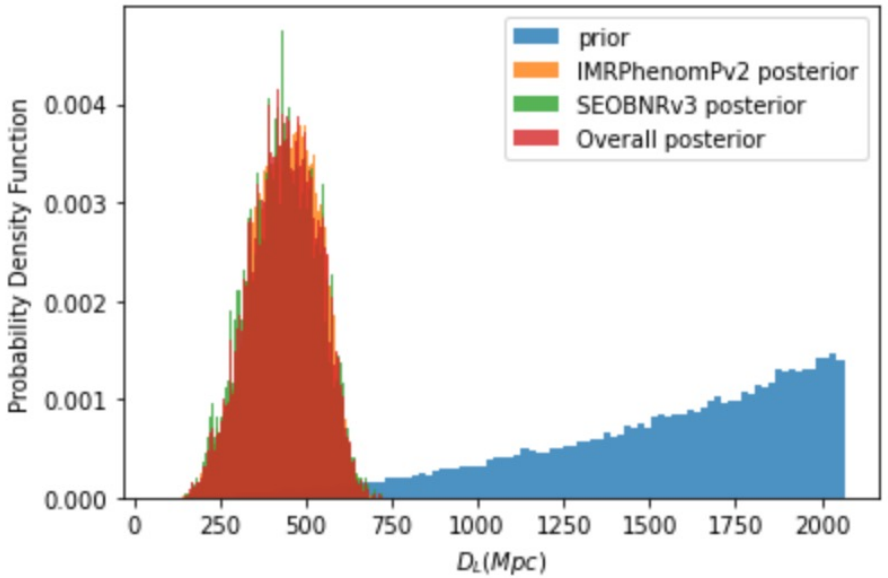
Posterior samples plot

```
corner.corner(samp
```



PDF (Probability Density Function)

```
plt.hist(posterior['prior']['luminosity_distance_Mpc'], bins = 100, label='prior', alpha=0.8, density=True)
plt.hist(posterior['IMRPhenomPv2_posterior']['luminosity_distance_Mpc'], bins = 100, label='IMRPhenomPv2 posterior', alpha=0.8, density=Tr
plt.hist(posterior['SEOBNRv3_posterior']['luminosity_distance_Mpc'], bins = 100, label='SEOBNRv3 posterior', alpha=0.8, density=True)
plt.hist(posterior['Overall_posterior']['luminosity_distance_Mpc'], bins = 100, label='Overall posterior', alpha=0.8, density=True)
plt.xlabel(r'$D_L (Mpc)$')
plt.ylabel('Probability Density Function')
plt.legend()
plt.show()
```



Bilby – Download data

```
import bilby
from bilby.core.prior import Uniform
from bilby.gw.conversion import
convert_to_lal_binary_black_hole_parameters,
generate_all_bbh_parameters

from gwpy.timeseries import TimeSeries
```

```
* Download data

# Definite times in relation to the trigger time (time_of_event), duration and post_trigger_duratic
post_trigger_duration = 2
duration = 4
analysis_start = time_of_event + post_trigger_duration - duration

# Use gwpy to fetch the open data
H1_analysis_data = TimeSeries.fetch_open_data(
    "H1", analysis_start, analysis_start + duration, sample_rate=4096, cache=True)

L1_analysis_data = TimeSeries.fetch_open_data(
    "L1", analysis_start, analysis_start + duration, sample_rate=4096, cache=True)
```

Bilby – PSD download

* Set up empty interferometers

```
H1 = bilby.gw.detector.get_empty_interferometer("H1")
L1 = bilby.gw.detector.get_empty_interferometer("L1")

H1.set_strain_data_from_gwpy_timeseries(H1_analysis_data)
L1.set_strain_data_from_gwpy_timeseries(L1_analysis_data)
```

* Download the PSD data

```
psd_duration = duration * 32
psd_start_time = analysis_start - psd_duration

H1_psd_data = TimeSeries.fetch_open_data(
    "H1", psd_start_time, psd_start_time + psd_duration, sample_rate=4096, cache=True)

L1_psd_data = TimeSeries.fetch_open_data(
    "L1", psd_start_time, psd_start_time + psd_duration, sample_rate=4096, cache=True)
```

```
psd_alpha = 2 * H1.strain_data.roll_off / duration
H1_psd = H1_psd_data.psd(fftlength=duration, overlap=0, window=("tukey", psd_alpha), method="median")
L1_psd = L1_psd_data.psd(fftlength=duration, overlap=0, window=("tukey", psd_alpha), method="median")
```

Bilby – PSD

```
    ⁎Initialise the PSD

 H1.power_spectral_density = bilby.gw.detector.PowerSpectralDensity(
     frequency_array=H1_psd.frequencies.value, psd_array=H1_psd.value)
 L1.power_spectral_density = bilby.gw.detector.PowerSpectralDensity(
     frequency_array=H1_psd.frequencies.value, psd_array=L1_psd.value)


fig, ax = plt.subplots()
idxs = H1.strain_data.frequency_mask  # This is a boolean
mask of the frequencies which we'll use in the analysis
ax.loglog(H1.strain_data.frequency_array[idxs],

np.abs(H1.strain_data.frequency_domain_strain[idxs]))
ax.loglog(H1.power_spectral_density.frequency_array[idxs],
          H1.power_spectral_density.asd_array[idxs])
ax.set_xlabel("Frequency [Hz]")
ax.set_ylabel("Strain [strain/$\sqrt{Hz}$]")
plt.show()
```

Bilby - Prior

```
prior = bilby.core.prior.PriorDict()
prior['chirp_mass'] = Uniform(name='chirp_mass', latex_label='$M$', minimum=5., maximum=15., unit='$M_{\\odot}$')
prior['mass_ratio'] = Uniform(name='mass_ratio', minimum=0.9, maximum=1.1)
prior['phase'] = Uniform(name="phase", minimum=0, maximum=2*np.pi)
prior['geocent_time'] = Uniform(name="geocent_time", minimum=time_of_event_est-0.5, maximum=time_of_event_est+0.5)
prior['a_1'] =  0.0
prior['a_2'] =  0.0
prior['tilt_1'] =  0.0
prior['tilt_2'] =  0.0
prior['phi_12'] =  0.0
prior['phi_jl'] =  0.0
prior['luminosity_distance'] = bilby.gw.prior.UniformSourceFrame(name='luminosity_distance',
                                                                 minimum=40, maximum=1000)
prior['theta_jn'] = Sine(name='theta_jn')
prior['ra'] = Uniform(name='ra', minimum=0, maximum=2 * np.pi, boundary='periodic')
prior['psi'] = Uniform(name='psi', minimum=0, maximum=np.pi, boundary='periodic')
prior['dec'] = Cosine(name='dec')
```

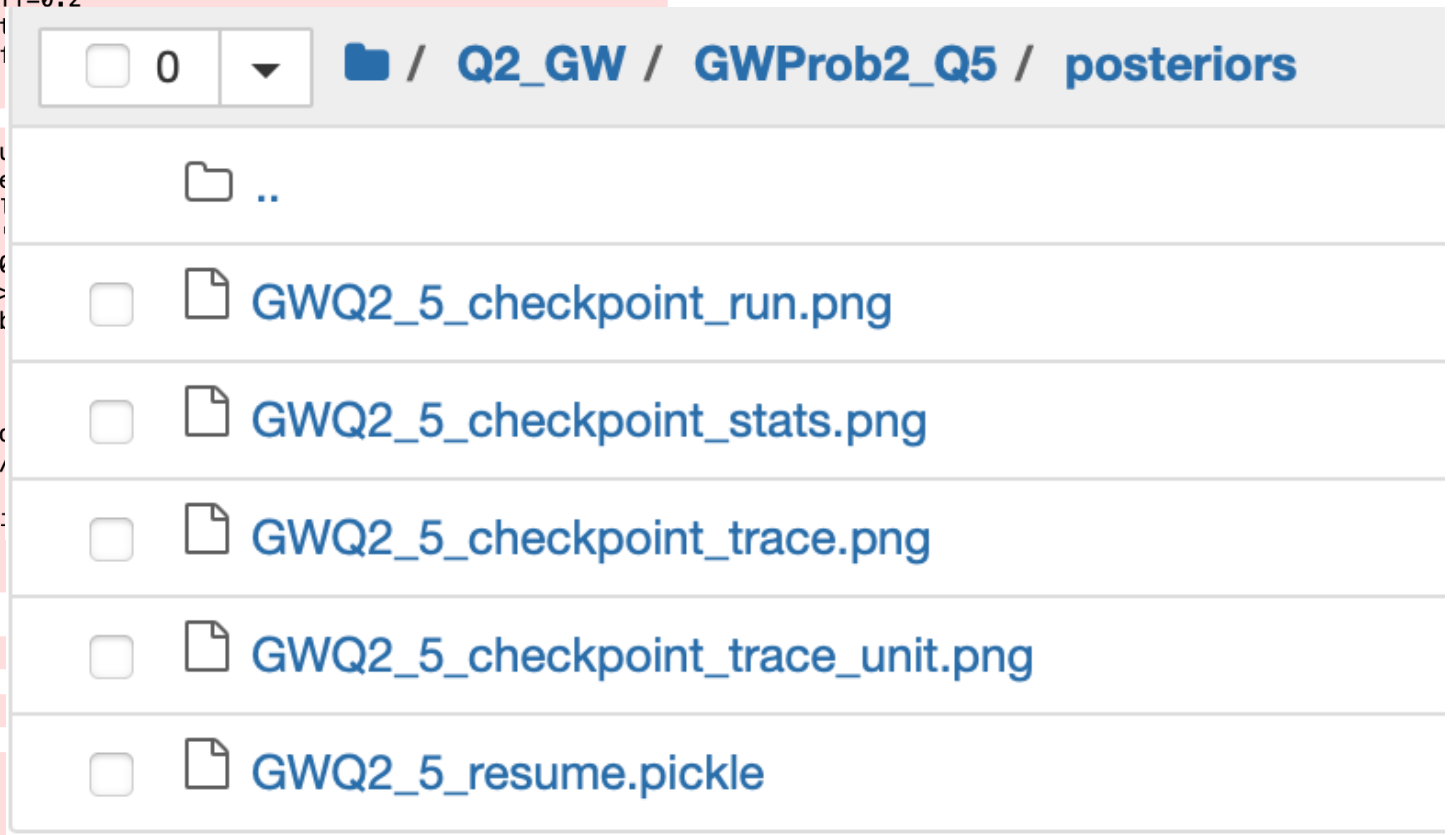
Bilby - likelihood

```
# Finally, create our likelihood, passing in what is needed to get going
likelihood = bilby.gw.likelihood.GravitationalWaveTransient(
    interferometers, waveform_generator, priors=prior,
    time_marginalization=True, phase_marginalization=True,
distance_marginalization=False)
```

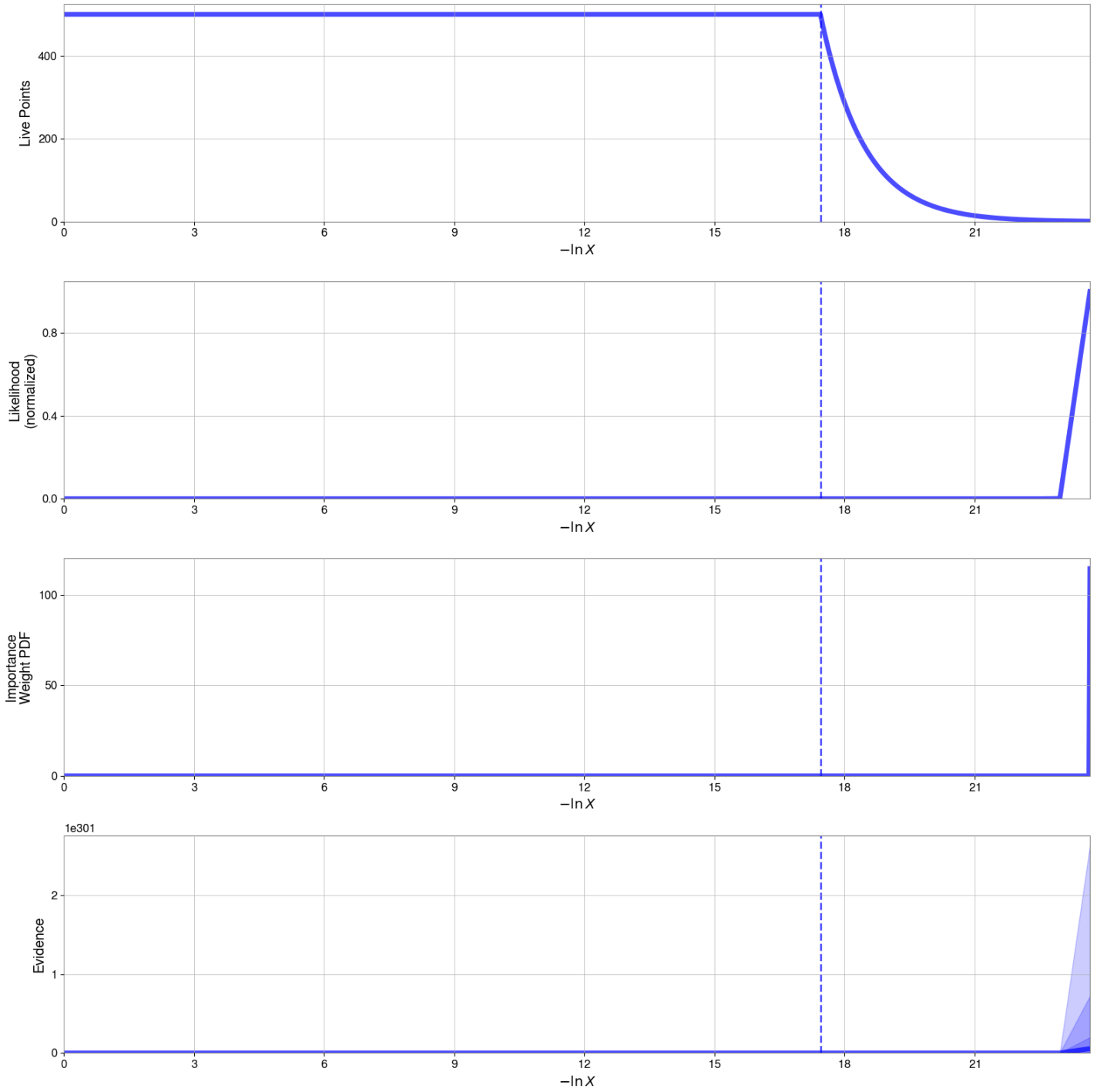
Bilby - run

```
result = bilby.run_sampler(
    likelihood, prior, sampler='dynesty', outdir='./GWProb2_Q5/posteriors', label="GWQ2_5",
    conversion_function=bilby.gw.conversion.generate_all_bbh_parameters, nlive = 500, dlogz=3)
```

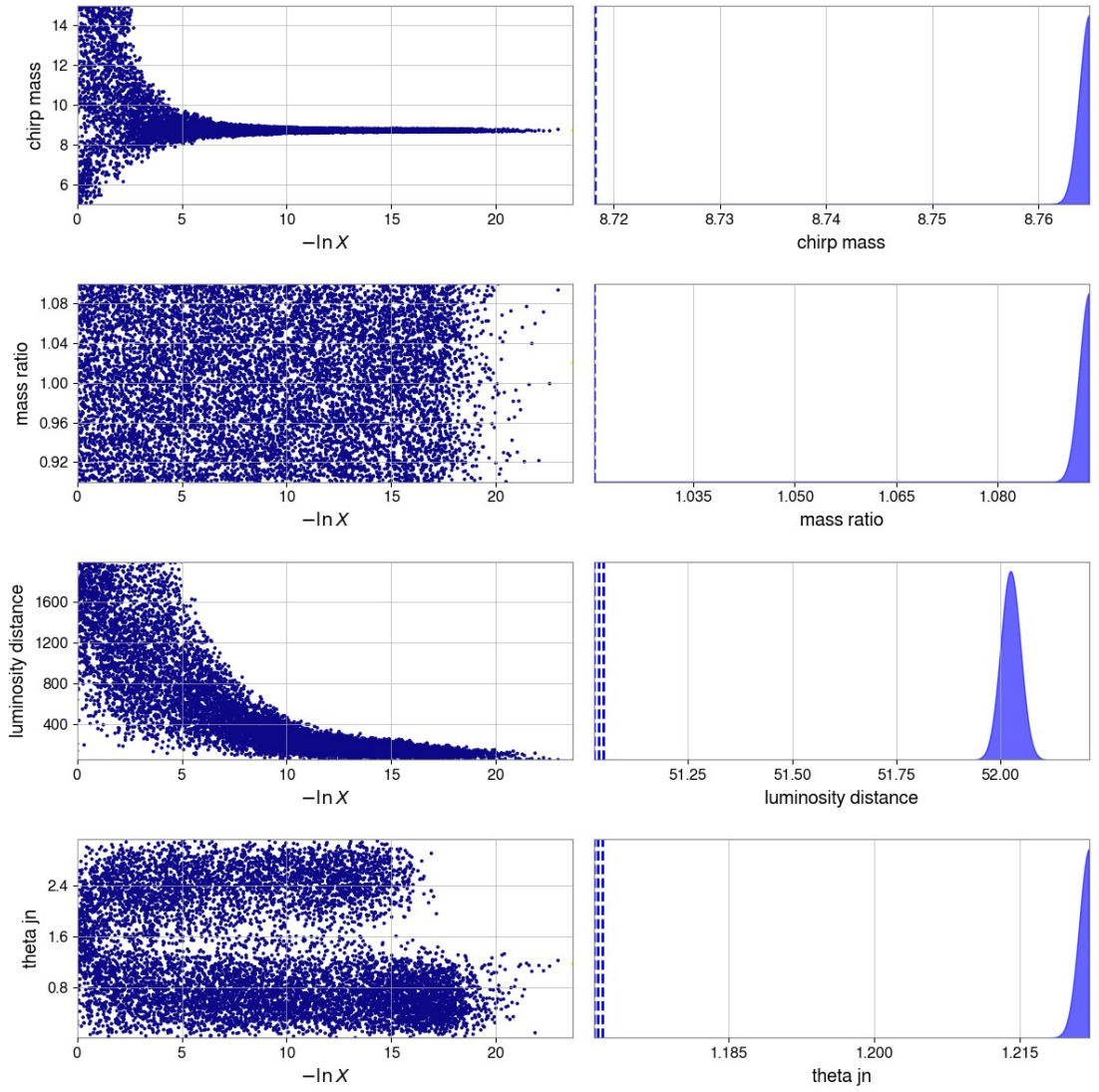
```
08:26 bilby INFO    : Generating frequency domain strain from given time domain strain.
08:26 bilby INFO    : Applying a tukey window with alpha=0.1, roll off=0.2
08:26 bilby INFO    : Generating frequency domain strain from given t
08:26 bilby INFO    : Applying a tukey window with alpha=0.1, roll of
08:26 bilby INFO    : Single likelihood evaluation took 8.877e-03 s

08:26 bilby INFO    : Using sampler Dynesty with kwargs {'bound': 'mu
iodic': None, 'reflective': None, 'check_point_delta_t': 1800, 'nlive
pdim': None, 'rstate': None, 'queue_size': 1, 'pool': None, 'use_pool
e, 'logl_kwargs': None, 'ptform_args': None, 'ptform_kwargs': None,
.5, 'vol_check': 8.0, 'facc': 0.2, 'slices': 5, 'update_interval': 30
func of <bilby.core.sampler.dynesty.Dynesty object at 0x2b702327aaf0>
e, 'logl_max': inf, 'add_live': True, 'print_progress': True, 'save_b
5000, 'nact': 5, 'print_method': 'tqdm'}
08:26 bilby INFO    : Checkpoint every check_point_delta_t = 600s
08:26 bilby INFO    : Using dynesty version 1.0.1
08:26 bilby INFO    : Using the bilby-implemented rwalk sample method
08:26 bilby INFO    : Reading resume file ./GWProb2_Q5_v2/posteriors/
08:26 bilby INFO    : Resume file successfully loaded.
08:33 bilby INFO    : Written checkpoint file ./GWProb2_Q5_v2/poster
12:56 bilby INFO    : Sampling time: 0:00:08.084629
12:56 bilby INFO    : Reconstructing marginalised parameters.

  0%|          | 0/1413 [00:00<?, ?it/s]
12:56 bilby INFO    : Generating sky frame parameters.
  0%|          | 0/1413 [00:00<?, ?it/s]
12:56 bilby INFO    : Computing SNRs for every sample.
  0%|          | 0/1413 [00:00<?, ?it/s]
12:57 bilby INFO    : Summary of results:
nsamples: 1413
ln_noise_evidence: -8534.562
ln_evidence: -8266.386 +/-  0.131
ln_bayes_factor: 268.175 +/-  0.131
```

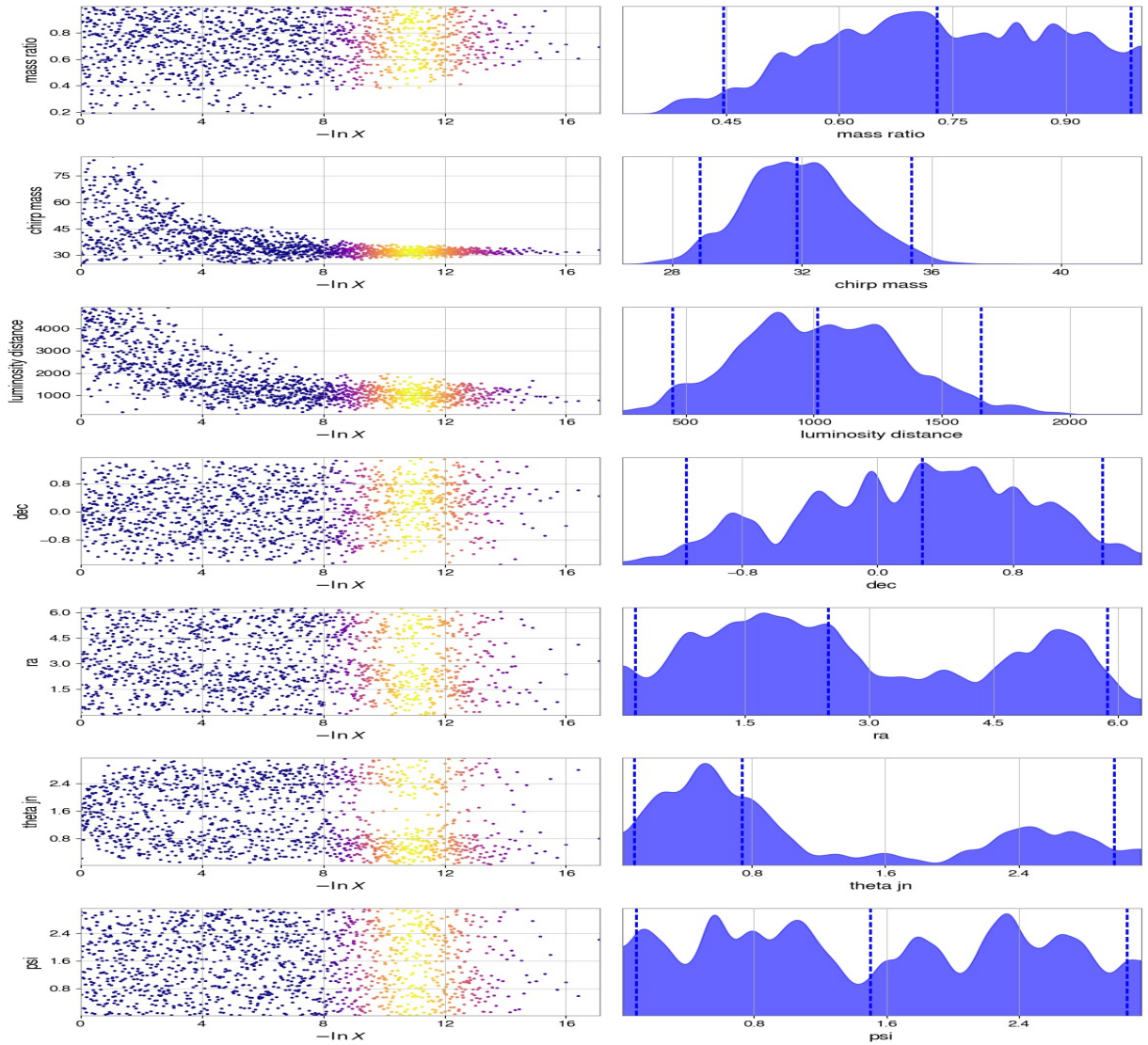


Bilby

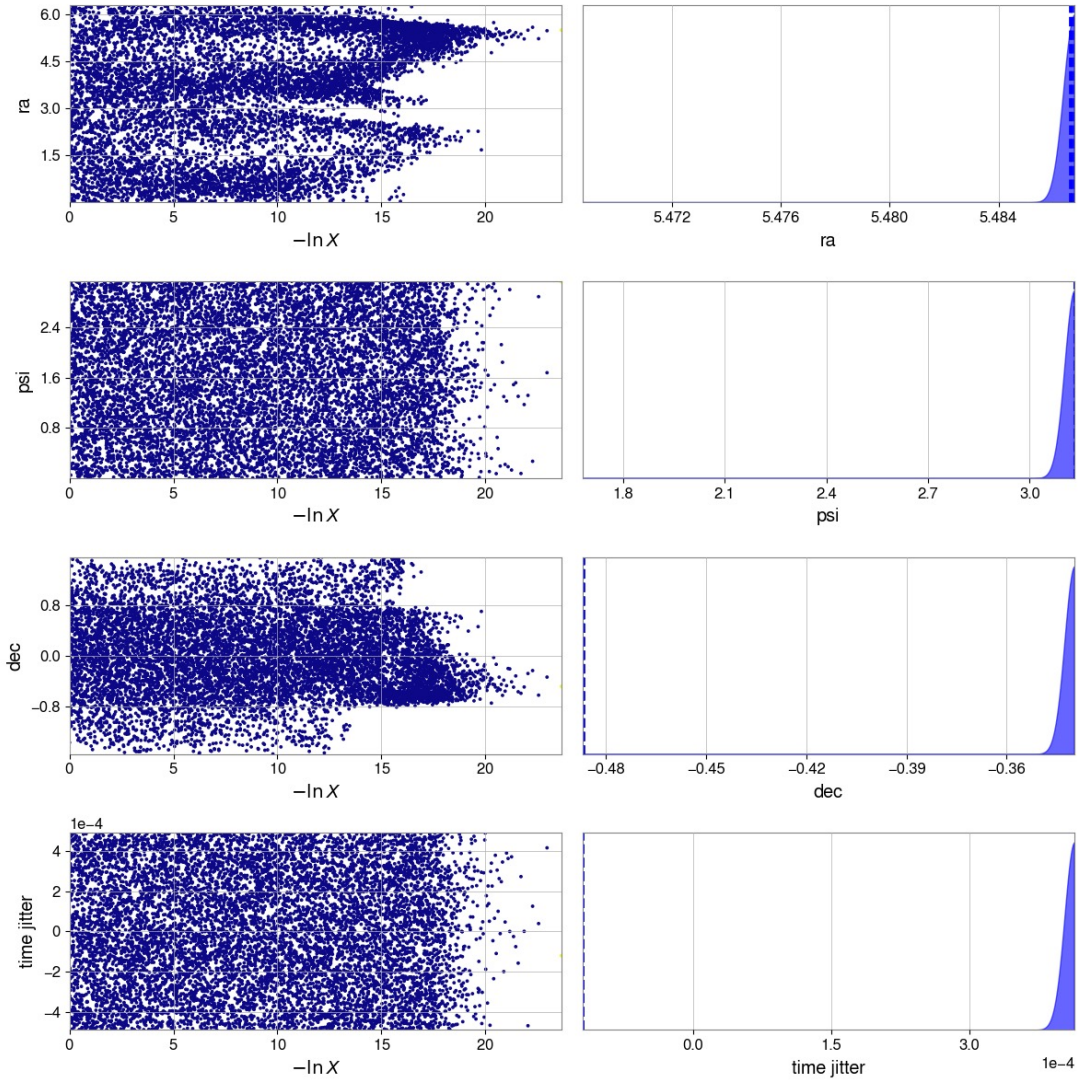

Live Points
400
200
0
0
3
6
9
12
15
18
21
−ln X
Likelihood (normalized)
0.8
0.4
0.0
Importance Weight PDF
100
50
1e301
Evidence
2
1

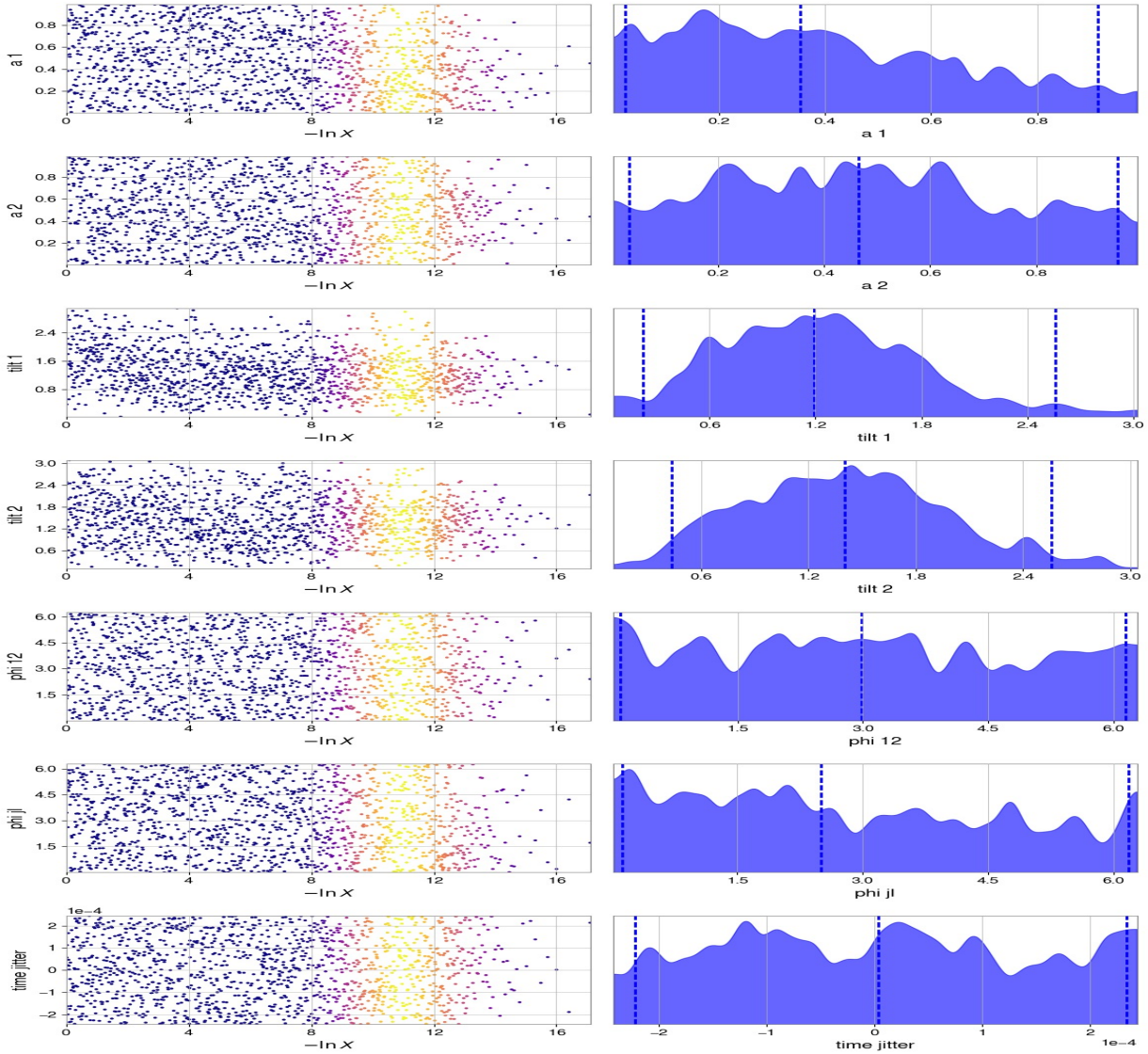
Bilby


chirp mass
−ln X
chirp mass
mass ratio
−ln X
mass ratio
luminosity distance
−ln X
luminosity distance
theta jn
−ln X
theta jn


mass ratio
−ln X
mass ratio
chirp mass
−ln X
chirp mass
luminosity distance
−ln X
luminosity distance
dec
−ln X
dec
ra
−ln X
ra
theta jn
−ln X
theta jn
psi
−ln X
psi

Bilby


ra
−ln X
ra
psi
−ln X
psi
dec
−ln X
dec
time jitter
−ln X
time jitter
1e−4

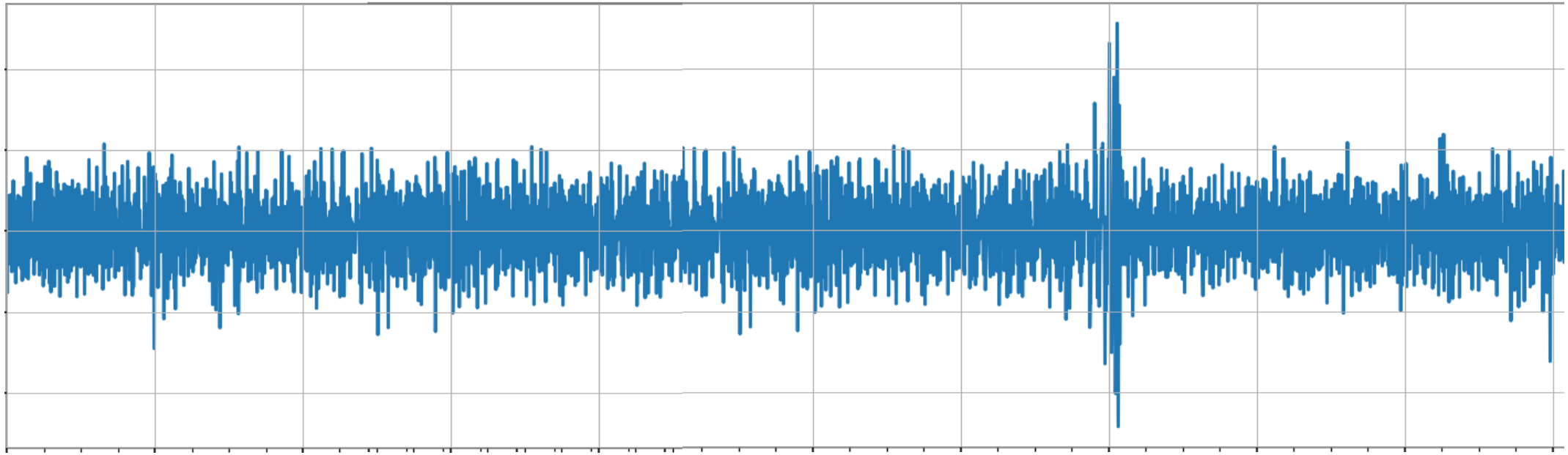

a 1
−ln X
a 1
a 2
−ln X
a 2
tilt 1
−ln X
tilt 1
tilt 2
−ln X
tilt 2
phi 12
−ln X
phi 12
phi jl
−ln X
phi jl
time jitter
−ln X
time jitter
1e−4

Q2-5. [30점] 중력파 모수추정 2

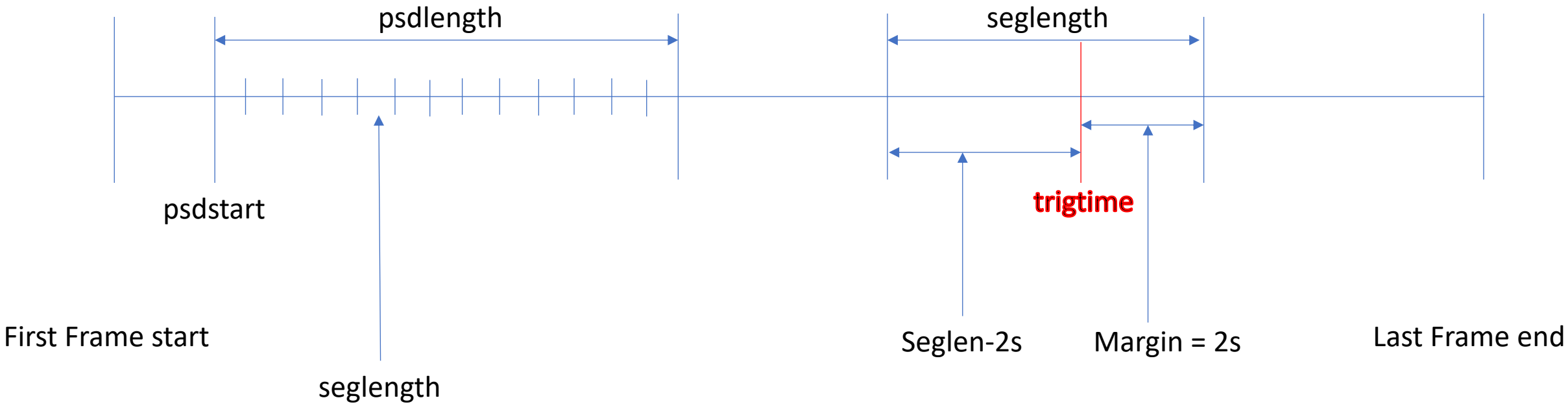
문제 5. 주어진 중력파 데이터 (GWNR_Prob2B_H1.gwf, GWNR_Prob2B_L1.gwf)로부터 SNR을 구하고, Bilby를 활용한 중력파 모수추정을 수행하여 여러매개 변수중 chirp mass, luminosity distance의 posterior의 Median 값을 출력하는 코드를 작성하고, chirpmass와 distance의 median값을 순서대로 출력한다. 또한 Bilby의 결과를 통해 얻은 Bayes factor를 입력한다.

- 소스파일 : GWProb_Q_NN.ipynb
- 출력파일 : GWQ5_A_NN.txt

```
f= open("GWQ5_A_NN.txt","w")
snr=float(input("SNR값을 입력하세요 : "))
mc_median = float(input("chirpmass의 median값을 입력하세요 : "))
distance_median = float(input("distance의 median값을 입력하세요 : "))
bayes_factor = float(input("Bilby결과에서 bayes_factor 입력하세요 : "))
f.write("{:.3f} {:.3f} {:.3f} {:.3f}" .format(snr,mc_median, distance_median,bayes_factor))
f.close()
```

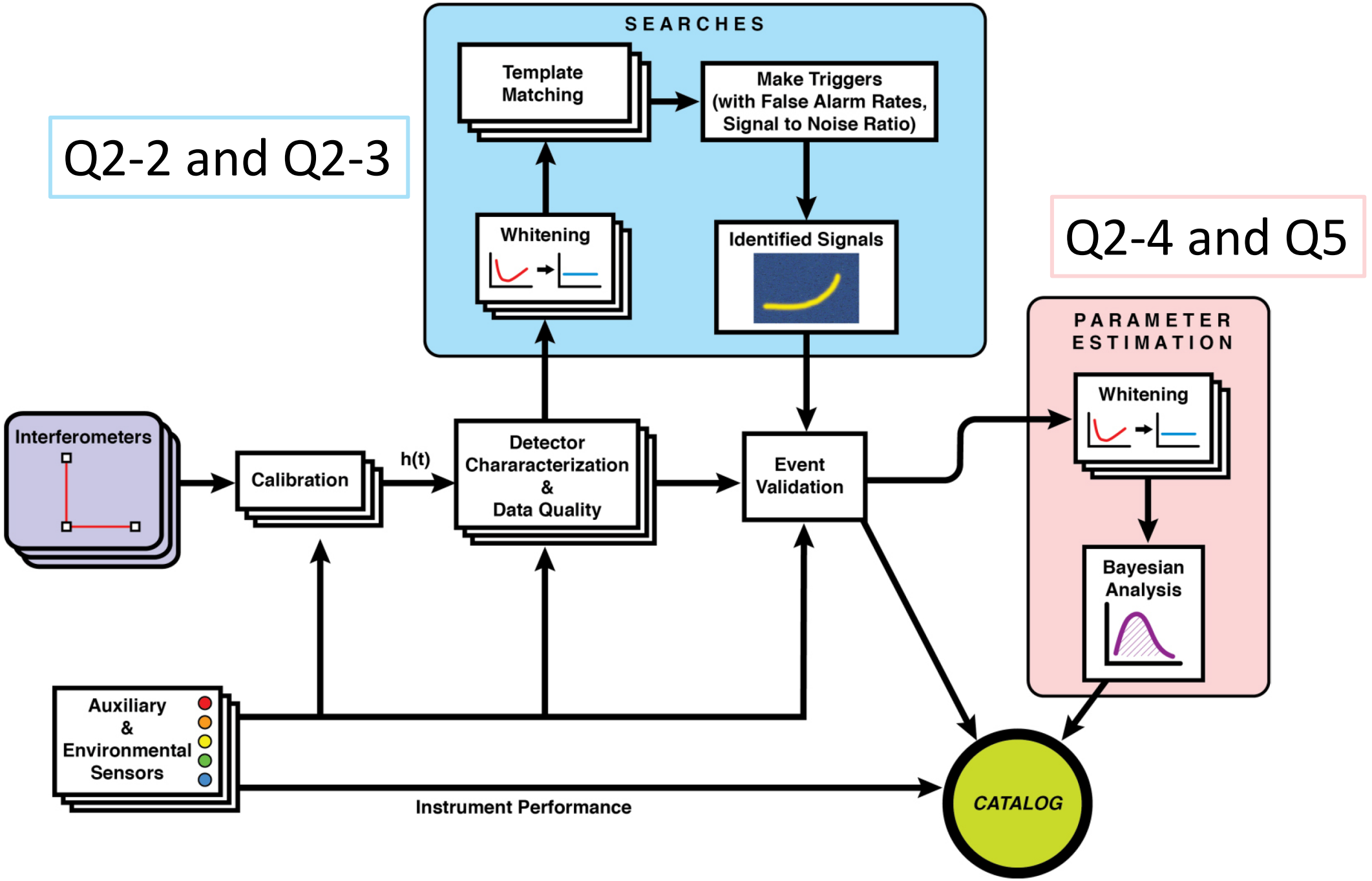


4096s


psdlength
seglength
psdstart
trigtime
First Frame start
seglength
Seglen-2s
Margin = 2s
Last Frame end

Q2-2 and Q2-3

Q2-4 and Q5



제출 유의사항

- jupyter hub 쓸 때 logout을 누르면 할당됐던 자원이 종료되어 버림. 팀원이 logout 눌러버리면 다른 팀원 작업 하던 내용이 날아갈 수 있음.
- 소스코드 `GWProb_Q_NN.ipynb`를 제출한다.
  - 소스코드를 제출하지 않는 경우 0점 처리한다.
- 특별히 적시한 내용이 없을시에는 모든 출력파일은 소수점 아래 셋째자리까지 출력한다.
- 출력파일은 다음의 파일명을 가진다.
  - `GWQ1_A_NN.txt`
  - `GWQ2_A_NN.txt`
  - `GWQ3_A_NN.txt`
  - `GWQ4_A_NN.txt`
  - `GWQ5_A_NN.txt`
- 소스코드 및 출력파일의 NN에는 자신의 팀 번호를 적는다.
- 소스코드 및 출력파일은 다음과 같이 git을 통해 제출한다.

참고문헌

- Pycbc : http://pycbc.org/pycbc/latest/html/index.html
- Bilby : https://lscsoft.docs.ligo.org/bilby/index.html
- Gwpy : https://gwpy.github.io/docs/stable/

참고 사이트

- 1. https://colab.research.google.com/
- 2. Search “gw-odw/odw-2022” in GitHub Tab.
  - Go ‘file’ tab > ‘Save a copy in Drive’

Gravitational Wave Open Data Workshop #6 (2023)

May 15-17, 2023

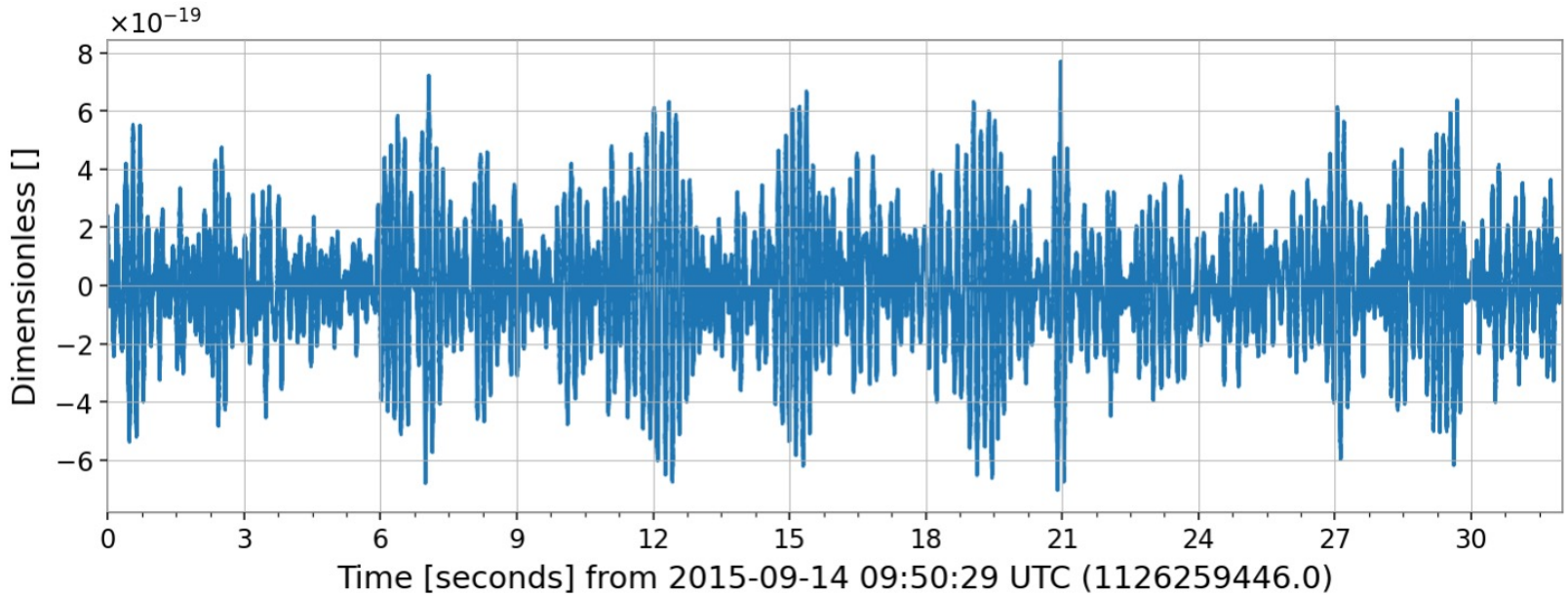
Planning in progress!

[1]
```
# -- Set a GPS time:
t0 = 1126259462.4    # -- GW150914

#-- Choose detector as H1, L1, or V1
detector = 'H1'
```

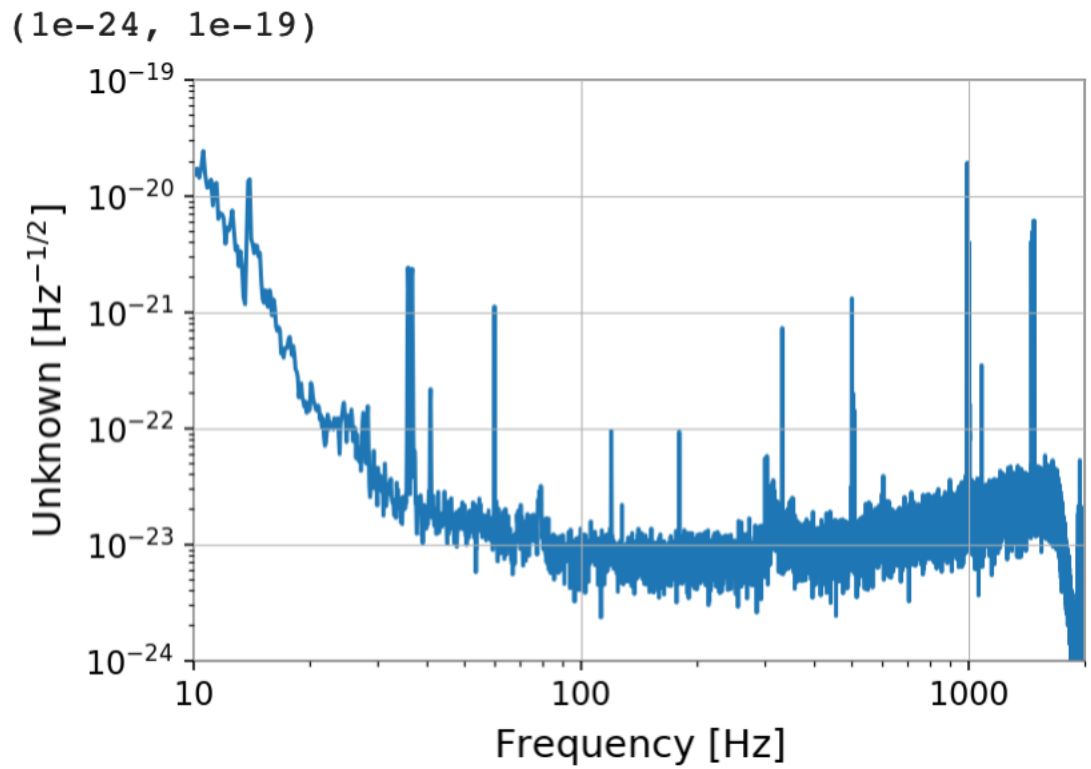
[3]
```
from gwosc.locate import get_urls
url = get_urls(detector, t0, t0)[-1]

print('Downloading: ' , url)
fn = os.path.basename(url)
with open(fn,'wb') as strainfile:
    straindata = requests.get(url)
    strainfile.write(straindata.content)
```

```
# -- Read strain data
strain = TimeSeries.read(fn,format='hdf5.losc')
center = int(t0)
strain = strain.crop(center-16, center+16)
fig1 = strain.plot()
```

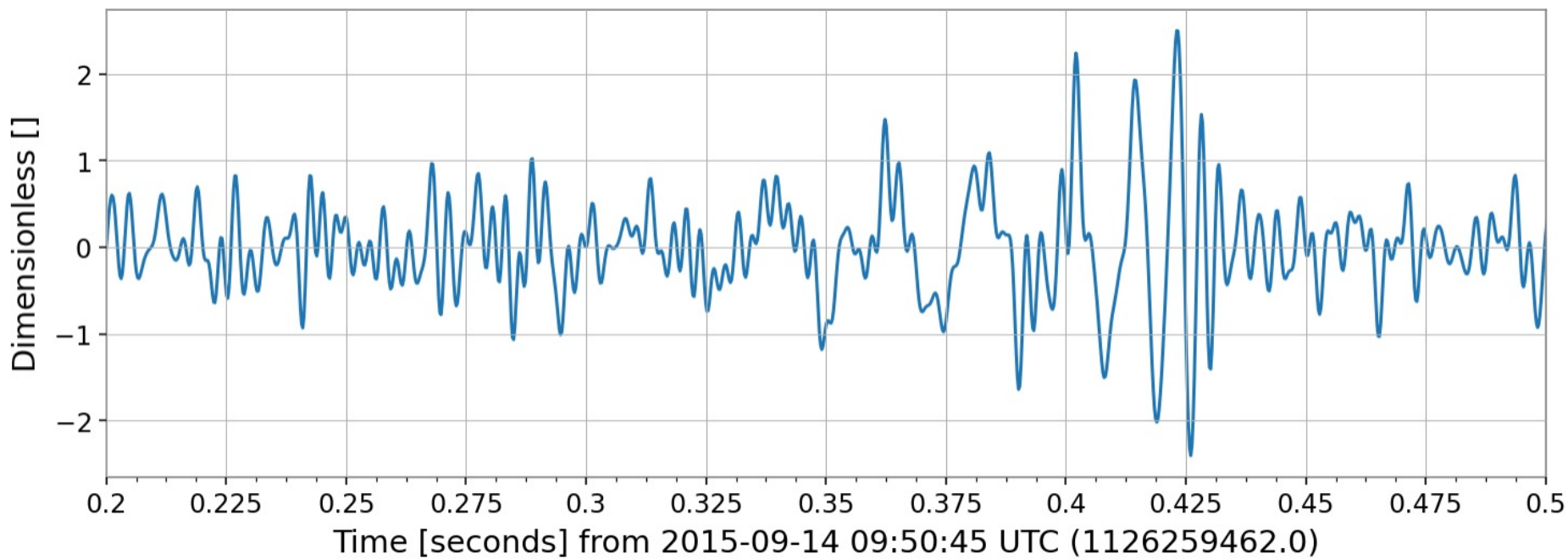
```
# -- Read strain data
strain = TimeSeries.read(fn,format='hdf5.losc')
center = int(t0)
strain = strain.crop(center-16, center+16)
fig1 = strain.plot()
```



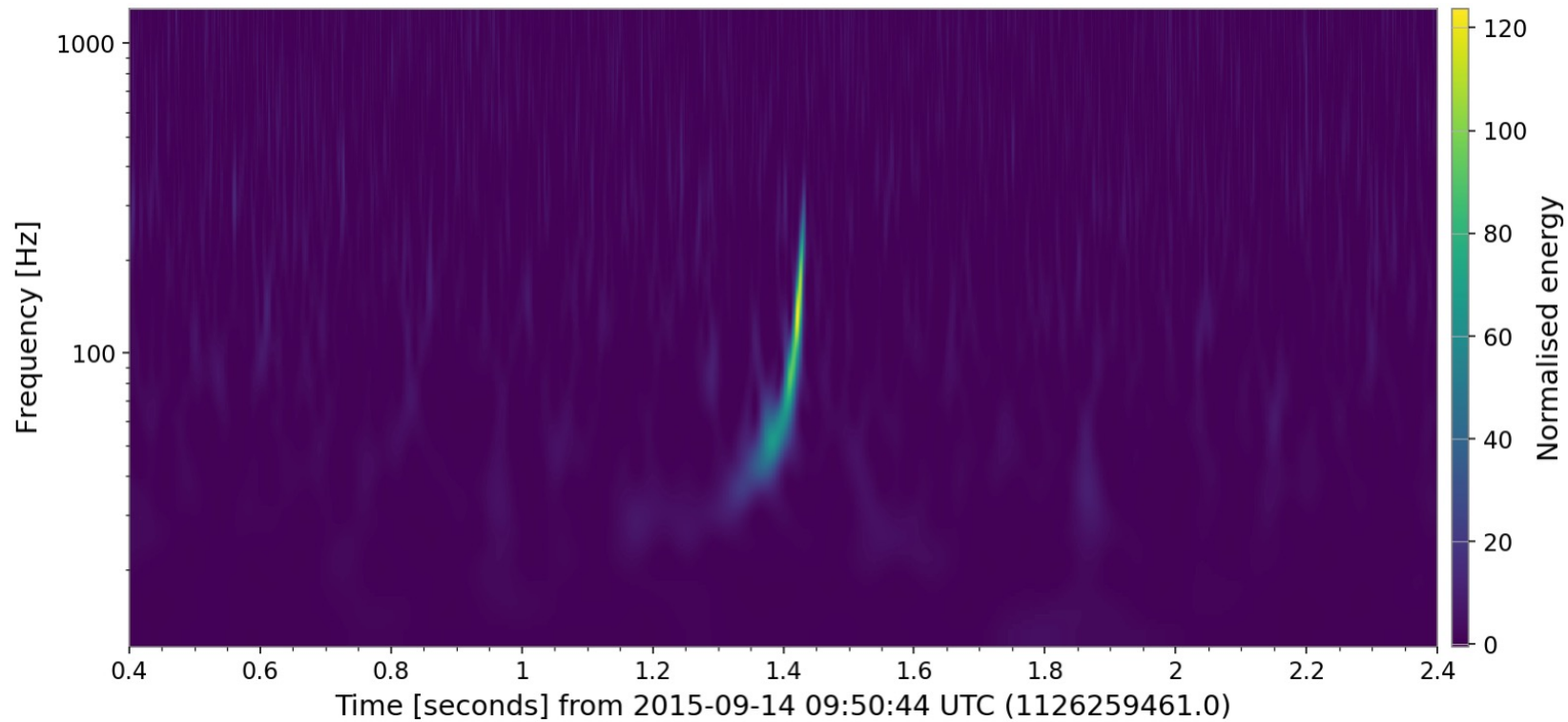
```
# -- Plot ASD
fig2 = strain.asd(fftlength=8).plot()
plt.xlim(10,2000)
plt.ylim(1e-24, 1e-19)
```



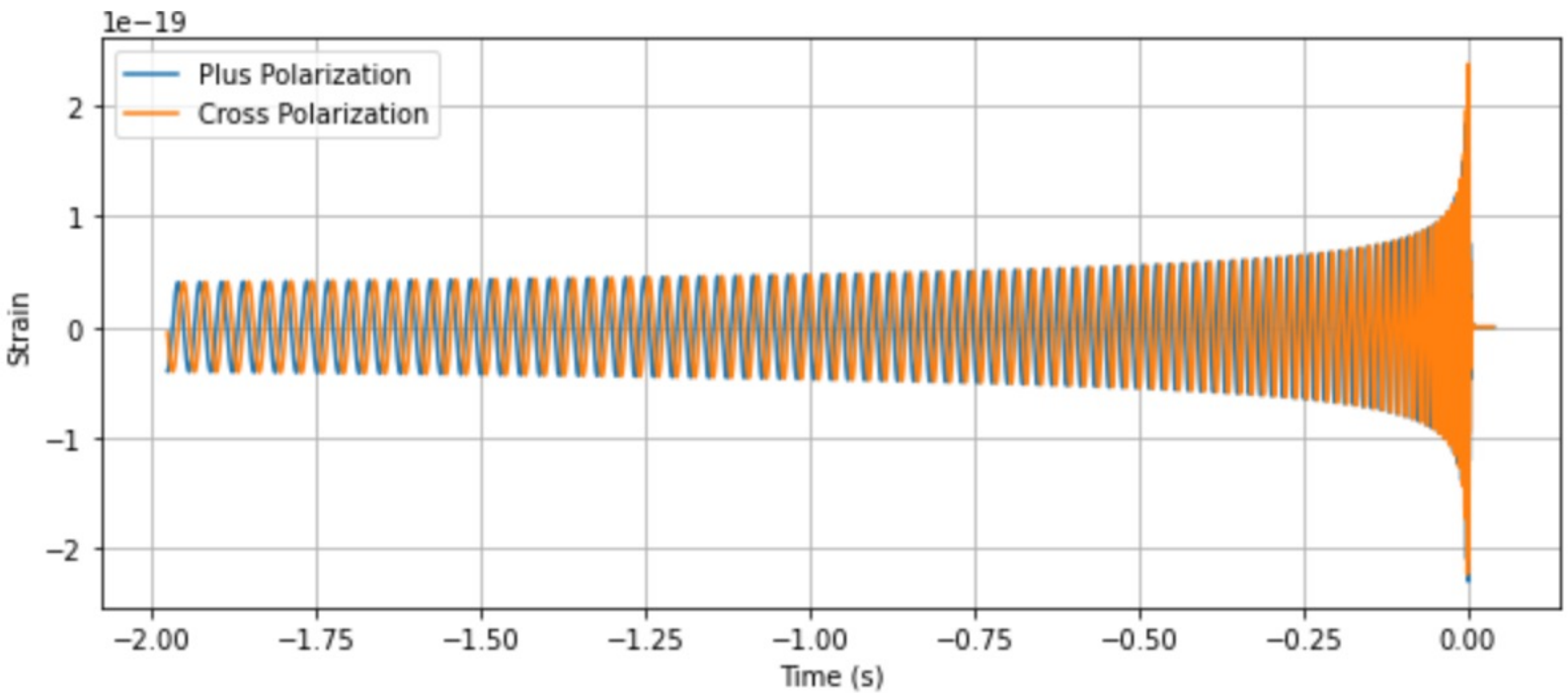
```
# -- Whiten and bandpass data
white_data = strain.whiten()
bp_data = white_data.bandpass(30, 400)
fig3 = bp_data.plot()
plt.xlim(t0-0.2, t0+0.1)
```



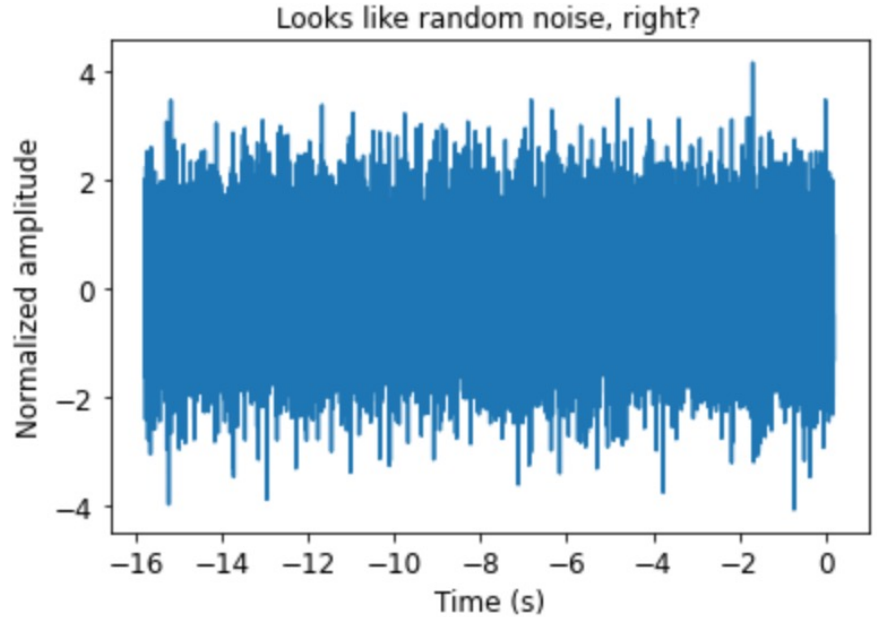
```
dt = 1  #-- Set width of q-transform plot, in seconds
hq = strain.q_transform(outseg=(t0-dt, t0+dt))
fig4 = hq.plot()
ax = fig4.gca()
fig4.colorbar(label="Normalised energy")
ax.grid(False)
ax.set_yscale('log')
```

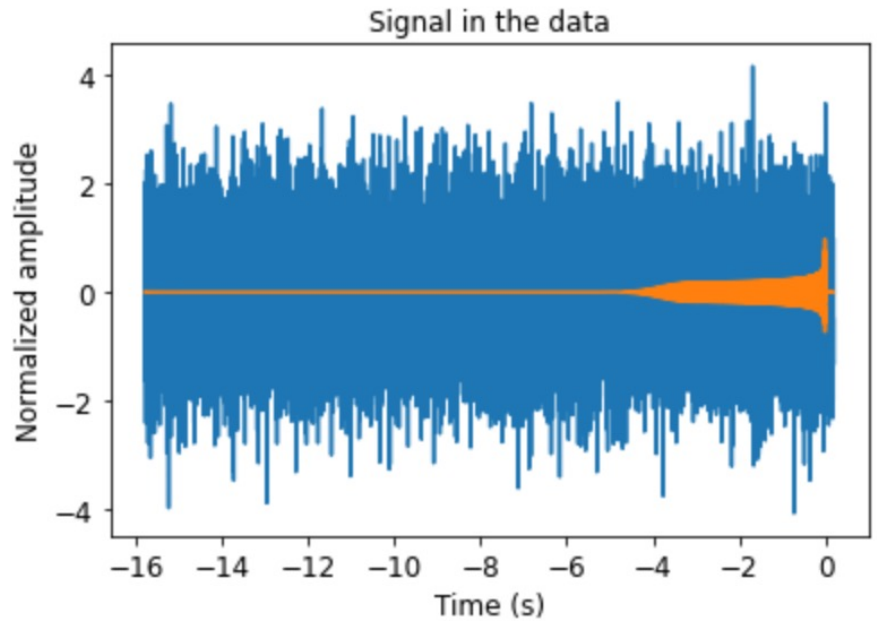


```
hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                         mass1=10,
                         mass2=10,
                         delta_t=1.0/16384,
                         f_lower=30)

pylab.figure(figsize=pylab.figaspect(0.4))
pylab.plot(hp.sample_times, hp, label='Plus Polarization')
pylab.plot(hp.sample_times, hc, label='Cross Polarization')
pylab.xlabel('Time (s)')
pylab.ylabel('Strain')
pylab.legend()
pylab.grid()
pylab.show()
```

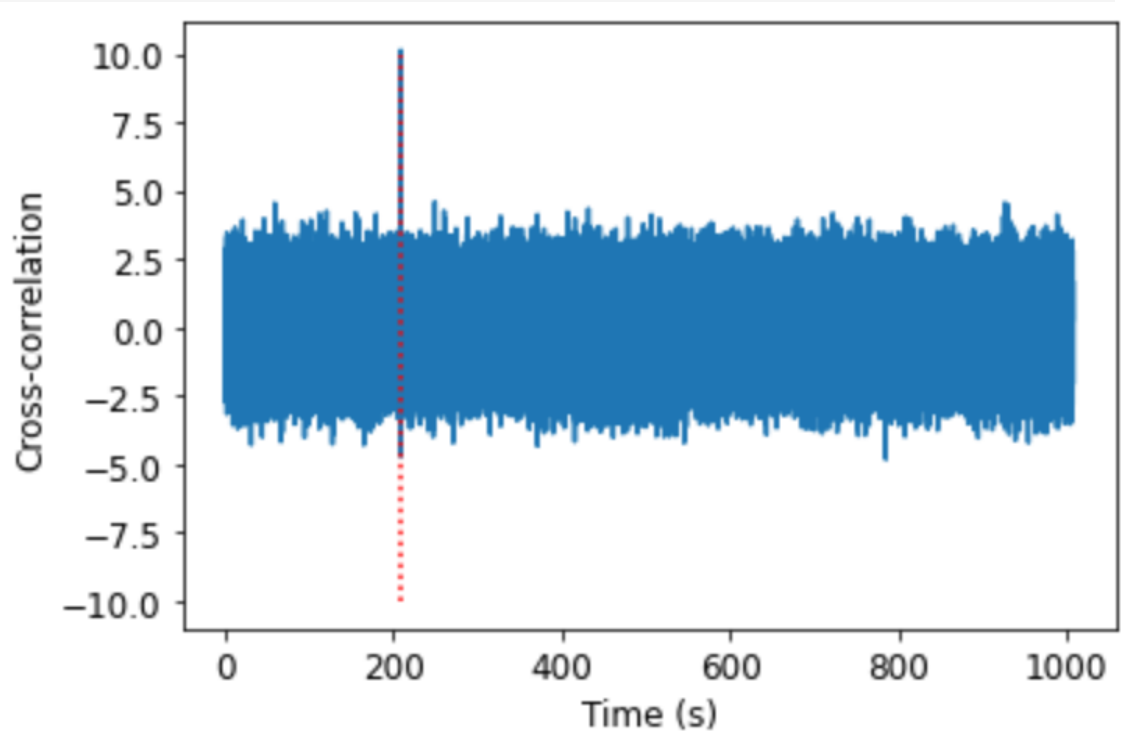


```
# Shift the waveform to start at a random time in the Gaussian noise dat
waveform_start = numpy.random.randint(0, len(data) - len(hp1))
data[waveform_start:waveform_start+len(hp1)] += 10 * hp1.numpy()

pylab.figure()
pylab.title("Looks like random noise, right?")
pylab.plot(hp1.sample_times, data[waveform_start:waveform_start+len(hp1)
pylab.xlabel('Time (s)')
pylab.ylabel('Normalized amplitude')

pylab.figure()
pylab.title("Signal in the data")
pylab.plot(hp1.sample_times, data[waveform_start:waveform_start+len(hp1)
pylab.plot(hp1.sample_times, 10 * hp1)
pylab.xlabel('Time (s)')
pylab.ylabel('Normalized amplitude')
```





```
%matplotlib inline
cross_correlation = numpy.zeros([len(data)-len(hp1)])
hp1_numpy = hp1.numpy()
for i in range(len(data) - len(hp1_numpy)):
    cross_correlation[i] = (hp1_numpy * data[i:i+len(hp1_numpy)]).sum()

# plot the cross-correlated data vs time. Superimpose the location of the end of the signal;
# this is where we should find a peak in the cross-correlation.
pylab.figure()
times = numpy.arange(len(data) - len(hp1_numpy)) / float(sample_rate)
pylab.plot(times, cross_correlation)
pylab.plot([waveform_start/float(sample_rate), waveform_start/float(sample_rate)], [-10,10],'r:')
pylab.xlabel('Time (s)')
pylab.ylabel('Cross-correlation')
```



Bilby – Download data

```
import bilby
from bilby.core.prior import Uniform
from bilby.gw.conversion import
convert_to_lal_binary_black_hole_parameters,
generate_all_bbh_parameters

from gwpy.timeseries import TimeSeries
```

```
* Download data

# Definite times in relation to the trigger time (time_of_event), duration and post_trigger_duratic
post_trigger_duration = 2
duration = 4
analysis_start = time_of_event + post_trigger_duration - duration

# Use gwpy to fetch the open data
H1_analysis_data = TimeSeries.fetch_open_data(
    "H1", analysis_start, analysis_start + duration, sample_rate=4096, cache=True)

L1_analysis_data = TimeSeries.fetch_open_data(
    "L1", analysis_start, analysis_start + duration, sample_rate=4096, cache=True)
```

Bilby – PSD download

* Set up empty interferometers

```
H1 = bilby.gw.detector.get_empty_interferometer("H1")
L1 = bilby.gw.detector.get_empty_interferometer("L1")

H1.set_strain_data_from_gwpy_timeseries(H1_analysis_data)
L1.set_strain_data_from_gwpy_timeseries(L1_analysis_data)
```

* Download the PSD data

```
psd_duration = duration * 32
psd_start_time = analysis_start - psd_duration

H1_psd_data = TimeSeries.fetch_open_data(
    "H1", psd_start_time, psd_start_time + psd_duration, sample_rate=4096, cache=True)

L1_psd_data = TimeSeries.fetch_open_data(
    "L1", psd_start_time, psd_start_time + psd_duration, sample_rate=4096, cache=True)
```

```
psd_alpha = 2 * H1.strain_data.roll_off / duration
H1_psd = H1_psd_data.psd(fftlength=duration, overlap=0, window=("tukey", psd_alpha), method="median")
L1_psd = L1_psd_data.psd(fftlength=duration, overlap=0, window=("tukey", psd_alpha), method="median")
```

Bilby – PSD

```
    ⁎Initialise the PSD

 H1.power_spectral_density = bilby.gw.detector.PowerSpectralDensity(
     frequency_array=H1_psd.frequencies.value, psd_array=H1_psd.value)
 L1.power_spectral_density = bilby.gw.detector.PowerSpectralDensity(
     frequency_array=H1_psd.frequencies.value, psd_array=L1_psd.value)


fig, ax = plt.subplots()
idxs = H1.strain_data.frequency_mask  # This is a boolean
mask of the frequencies which we'll use in the analysis
ax.loglog(H1.strain_data.frequency_array[idxs],

np.abs(H1.strain_data.frequency_domain_strain[idxs]))
ax.loglog(H1.power_spectral_density.frequency_array[idxs],
          H1.power_spectral_density.asd_array[idxs])
ax.set_xlabel("Frequency [Hz]")
ax.set_ylabel("Strain [strain/$\sqrt{Hz}$]")
plt.show()
```

Bilby - Prior

```
prior = bilby.core.prior.PriorDict()
prior['chirp_mass'] = Uniform(name='chirp_mass', latex_label='$M$', minimum=5., maximum=15., unit='$M_{\\odot}$')
prior['mass_ratio'] = Uniform(name='mass_ratio', minimum=0.9, maximum=1.1)
prior['phase'] = Uniform(name="phase", minimum=0, maximum=2*np.pi)
prior['geocent_time'] = Uniform(name="geocent_time", minimum=time_of_event_est-0.5, maximum=time_of_event_est+0.5)
prior['a_1'] =  0.0
prior['a_2'] =  0.0
prior['tilt_1'] =  0.0
prior['tilt_2'] =  0.0
prior['phi_12'] =  0.0
prior['phi_jl'] =  0.0
prior['luminosity_distance'] = bilby.gw.prior.UniformSourceFrame(name='luminosity_distance',
                                                                 minimum=40, maximum=1000)
prior['theta_jn'] = Sine(name='theta_jn')
prior['ra'] = Uniform(name='ra', minimum=0, maximum=2 * np.pi, boundary='periodic')
prior['psi'] = Uniform(name='psi', minimum=0, maximum=np.pi, boundary='periodic')
prior['dec'] = Cosine(name='dec')
```

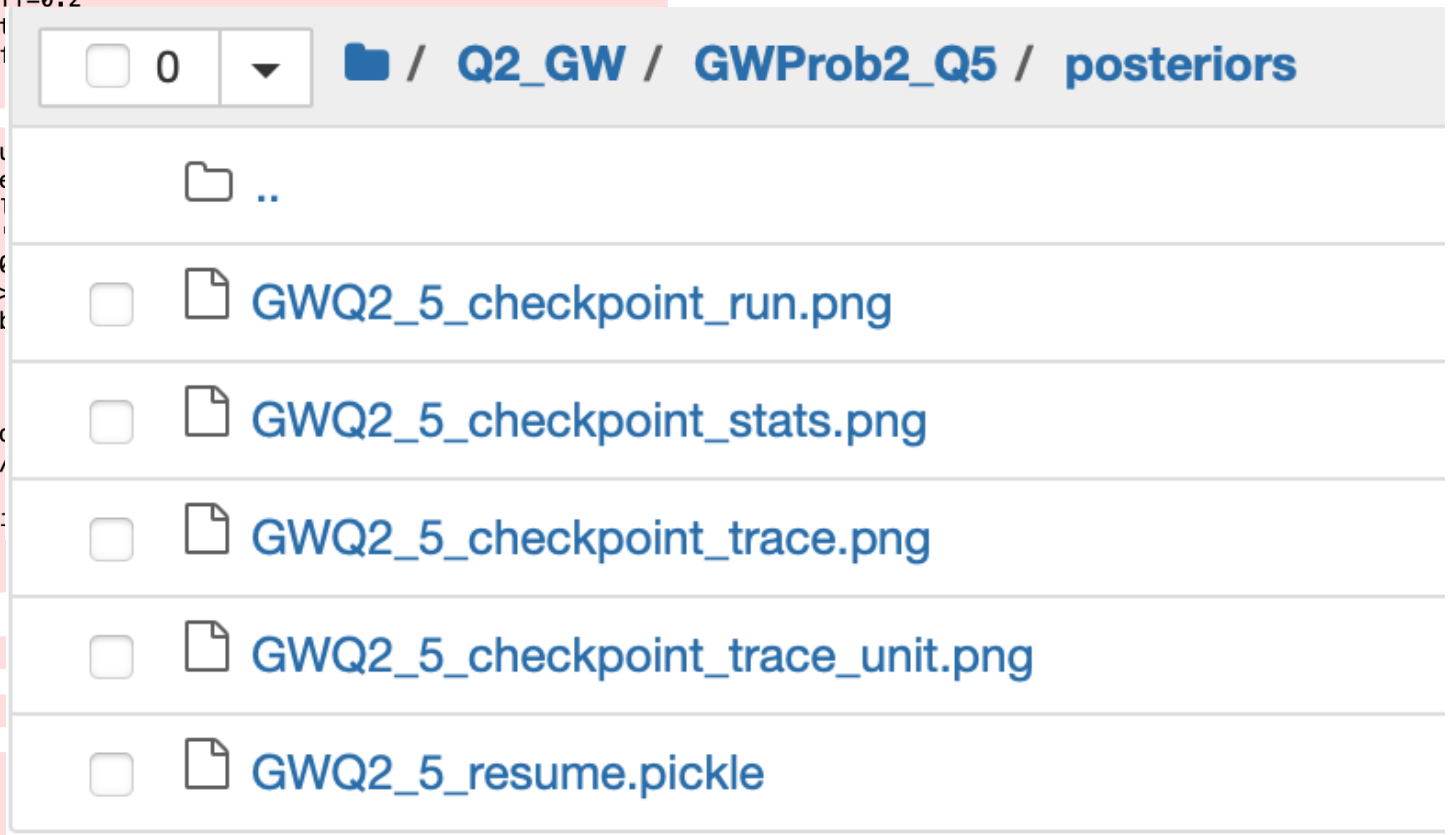
Bilby - likelihood

```
# Finally, create our likelihood, passing in what is needed to get going
likelihood = bilby.gw.likelihood.GravitationalWaveTransient(
    interferometers, waveform_generator, priors=prior,
    time_marginalization=True, phase_marginalization=True,
distance_marginalization=False)
```

Bilby - run

```
result = bilby.run_sampler(
    likelihood, prior, sampler='dynesty', outdir='./GWProb2_Q5/posteriors', label="GWQ2_5",
    conversion_function=bilby.gw.conversion.generate_all_bbh_parameters, nlive = 500, dlogz=3)
```

```
08:26 bilby INFO    : Generating frequency domain strain from given time domain strain.
08:26 bilby INFO    : Applying a tukey window with alpha=0.1, roll off=0.2
08:26 bilby INFO    : Generating frequency domain strain from given t
08:26 bilby INFO    : Applying a tukey window with alpha=0.1, roll of
08:26 bilby INFO    : Single likelihood evaluation took 8.877e-03 s

08:26 bilby INFO    : Using sampler Dynesty with kwargs {'bound': 'mu
iodic': None, 'reflective': None, 'check_point_delta_t': 1800, 'nlive
pdim': None, 'rstate': None, 'queue_size': 1, 'pool': None, 'use_pool
e, 'logl_kwargs': None, 'ptform_args': None, 'ptform_kwargs': None,
.5, 'vol_check': 8.0, 'facc': 0.2, 'slices': 5, 'update_interval': 30
func of <bilby.core.sampler.dynesty.Dynesty object at 0x2b702327aaf0>
e, 'logl_max': inf, 'add_live': True, 'print_progress': True, 'save_b
5000, 'nact': 5, 'print_method': 'tqdm'}
08:26 bilby INFO    : Checkpoint every check_point_delta_t = 600s
08:26 bilby INFO    : Using dynesty version 1.0.1
08:26 bilby INFO    : Using the bilby-implemented rwalk sample method
08:26 bilby INFO    : Reading resume file ./GWProb2_Q5_v2/posteriors/
08:26 bilby INFO    : Resume file successfully loaded.
08:33 bilby INFO    : Written checkpoint file ./GWProb2_Q5_v2/poster
12:56 bilby INFO    : Sampling time: 0:00:08.084629
12:56 bilby INFO    : Reconstructing marginalised parameters.

  0%|          | 0/1413 [00:00<?, ?it/s]
12:56 bilby INFO    : Generating sky frame parameters.
  0%|          | 0/1413 [00:00<?, ?it/s]
12:56 bilby INFO    : Computing SNRs for every sample.
  0%|          | 0/1413 [00:00<?, ?it/s]
12:57 bilby INFO    : Summary of results:
nsamples: 1413
ln_noise_evidence: -8534.562
ln_evidence: -8266.386 +/-  0.131
ln_bayes_factor: 268.175 +/-  0.131
```

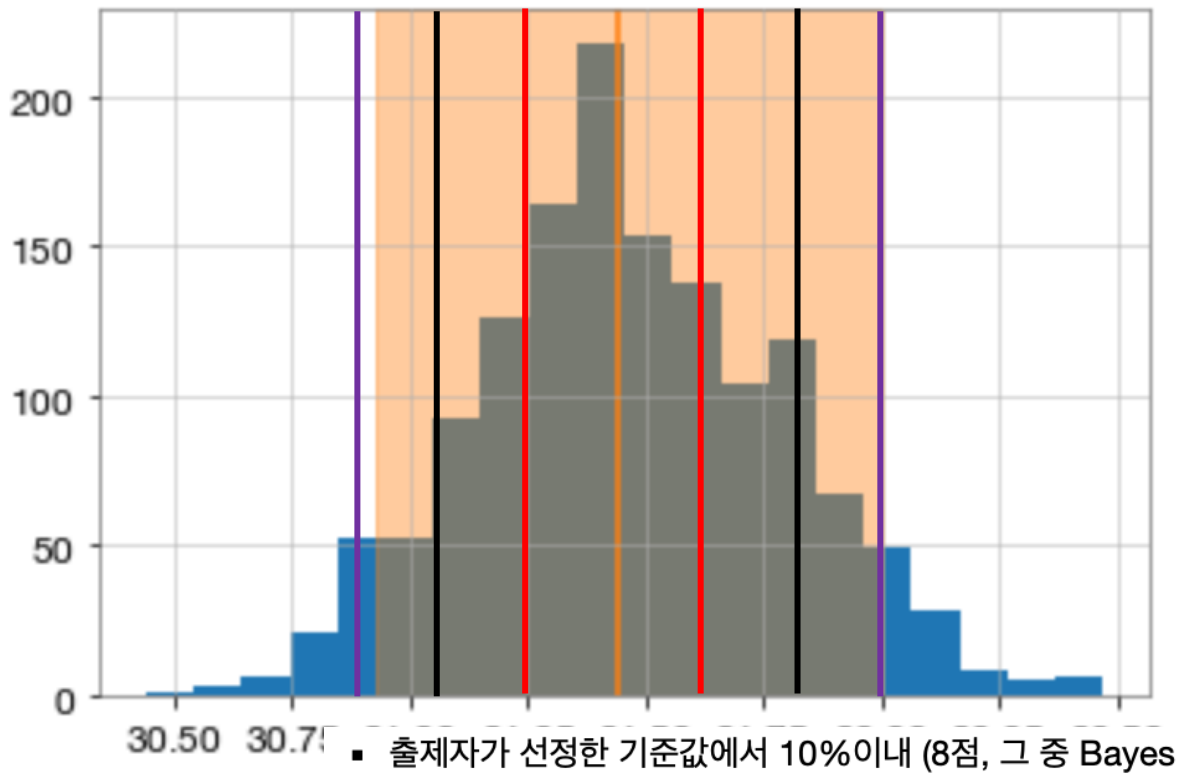


모수추정 채점기준

30% 이내

20% 이내

10% 이내



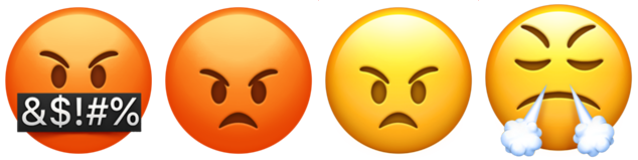
- 출제자가 선정한 기준값에서 10%이내 (8점, 그 중 Bayes factor가 큰 순서대로 +2,+1씩 차등 점수를 부여한다.
  - 즉, 10점(1팀),9점(1팀),8점(그 외)
- 출제자가 선정한 기준값에서 20%이내 (5점, 그 중 Bayes factor가 큰 순서대로 +2,+1씩 차등 점수를 부여한다.
  - 즉, 7점(1팀),6점(1팀),5점(그 외)
- 출제자가 선정한 기준값에서 30%이내 (2점, 그 중 Bayes factor가 큰 순서대로 ,+2,+1씩 차등 점수를 부여한다.
  - 즉, 4점(1팀),3점(1팀),2점(그 외)

How to read data

Read local file :

```
ifo_list = bilby.gw.detector.InterferometerList([])
ifo = bilby.gw.detector.get_empty_interferometer(IFO)
ifo.strain_data.set_from_frame_file(file, channel name, sampling_frequency,
                                    start_time, duration)
ifo_list.append(ifo)
```

Bilby




```
TypeError: read() got an unexpected keyword argument 'resample'
```