2021 NRGW winter school @ 2022.01.17~21

계산천체물리 문제 I: 중력파데이터분석

김영민 (UNIST)

ymkim715@gmail.com
ymkim715@unist.ac.kr

데이터 세트

1. BBH1 : time segment = 1240642018 - 1240643042
   - BBH1-H1.gwf, BBH1-L1.gwf
2. BBH2 : time segment = 1240641118 - 1240642142
   - BBH2-H1.gwf, BBH2-L1.gwf
3. BNS1 : time segment = 1262492106 - 1262493130
   - BNS1-H1.gwf, BNS1-L1.gwf
4. BNS2 : time segment = 1262492018 - 1262493042
   - BNS2-H1.gwf, BNS1-L1.gwf

Channel name:
“H1:HWINJ_INJECTED” for H1
“L1:HWINJ_INJECTED” for L1

How to read data

Pycbc

```
Read local file:
pycbc.frame.read_frame(file, channel_name)
```

Bilby

```
ifo_list = bilby.gw.detector.InterferometerList([])
ifo = bilby.gw.detector.get_empty_interferometer('H1')
ifo.strain_data.set_from_frame_file(frame_file=h1gwf,
                                    channel=h1channel,
                                    sampling_frequency=4096,
                                    start_time=gps_start_time,
                                    duration=duration)
ifo_list.append(ifo)
```

블랙홀 쌍성병합 문제

문제 **1: Signal Search [15점]**
주어진 시계열 데이터 별로 SNR > 15인 중력파 신호들을 찾고, event time, mass parameters (M1, M2), luminosity distance를 찾는 프로그램을 작성하시오.

- 소스파일: prob1.py
- 출력파일: output1.txt

– 첫번째 행에는 BBH1에 대한 4개의 숫자 (event time, M1, M2, luminosity distance 순서로)
– 두번째 행에는 BBH2에 대한 4개의 숫자 (event time, M1, M2, luminosity distance 순서로)

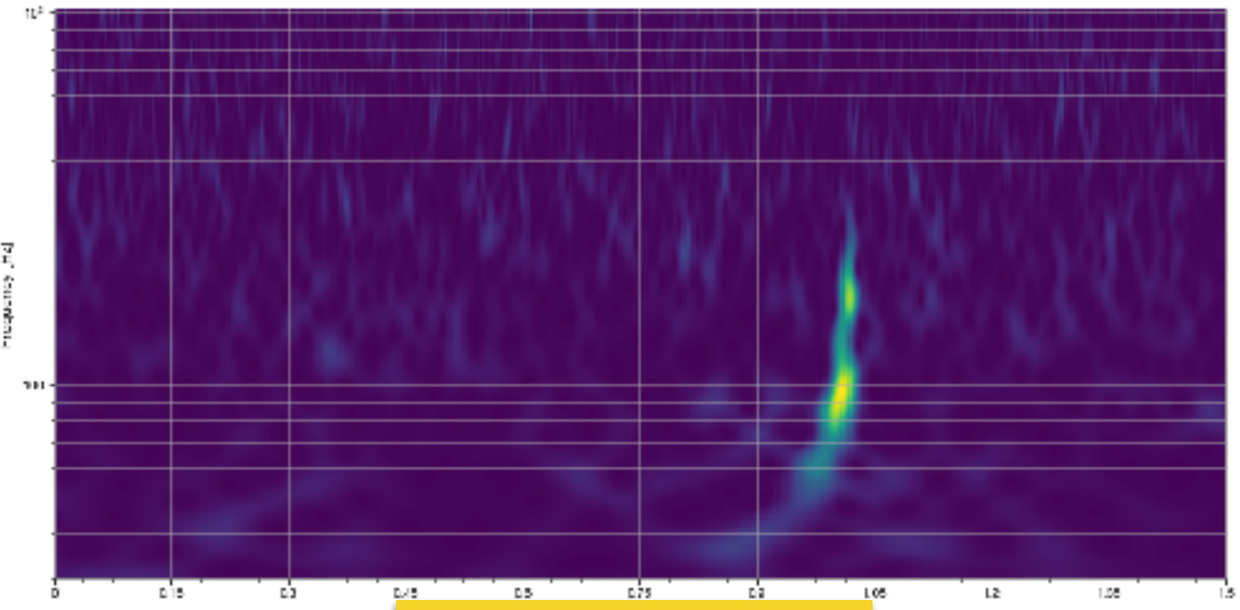
문제 **2: Parameter Estimation [15점]**
BBH2에 대해 문제 1에서 찾은 event time 주변에서 모수추정을 하시오. M1, M2, luminosity distance 의 posterior samples 을 이용하여 50%와 90% credible region을 찾는 프로그램을 작성하시오.

- 소스파일: prob2.py
- 출력파일: output2.txt

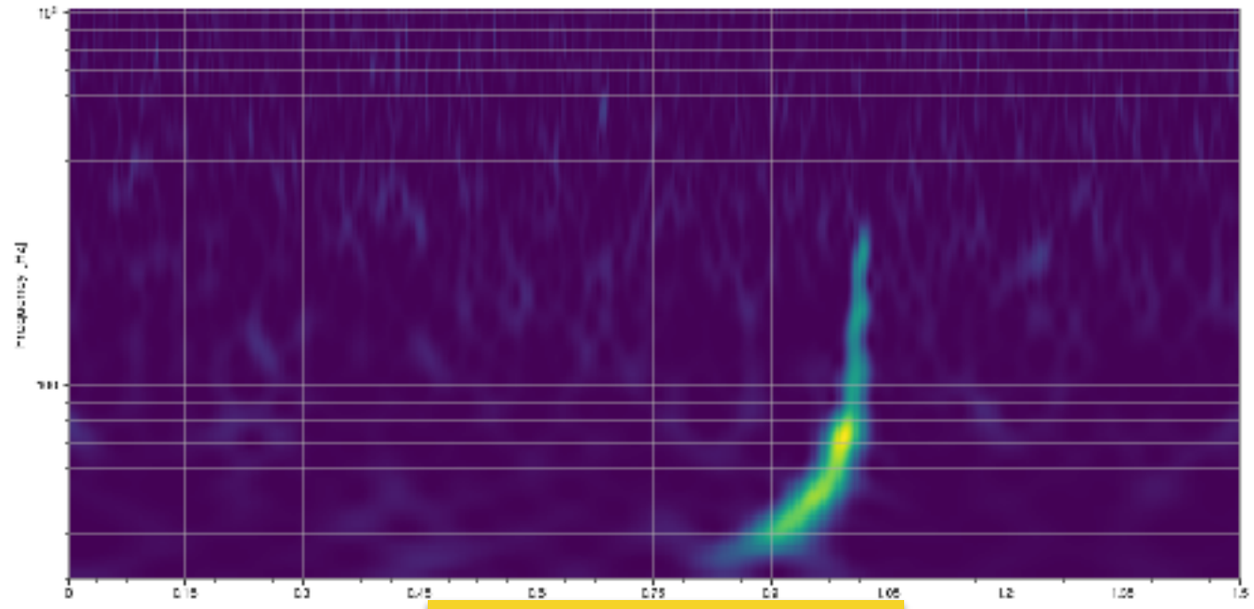
– 첫번째 행에는 50% credible region에 대한 6개의 숫자 (M1, M2, luminosity distance)
– 두번째 행에는 90% credible region에 대한 6개의 숫자 (M1, M2, luminosity distance)

Hint: M_i=[10,100] Msun, Luminosity distance=[400,2000] Mpc

BBH1

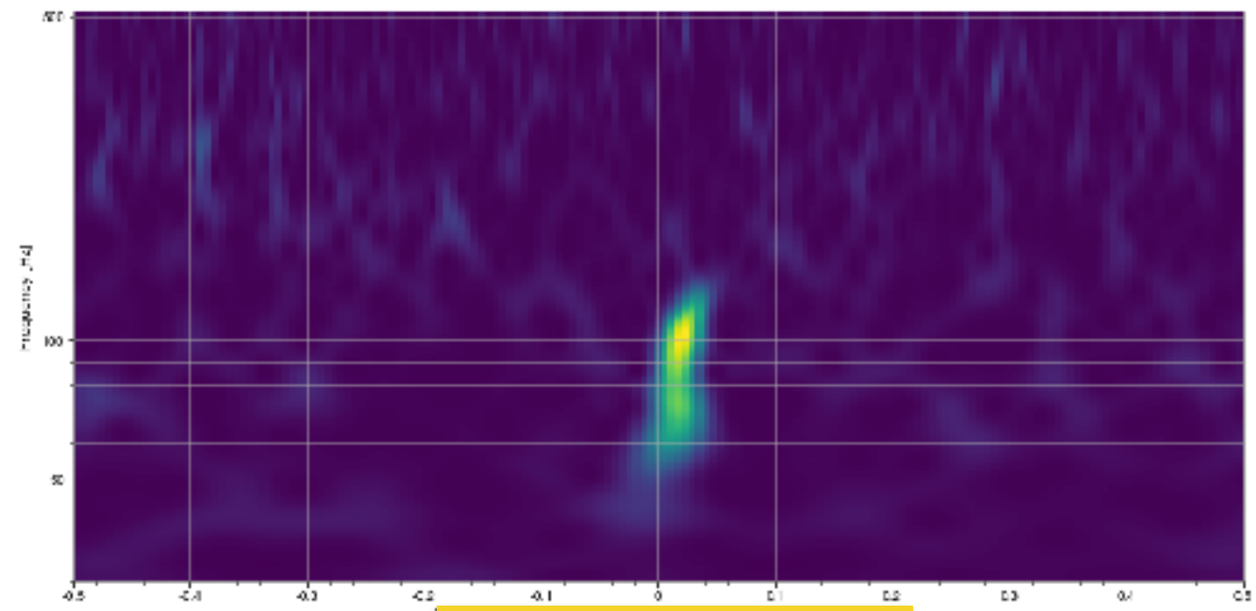


H1

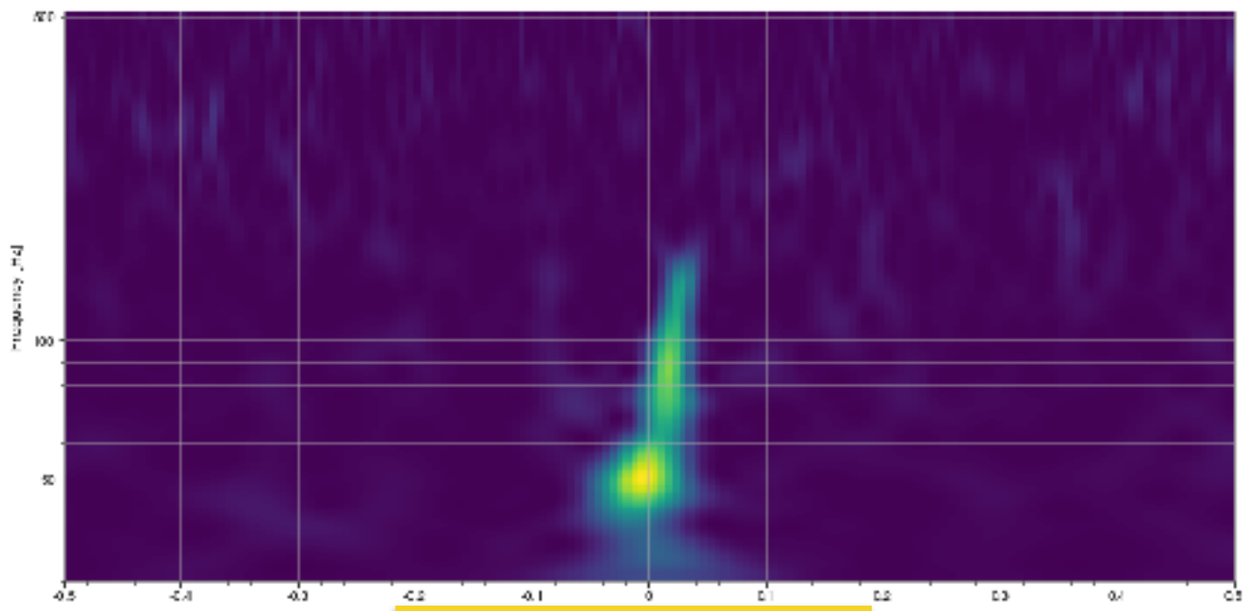


L1

BBH2



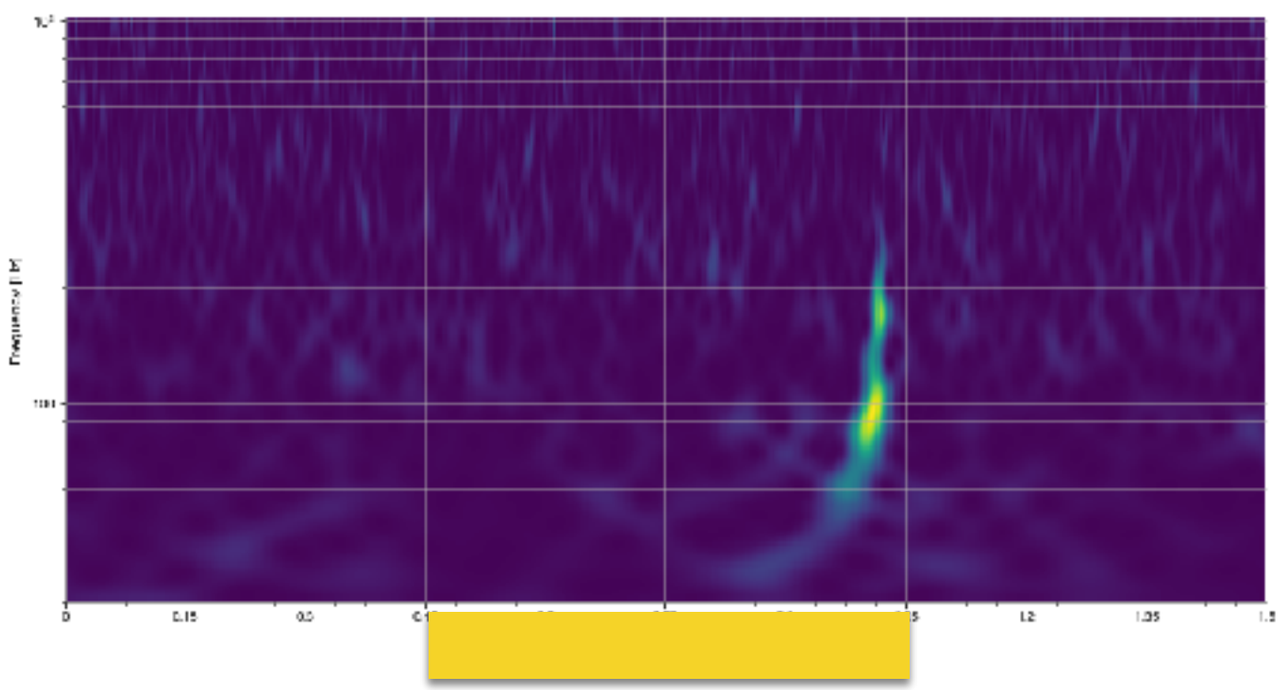
H1



L1

Spectrogram (Q-transform)

```
from gwpy.timeseries import TimeSeries
data = TimeSeries.read(‘BBH1-H1.gwf','H1:HWINJ_INJECTED',
                                start=GPS_START_TIME,end=GPS_END_TIME)
qspecgram = data.q_transform(qrange=(10,90),frange=(30, 1024),
                                        outseg=(start_time,end_time))
plot=qspecgram.plot(figsize=[16,8])
ax=plot.gca()
ax.set_xscale('seconds')
ax.set_yscale('log')
ax.set_ylim(30,1024)
ax.set_xlim(event_time-1.,event_time+0.5)
ax.set_ylabel('Frequency [Hz]')
ax.grid(True,axis='y', which=‘both')
ax.colorbar(cmap=‘viridis',
        label='Normalized energy')
plot.show()
plot.savefig(‘BBH1-H1.png')
```



중성자별 쌍성병합 문제 (I)

문제 **3: Signal Search [15**점**]**
주어진 시계열 데이터 별로 SNR > 15인 중력파 신호들을 찾고, event time, mass parameters (M1, M2), tidal deformabilities(Λ1, Λ2), luminosity distance를 찾는 프로그램을 작성하시오.

- 소스파일: prob3.py
- 출력파일: output3.txt
  - 첫번째 행에는 BNS1에 대한 6개의 숫자 (event time, M1 , M2 , Λ1 , Λ2 , luminosity distance)
  - 두번째 행에는 BNS2에 대한 6개의 숫자 (event time, M1 , M2 , Λ1 , Λ2 , luminosity distance)

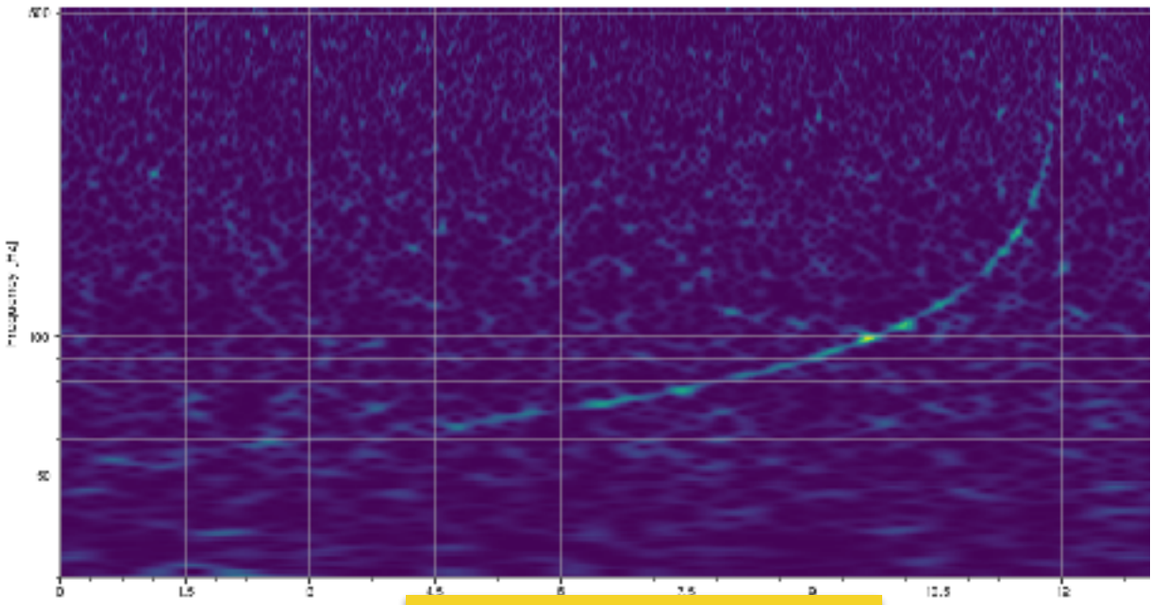
문제 **4: Parameter Estimation [15**점**]**

BNS1에 대해 문제 3에서 찾은 event time 주변에서 모수추정을 하시오. M1, M2, Λ1, Λ2, luminosity distance 의 posterior samples 을 이용하여 50%와 90% credible region을 찾는 프로그램을 작성하 시오.

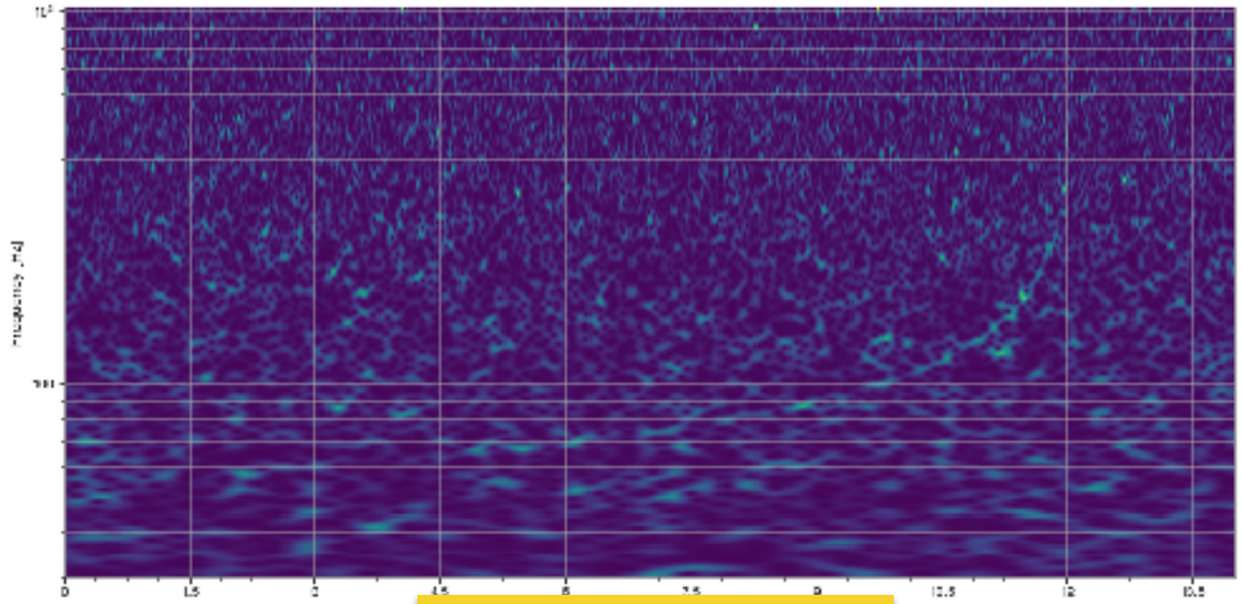
- 소스파일: prob4.py
- 출력파일: output4.txt
  - 첫번째 행에는 50% credible region에 대한 10개의 숫자 (M1, M2, Λ1, Λ2, luminosity distance)
  - 두번째 행에는 90% credible region에 대한 10개의 숫자 (M1, M2, Λ1, Λ2, luminosity distance)

Hint: M_i=[1,5] Msun, Λ_i=[10,800], D_L=[30,80] Mpc

BNS1

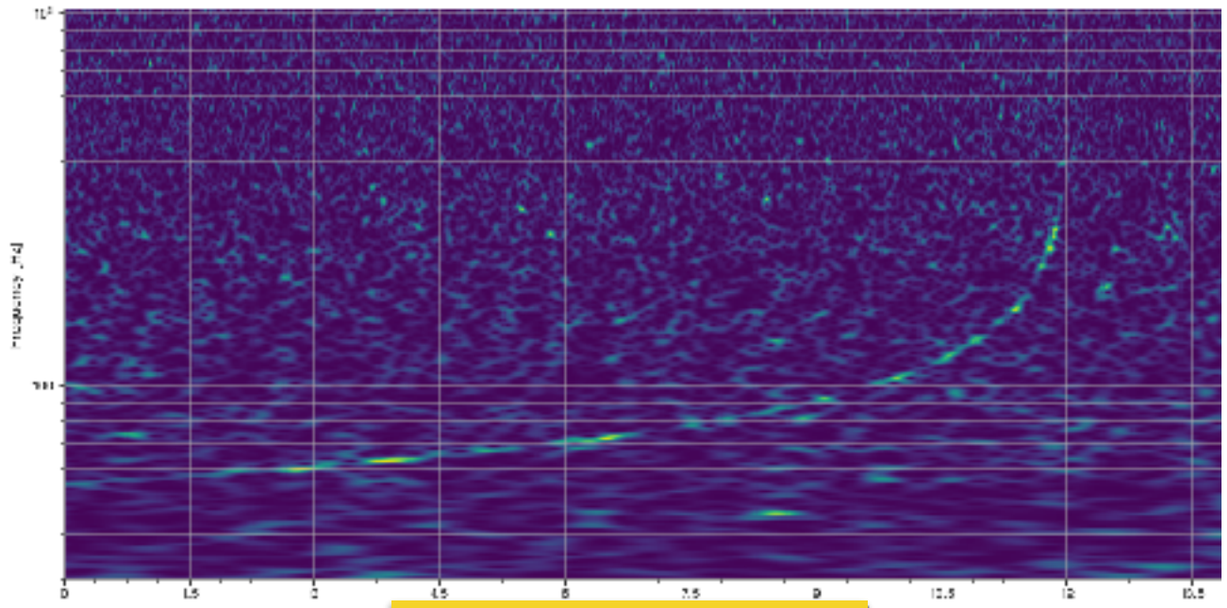


H1

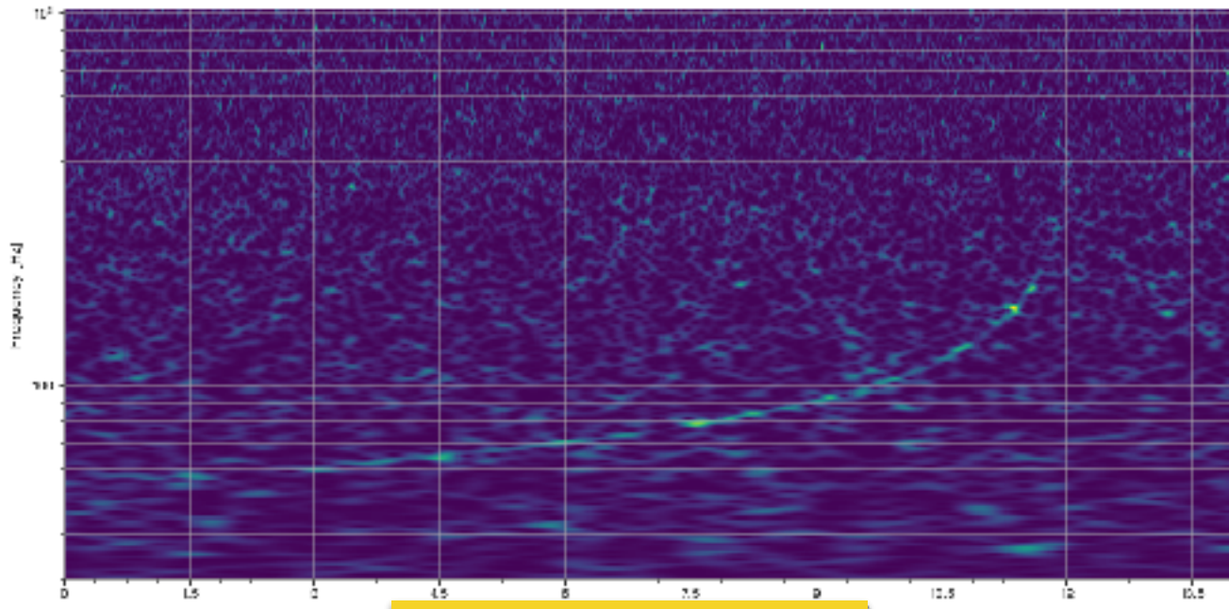


L1

BNS2



H1



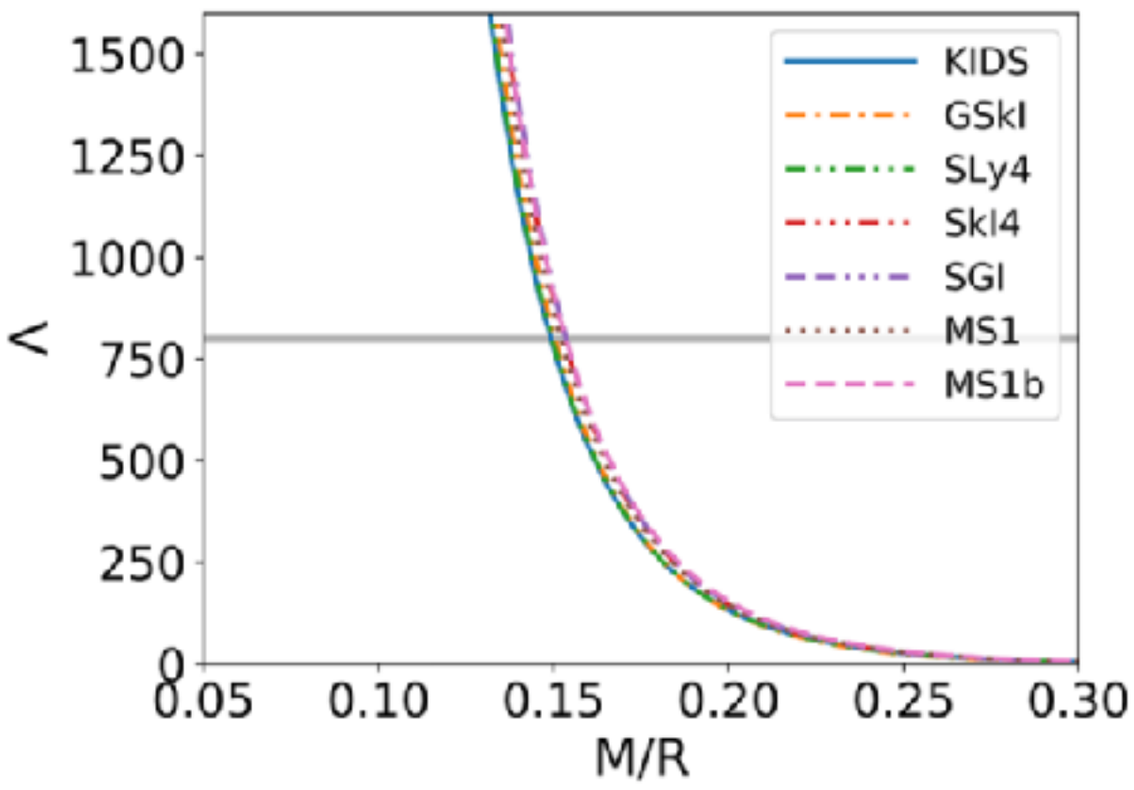
L1

중성자별 쌍성병합 문제 (2)

문제 **5: Radius estimation in BNS [10점]**
문제 4에서 찾은 중성자별 쌍성 신호에서 산출한 (M1, Λ1), (M2, Λ2) posterior samples 들로 부터 Λ–C 관계식을 이용하여 중성자별 반경을 산출하여, (M1, R1), (M2, R2)의 50%, 90% credible region을 찾 는 프로그램을 작성하시오.

- 소스파일: prob5.py
- 출력파일: output5.txt

– 첫번째 행에는 50% credible region에 대한 8개의 숫자 (M1, R1, M2, R2 순서로)
– 두번째 행에는 90% credible region에 대한 8개의 숫자 (M1, R1, M2, R2 순서로)



Insensitive to EoS

K. Yagi and N. Yunes, Phys. Rep. 681 (2017) 1

$$C = a_0 + a_1(\ln\Lambda) + a_2(\ln\Lambda)^2$$

a_0=0.360, a_1= - 0.0355, a_2= 0.000705

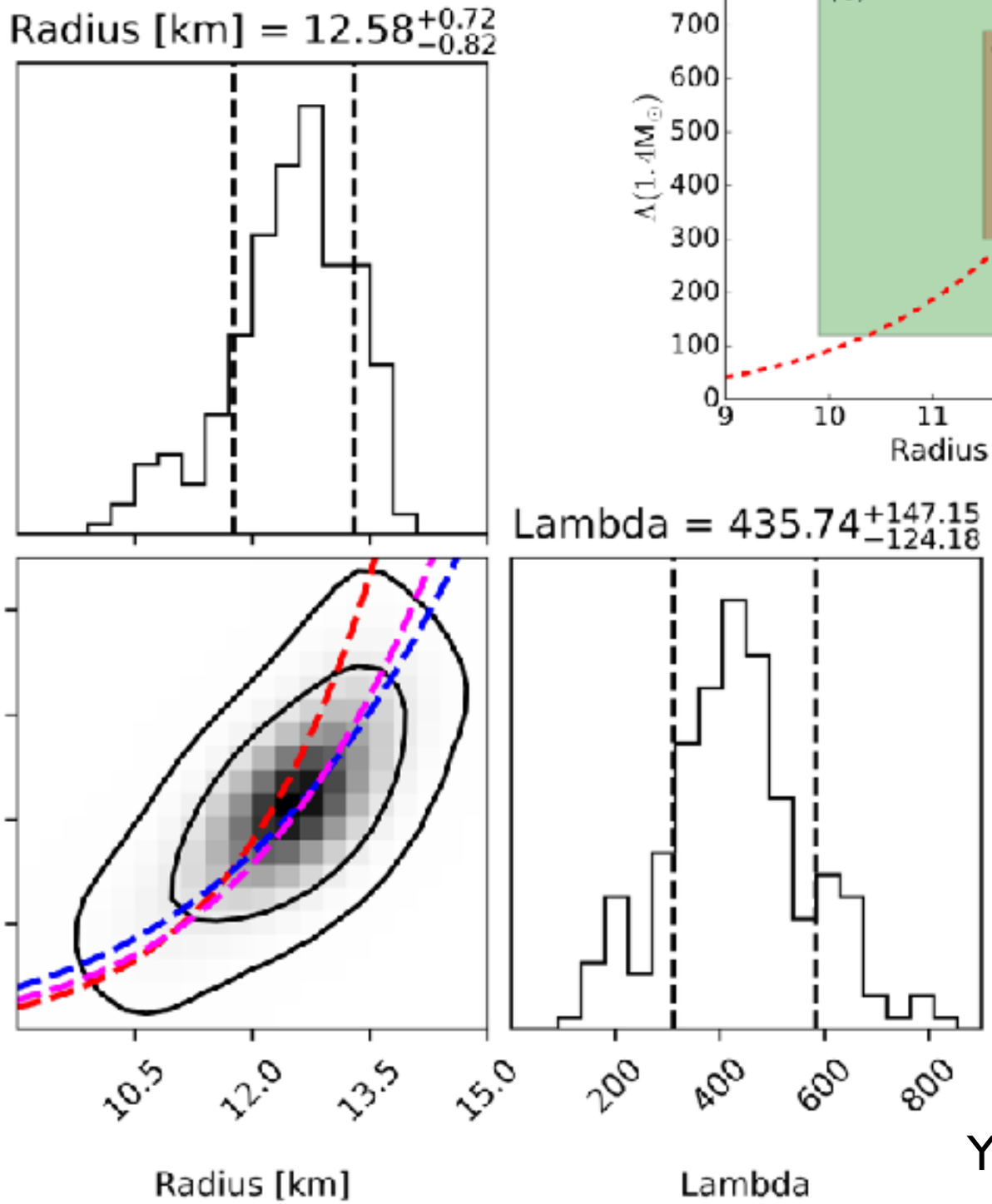
C=GM/Rc^2

중성자별 쌍성병합 문제 (3)

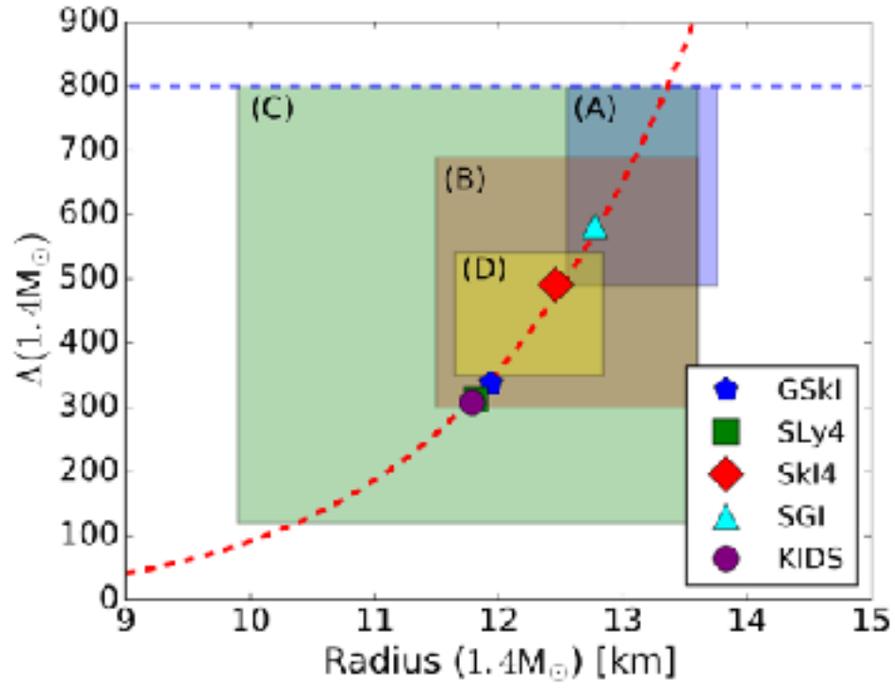
문제 **6: Relation of Radius and Tidal deformability in BNS [15점]**
최근 중성자별 상태방정식 연구들에서는 중성자별 질량 1.4M⊙일때, 조력변형성(Λ)과 반경(R)의 관계가 power law ($\Lambda \approx R^{\alpha}$)를 따름을 보여주고 있다. 문제5에서 구한 결과를 이용하여 α를 산출하고, 50%, 90% credible region을 찾는 프로그램을 작성하시오.(Bilby를 이용하여 직접 likelihood 함수를 구성하 여 결과를 산출하시오.)

- 소스파일: prob6.py
- 출력파일: output6.txt
  - 첫번째 행에는 50% credible region에 대한 2개의 숫자
  - 두번째 행에는 90% credible region에 대한 2개의 숫자

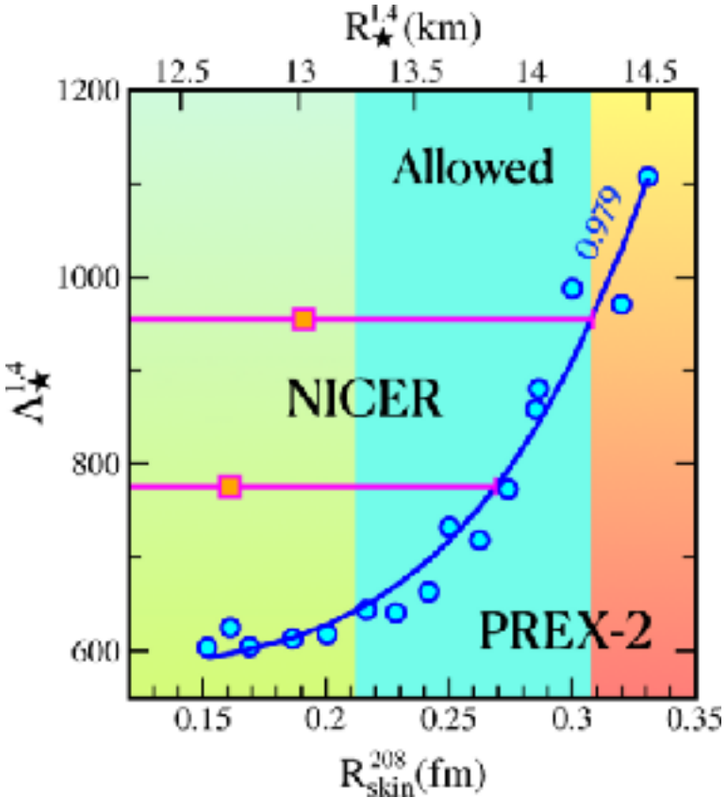
Lambda-Radius relation



Kim et al., PhysRevC.98.065805 (2018)



Implication of PREX 2 - PhysRevLett.126.172503 (2021)



Red line: $\Lambda(1.4M\odot) = 2.88 * 10^{-6}\ (R/km)^{7.5}$

[C] PhysRevLett.120.172703.pdf

Blue line: $\Lambda(1.4M\odot) = 1.35 * 10^{-3}\ (R/km)^{5.0}$

Implication of PREX 2 - PhysRevLett.126.172503 (2021)

=> $\Lambda \sim R^{4.8}$

Magenta line: $\Lambda(1.4M\odot) = 1.05 * 10^{-4}\ (R/km)^{6.0}$

Y.-M. Kim, in progress

중성자별 쌍성병합 문제 (4)

문제 **7: Challenge [15**점**]**

폴리트로프 상태방정식 ($P = K\varrho^{\Gamma}$)을 이용하여 중성자별의 질량, 반경, 조력변형성을 이론적으로 산출해 볼수 있다. 폴리트로프 상태방정식을 이용하여, 위에서 구한 (M,Λ) 또는 (M,R) 로부터 likelihood function을 직접 구성하여, (K, Γ) posterior samples 을 구하고, 50% credible region을 구하는 프로그램을 작성하시오. (Bilby를 이용하여 직접 likelihood 함수를 구성하여 결과를 산출하시오. LALSIMULATION, Bilby에서 제공하는 TOV solver 사용가능.)

- 소스파일: prob7.py
- 출력파일: output7.txt

– 첫번째 행에는 50% credible region에 대한 4개의 숫자 (K, Γ 순서로)

– 두번째 행에는 90% credible region에 대한 4개의 숫자 (K, Γ 순서로)

Hint: lalsimulation, bilby에선 상태방정식 테이블을 geometric unit(G=c=1)을 사용한다. 직접상태방정식을 만들때는 밀도, 압력을 geometric unit에서 m 단위로 바꿔 사용해야 함. lalsimulation은 MKS 단위 사용.

Polytropic equation of state

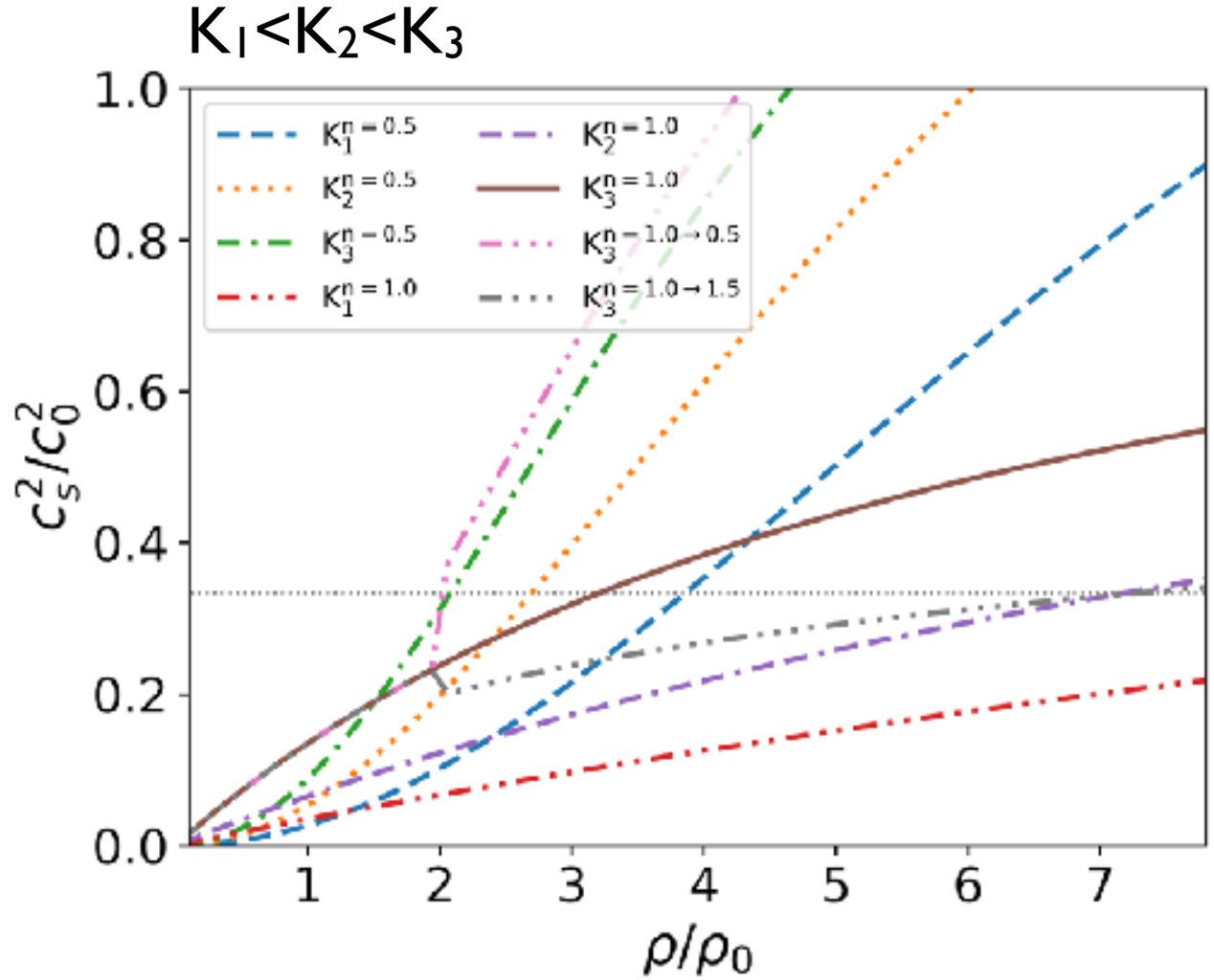
$$p = K\rho^{\Gamma} = K\rho^{1+1/n}$$

$$\epsilon = (1+a)\rho c_0^2 + \frac{p}{\Gamma - 1}$$

$$= (1+a)\rho c_0^2 + \frac{K}{\Gamma - 1}\rho^{\Gamma}.$$

(a=0)

$$\frac{c_s^2}{c_0^2} = \frac{\mathrm{d}p}{\mathrm{d}\epsilon} = \Gamma\frac{p}{\epsilon + p}$$

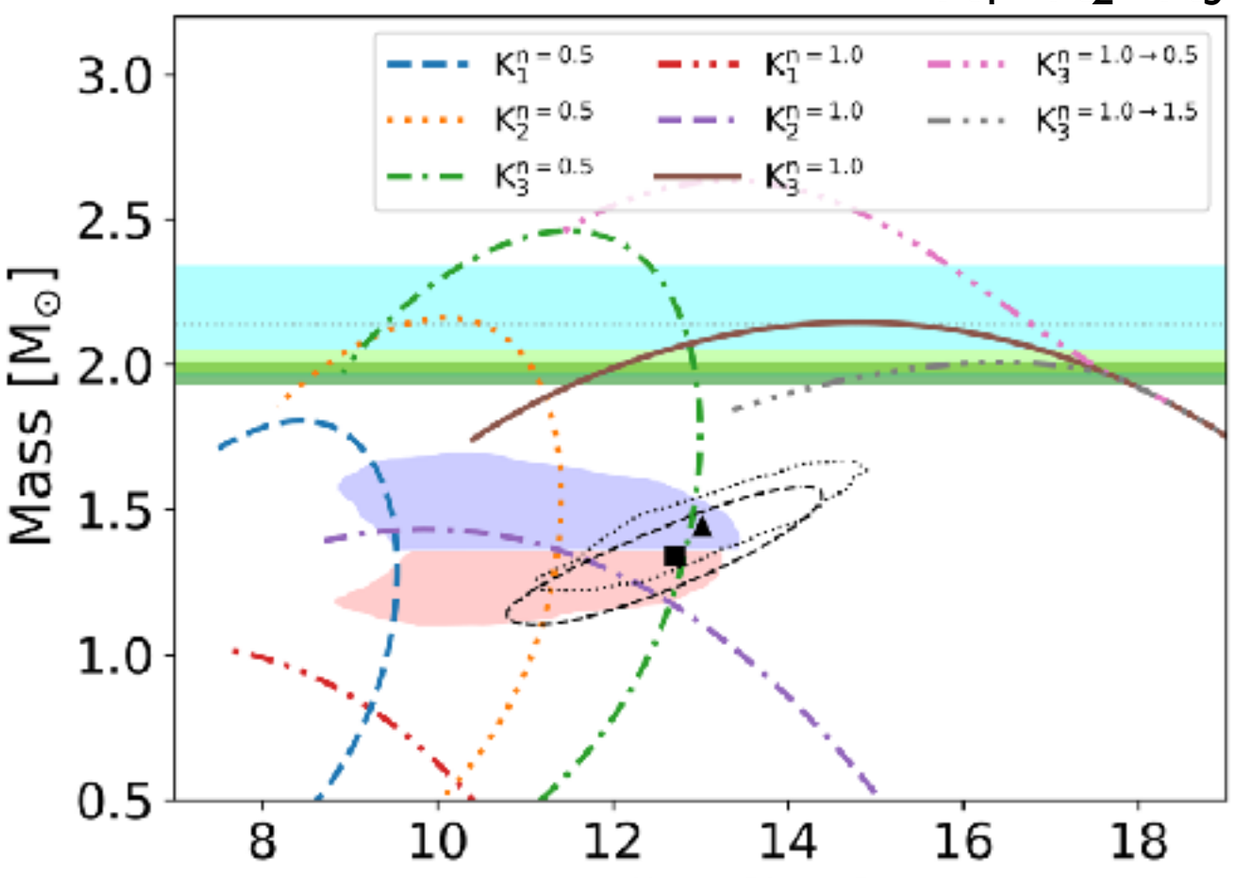


M. Kim et al., JKPS, 78, 932-941(2021)
https://link.springer.com/article/10.1007/s40042-021-00084-4

Polytropic EoSs vs. Realistic EoSs (1)

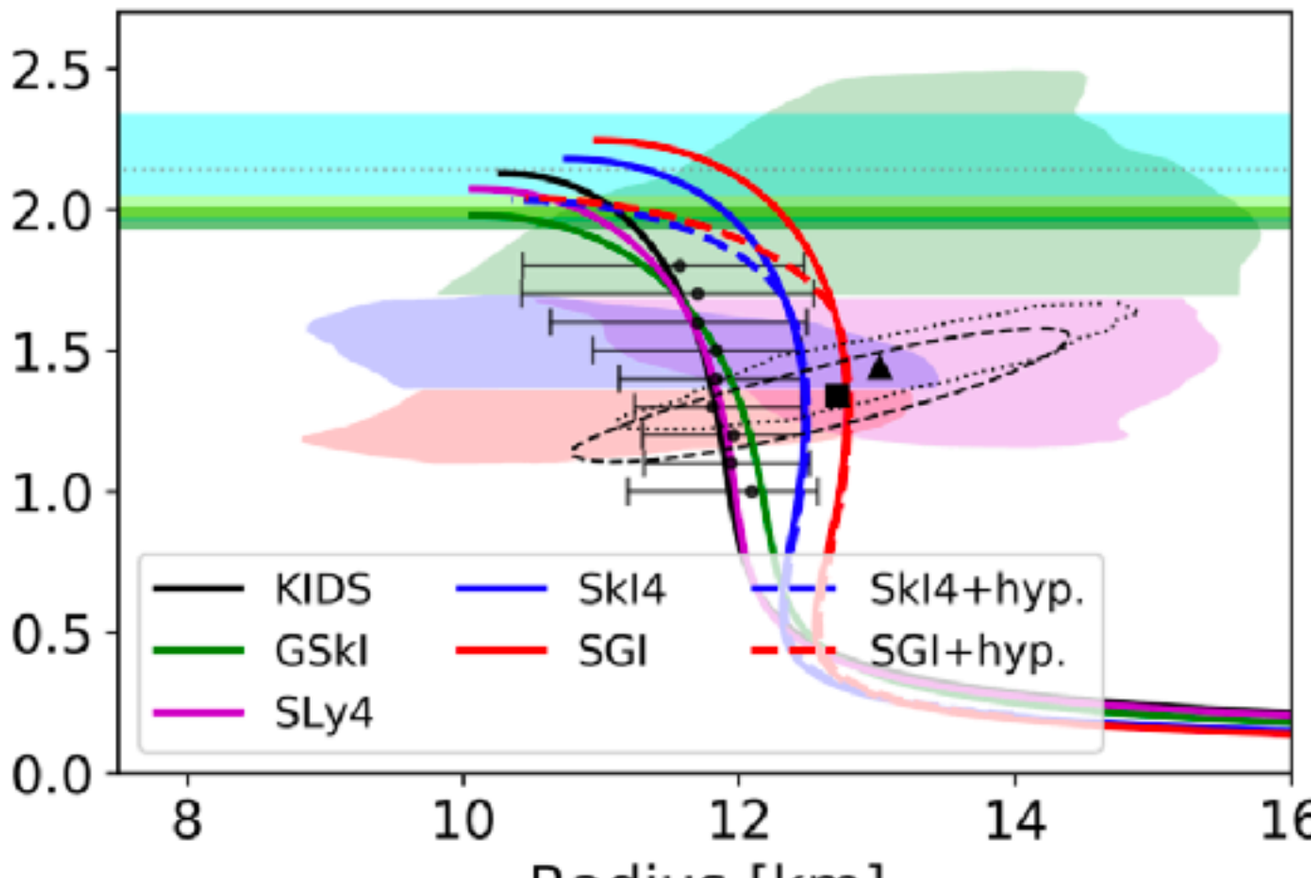
M. Kim et al. (JKPS 2021)

$K_1<K_2<K_3$



n < 1.0

M. Kim et al. (IJMPE 2020)

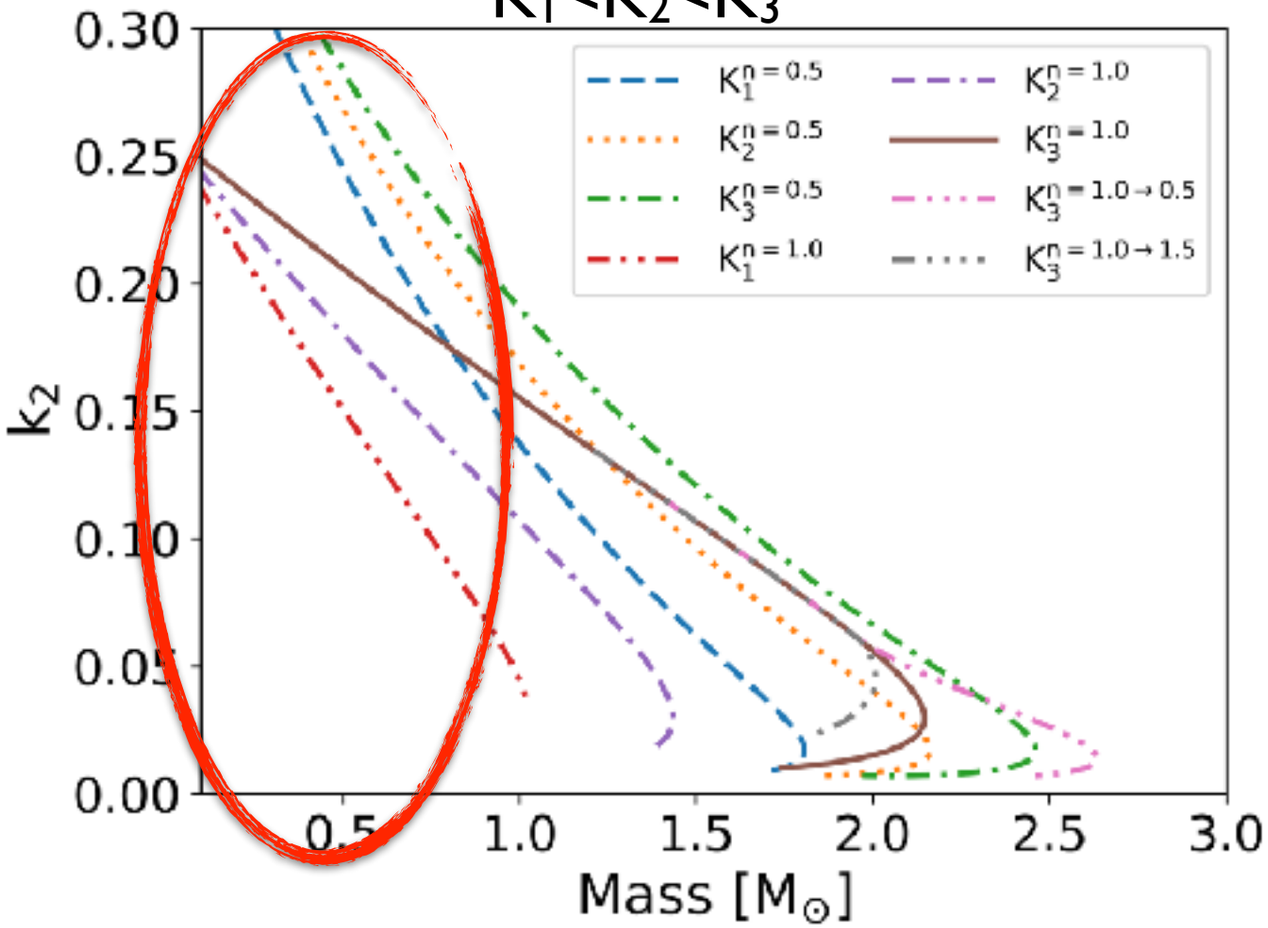


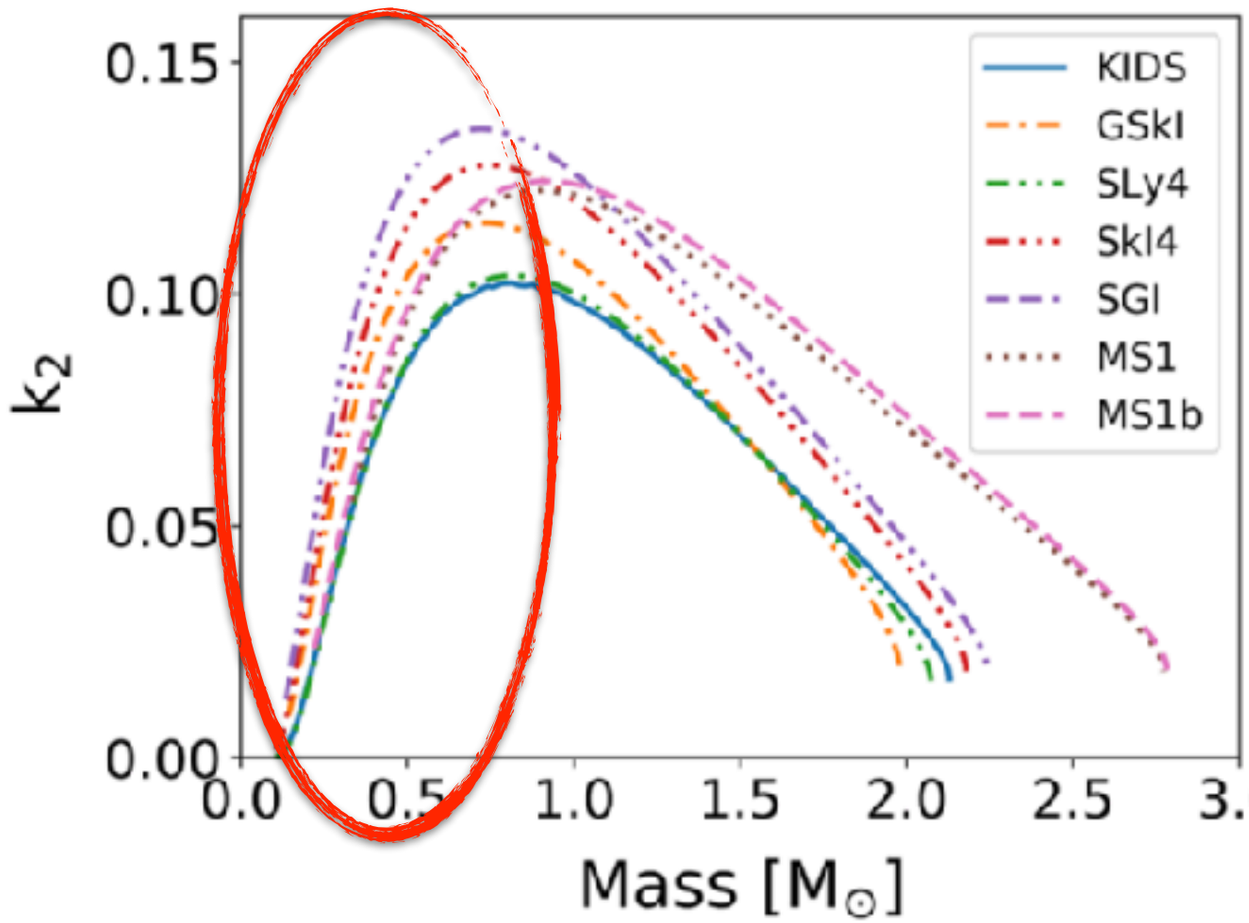
n >= 1.0

Polytropic EoSs vs. Realistic EoSs (2)

$$\Lambda = \lambda/M^5 \rightarrow G\left(\frac{c^2}{GM}\right)^5 \lambda = \frac{2}{3}\left(\frac{Rc^2}{GM}\right)^5 k_2$$

$K_1<K_2<K_3$





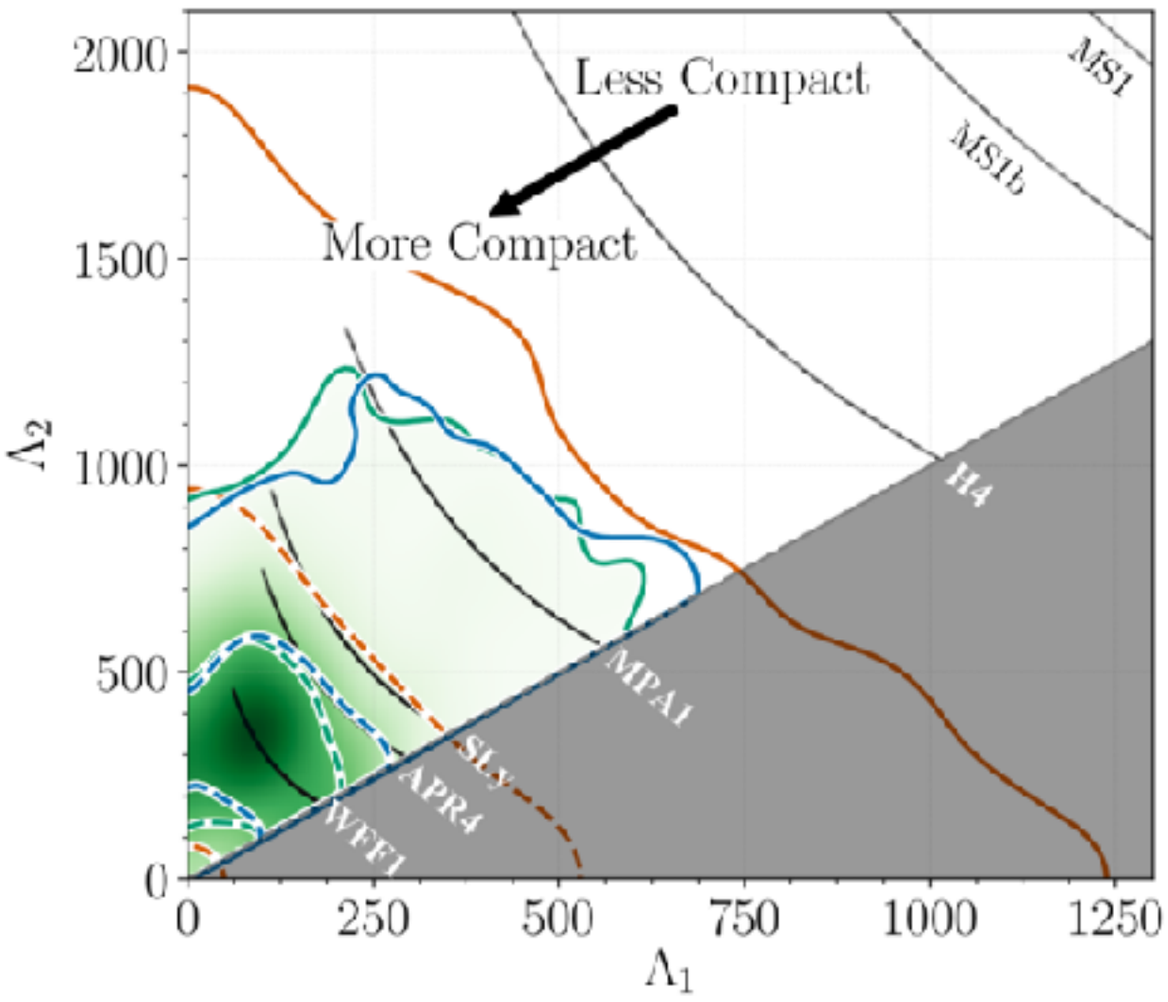
At the surface, p -> 0,

energy density -> non-zero for polytropic EoSs w/o nuclear matter crust.

zero for realistic EoSs w/ nuclear matter crust.

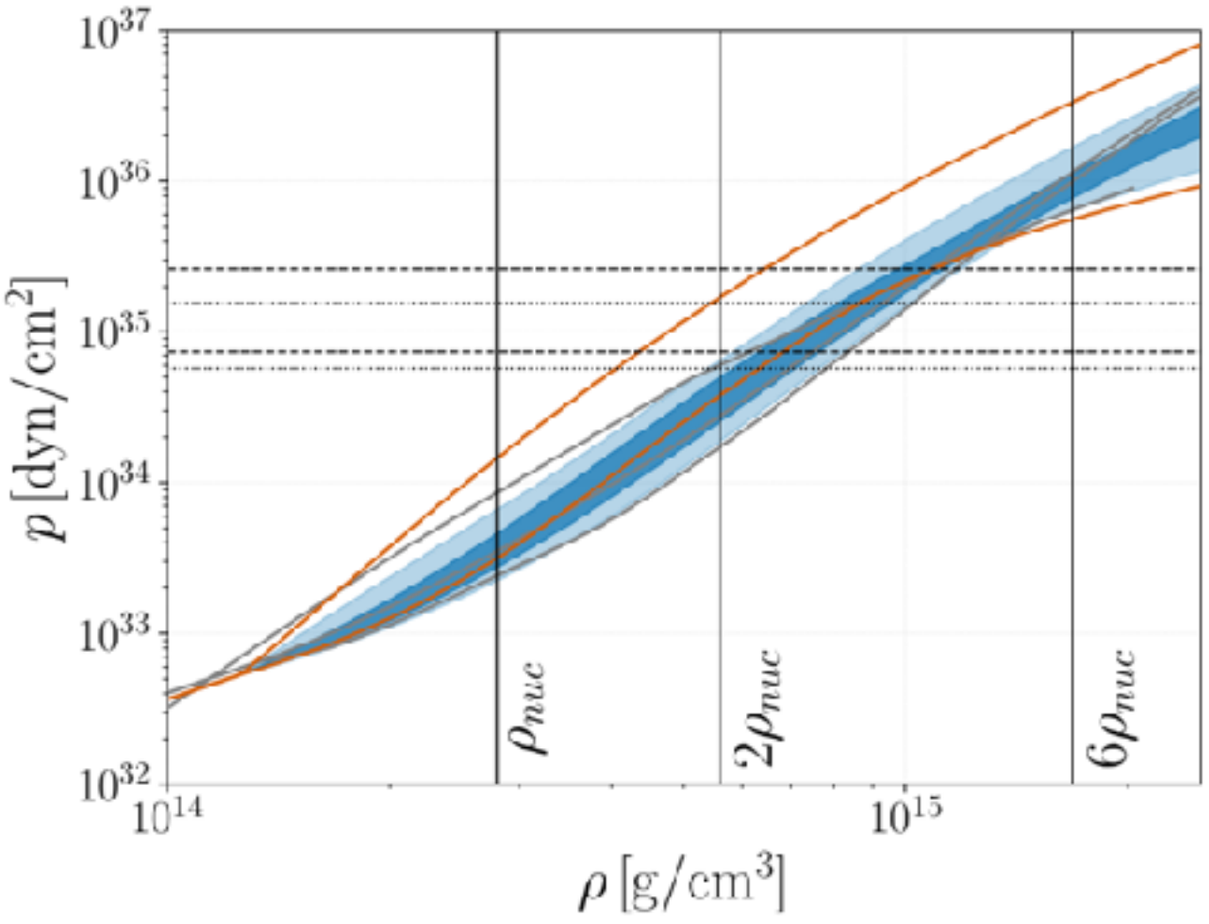
Measurements of GW170817

$\Lambda(1.4M_\odot)=190^{+390}_{-120}$



$P(2\ \rho_{nuc})= 3.5^{+2.7}_{-1.7} \times 10^{34}$ dyne/cm^2

$P(6\ \rho_{nuc})= 9.0^{+7.9}_{-2.6} \times 10^{35}$ dyne/cm^2



Abbott et al. (LSC and Virgo), arxiv:1805.11581 (PhysRevLett.121.161101)

$\rho_{nuc}= 2.8 \times 10^{14}$ g/cm^3

TOV solver in lalsimulation

```
import lalsimulation as lalsim
```

EoS generation

Polytrope EoS 내장함수 사용

```
eos = lalsim.SimNeutronStarEOSPolytrope(Gamma,
                                        reference_pressure_SI,
                                        reference_density_SI)
```

Polytrope EoS table 직접 계산해서 eosfile 생성 (첫번째 컬럼 pressure, 두번째 컬럼 density)

```
eos = lalsim.SimNeutronStarEOSFromFile(eosfile)
```

Solving TOV

```
eosfam = lalsim.CreateSimNeutronStarFamily(eos)

mass = 1.4 * lal.MSUN_SI
radius = lalsim.SimNeutronStarRadius(mass,eosfam)
k2 = lalsim.SimNeutronStarLoveNumberK2(mass,eosfam)
```

참고 자료: https://lscsoft.docs.ligo.org/lalsuite/lalsimulation/group___l_a_l_sim_neutron_star__h.html

주의사항

사용언어

코드는 python 언어로 작성해야한다.

단위

출력을 할 때 사용해야 하는 단위는 다음과 같다.

- even time: s
- mass: $M_\odot$ (solar mass)
- luminosity distance: Mpc
- neutron star radius: km
- neutron star density: cgs unit
- neutron star pressure: cgs unit

출력 정밀도

파일 출력시 정밀도 손실을 방지하기 위해 다음과 같이 소수점 16자리로 출력한다.

print (“{ 0: . 1 6 e } { 1: . 1 6 e } ”. format (value1, value2))

참고자료 (1)

1. Pycbc : http://pycbc.org/pycbc/latest/html/index.html

   - https://github.com/gw-odw/odw-2021/blob/master/Tutorials/Day_2/Tuto_2.2_Matched_Filtering_In_action.ipynb

   - https://github.com/gw-odw/odw-2021/blob/master/Tutorials/Day_2/Tuto_2.3_Signal_consistency_and_significance.ipynb

2. Bilby : https://lscsoft.docs.ligo.org/bilby/index.html

   - https://github.com/gw-odw/odw-2021/blob/master/Tutorials/Day_3/Tuto_3.2_Parameter_estimation_for_compact_object_mergers.ipynb

   - https://git.ligo.org/lscsoft/bilby/blob/master/examples/gw_examples/injection_examples/standard_15d_cbc_tutorial.py

   - https://git.ligo.org/lscsoft/bilby/-/blob/master/examples/gw_examples/injection_examples/binary_neutron_star_example.py

참고자료 (2)

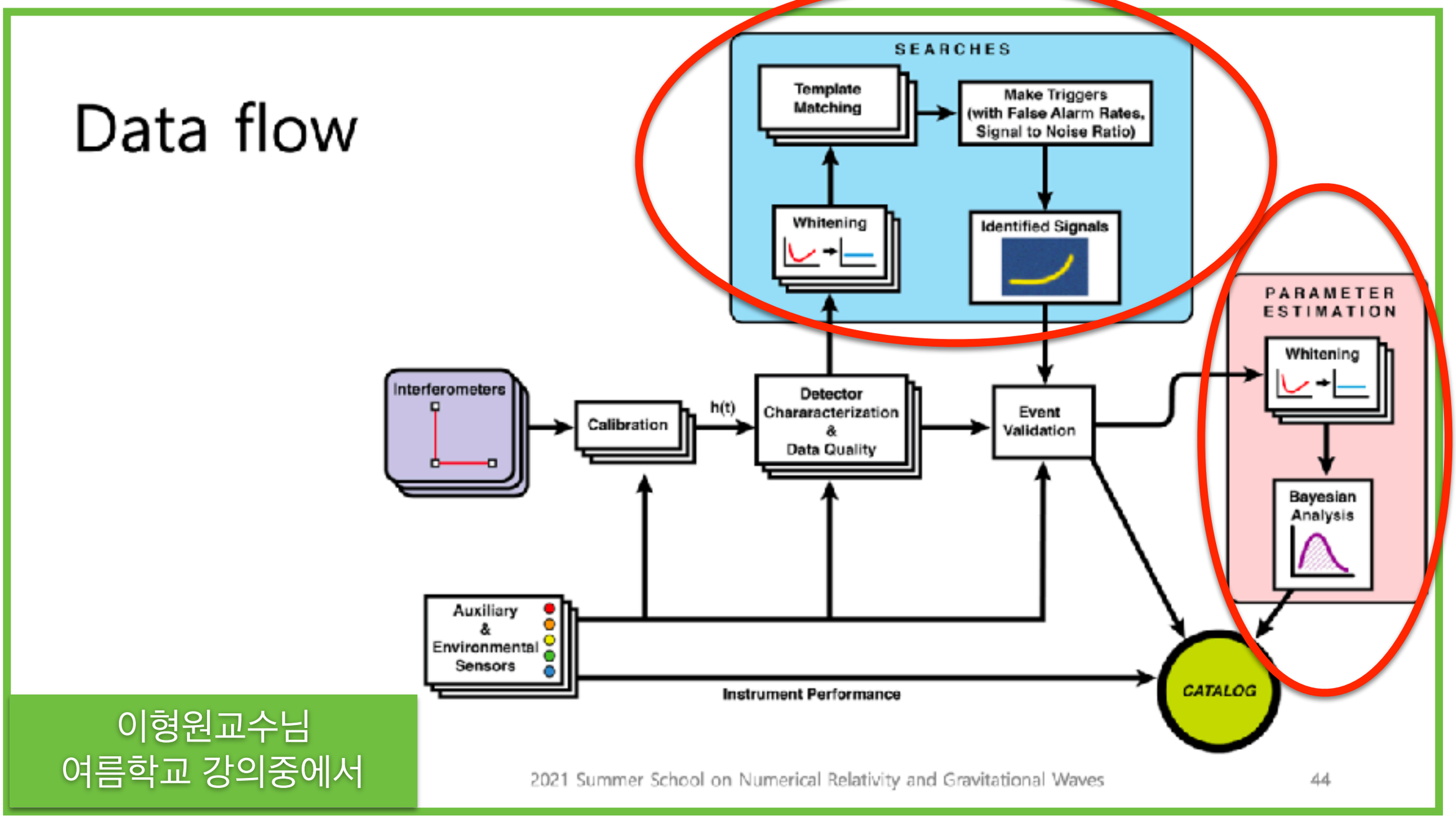
1. LAL에서의 단위계

   - G, gravitational constant = lal.G_SI
   - c, speed of light = lal.C_SI
   - Msun, solar mass = lal.MSUN_SI

2. Kernel Density Estimation

   - https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.gaussian_kde.html

알아야 될 것들.


Data flow
SEARCHES
Template Matching
Make Triggers (with False Alarm Rates, Signal to Noise Ratio)
Whitening
Identified Signals
PARAMETER ESTIMATION
Whitening
Bayesian Analysis
Interferometers
Calibration
h(t)
Detector Chararacterization & Data Quality
Event Validation
Auxiliary & Environmental Sensors
Instrument Performance
CATALOG
이형원교수님
여름학교 강의중에서
2021 Summer School on Numerical Relativity and Gravitational Waves
44

Gravitational-wave event searches

There are two types of searches, online and offline

- Online searches are low-latency searches which aim to get quick results in order to get rapid alerts of events
- Offline searches use archived data using more computationally expensive techniques to get deeper searches into the data

What searches are there?

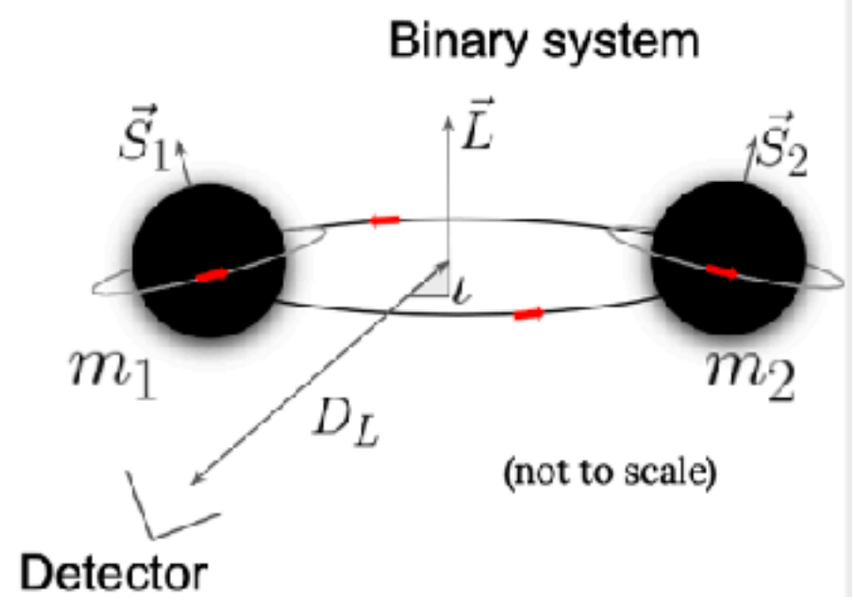
- Templated searches:
  - GstLAL - Online and Offline, lscsoft.docs.ligo.org/gstlal
  - **PyCBC** - Online and **Offline**, pycbc.org
  - MBTA - Online and Offline, T. Adams et al (2016)
  - SPIIR - Online only, Q. Chu (2017)
  - IAS - Offline only, Venumadhav et al. (2020)
- Non-templated search
  - cWB - Online and Offline gwburst.gitlab.io

Modelling colliding black holes

What will the signals from these systems look like in the data?

The signal from a binary system made up of black holes will be described by fifteen parameters

- Intrinsic parameters:
  - Component Masses: m_1 m_2
  - Component spins in each direction: s_{1x} s_{1y} s_{1z} s_{2x} s_{2y} s_{2z}
- Extrinsic Parameters:
  - Location: Right Ascension and Declination
  - Inclination angle between line of sight and orbital plane, ι
  - Polarisation angle,
  - Phase at coalescence
  - Luminosity distance, D_L
  - Time of coalescence



Generating Waveforms

```
from pycbc.waveform import get_td_waveform
import pylab
```

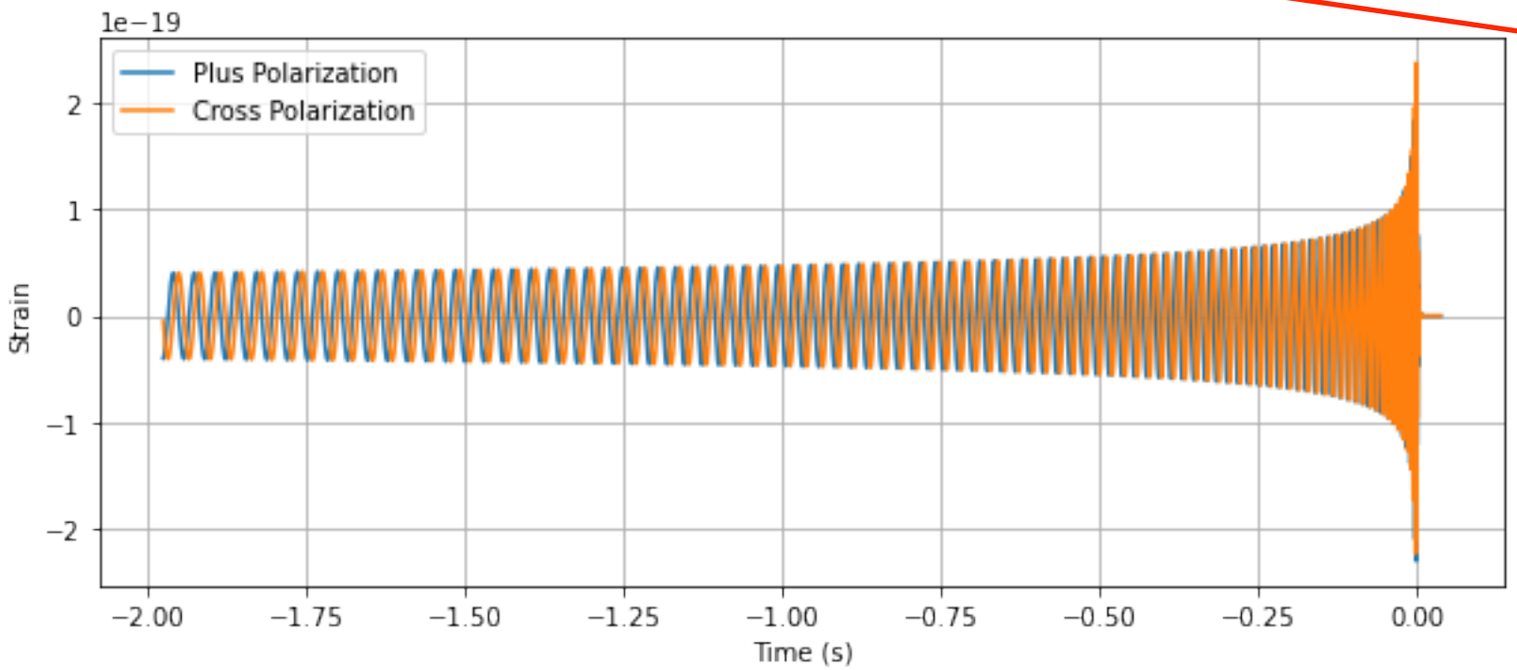
Spinning Effective One Body + Numerical Relativity

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

$M_\odot$ (solar mass)

Seconds

Hz



Generating Waveforms

```
from pycbc.waveform import get_td_waveform
import pylab
```

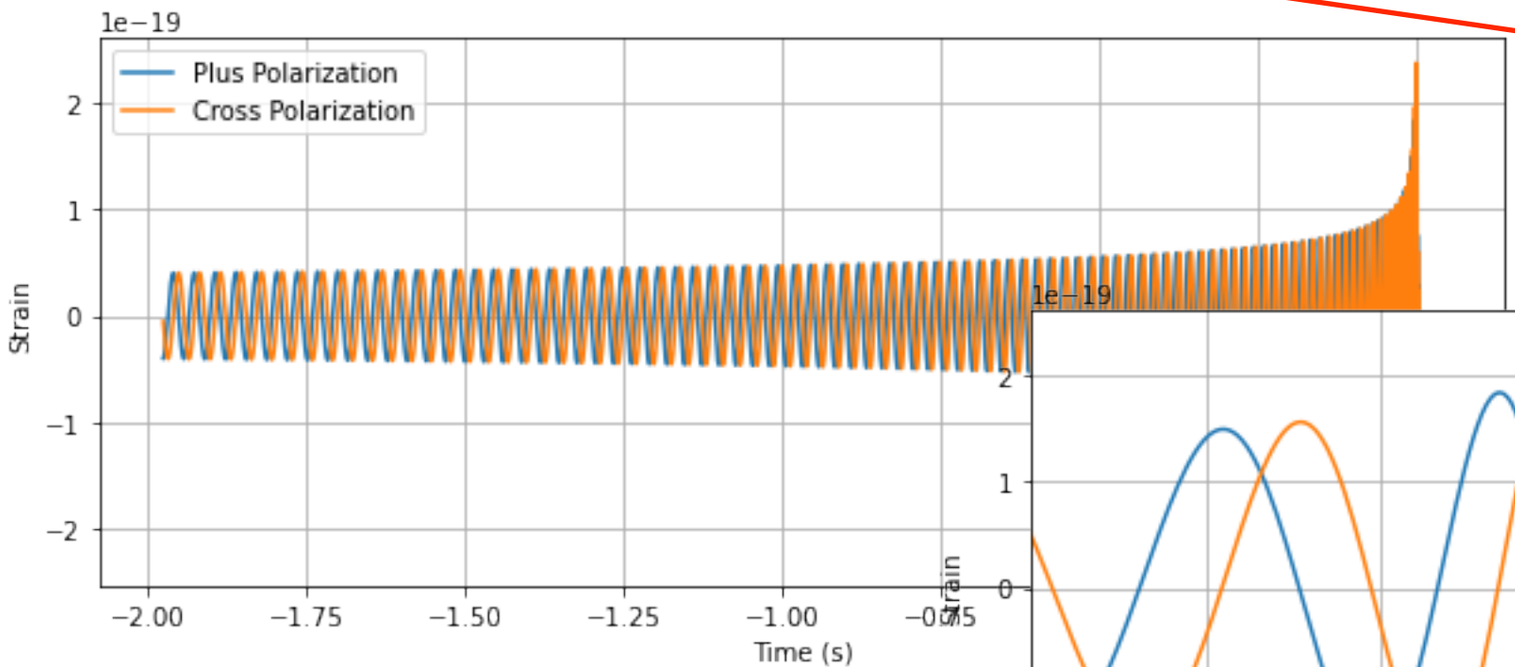
Spinning Effective One Body + Numerical Relativity

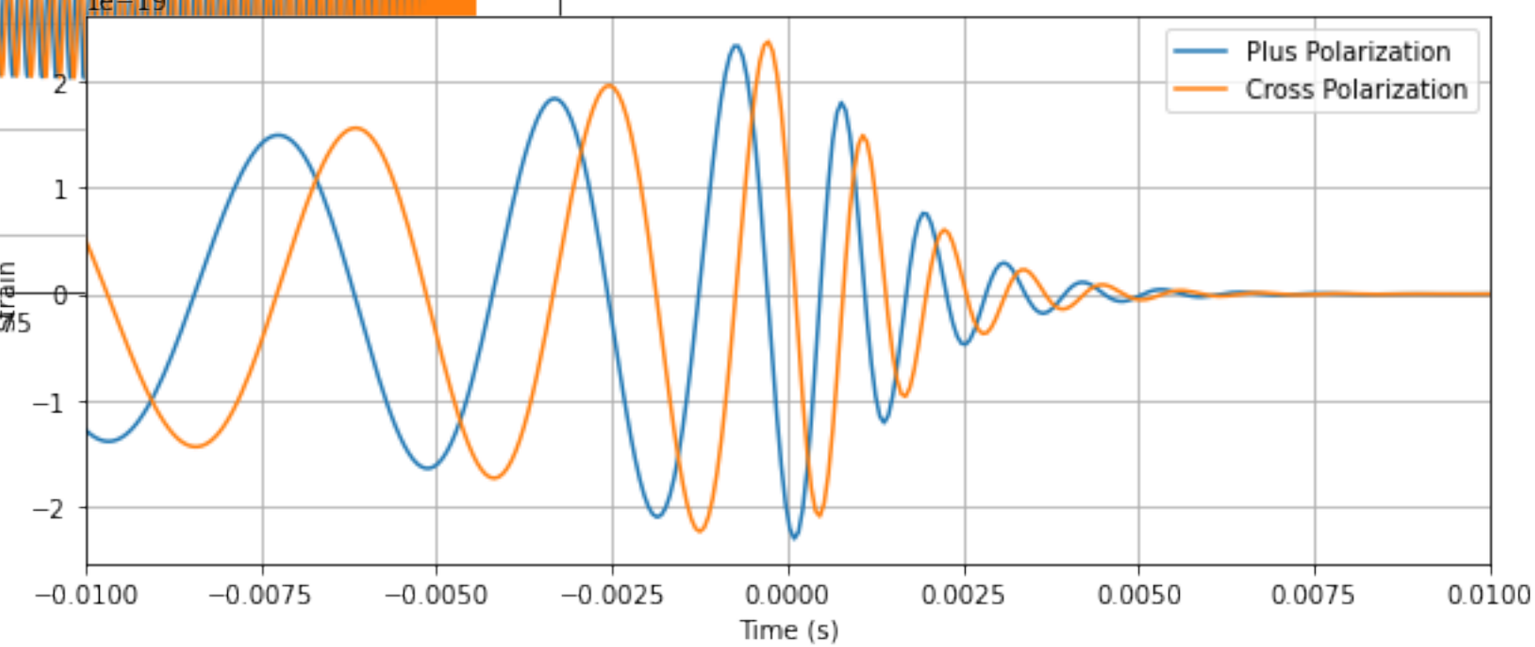
```
hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                         mass1=10,
                         mass2=10,
                         delta_t=1.0/16384,
                         f_lower=30)
```

$M_\odot$ (solar mass)

Seconds

Hz





Waveform Approximants

```
from pycbc.waveform import td_approximants, fd_approximants
print('Time domain waveform approximants: ',td_approximants())
print('Frequency domain waveform approximants: ', fd_approximants())
```

```
Time domain waveform approximants:  ['TaylorT1', 'TaylorT2', 'TaylorT3', 'SpinTaylorT1', 'SpinTaylorT4',
'SpinTaylorT5', 'PhenSpinTaylor', 'PhenSpinTaylorRD', 'EOBNRv2', 'EOBNRv2HM', 'TEOBResum_ROM', 'SEOBNRv1',
'SEOBNRv2', 'SEOBNRv2_opt', 'SEOBNRv3', 'SEOBNRv3_pert', 'SEOBNRv3_opt', 'SEOBNRv3_opt_rk4', 'SEOBNRv4',
'SEOBNRv4_opt', 'SEOBNRv4P', 'SEOBNRv4PHM', 'SEOBNRv2T', 'SEOBNRv4T', 'SEOBNRv4_ROM_NRTidalv2',
'SEOBNRv4_ROM_NRTidalv2_NSBH', 'HGimri', 'IMRPhenomA', 'IMRPhenomB', 'IMRPhenomC', 'IMRPhenomD',
'IMRPhenomD_NRTidalv2', 'IMRPhenomNSBH', 'IMRPhenomHM', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal',
'IMRPhenomPv2_NRTidalv2', 'TaylorEt', 'TaylorT4', 'EccentricTD', 'SpinDominatedWf', 'NR_hdf5', 'NRSur7dq2',
'NRSur7dq4', 'SEOBNRv4HM', 'NRHybSur3dq8', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM',
'IMRPhenomXP', 'IMRPhenomXPHM', 'TEOBResumS', 'IMRPhenomT', 'IMRPhenomTHM', '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', 'SpinTaylorF2', 'TaylorF2NL',
'PreTaylorF2', ‘SpinTaylorF2_SWAPPER']

Frequency domain waveform approximants:  ['EccentricFD', 'TaylorF2', 'TaylorF2Ecc', 'TaylorF2NLTides',
'TaylorF2RedSpin', 'TaylorF2RedSpinTidal', 'SpinTaylorF2', 'EOBNRv2_ROM', 'EOBNRv2HM_ROM',
'SEOBNRv1_ROM_EffectiveSpin', 'SEOBNRv1_ROM_DoubleSpin', 'SEOBNRv2_ROM_EffectiveSpin', '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_surrogate', 'IMRPhenomA',
'IMRPhenomB', 'IMRPhenomC', 'IMRPhenomD', 'IMRPhenomD_NRTidal', 'IMRPhenomD_NRTidalv2', 'IMRPhenomNSBH',
'IMRPhenomHM', 'IMRPhenomP', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal', 'IMRPhenomPv2_NRTidalv2', 'SpinTaylorT4Fourier',
'SpinTaylorT5Fourier', 'NRSur4d2s', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM', 'IMRPhenomXP',
'IMRPhenomXPHM', 'SpinTaylorF2_SWAPPER', 'TaylorF2NL', 'PreTaylorF2', 'multiband', 'TaylorF2_INTERP', 'SpinTaylorT5',
'SEOBNRv1_ROM_EffectiveSpin_INTERP', 'SEOBNRv1_ROM_DoubleSpin_INTERP', 'SEOBNRv2_ROM_EffectiveSpin_INTERP',
'SEOBNRv2_ROM_DoubleSpin_INTERP', 'EOBNRv2_ROM_INTERP', 'EOBNRv2HM_ROM_INTERP', 'SEOBNRv2_ROM_DoubleSpin_HI_INTERP',
'SEOBNRv4_ROM_INTERP', 'SEOBNRv4HM_ROM_INTERP', 'SEOBNRv4', 'SEOBNRv4P', 'IMRPhenomC_INTERP', 'IMRPhenomD_INTERP',
'IMRPhenomPv2_INTERP', 'IMRPhenomD_NRTidal_INTERP', 'IMRPhenomPv2_NRTidal_INTERP', 'IMRPhenomHM_INTERP',
'IMRPhenomPv3HM_INTERP', 'IMRPhenomXHM_INTERP', 'IMRPhenomXPHM_INTERP', 'SpinTaylorF2_INTERP', 'TaylorF2NL_INTERP',
'PreTaylorF2_INTERP', 'SpinTaylorF2_SWAPPER_INTERP']
```

Waveform Approximants

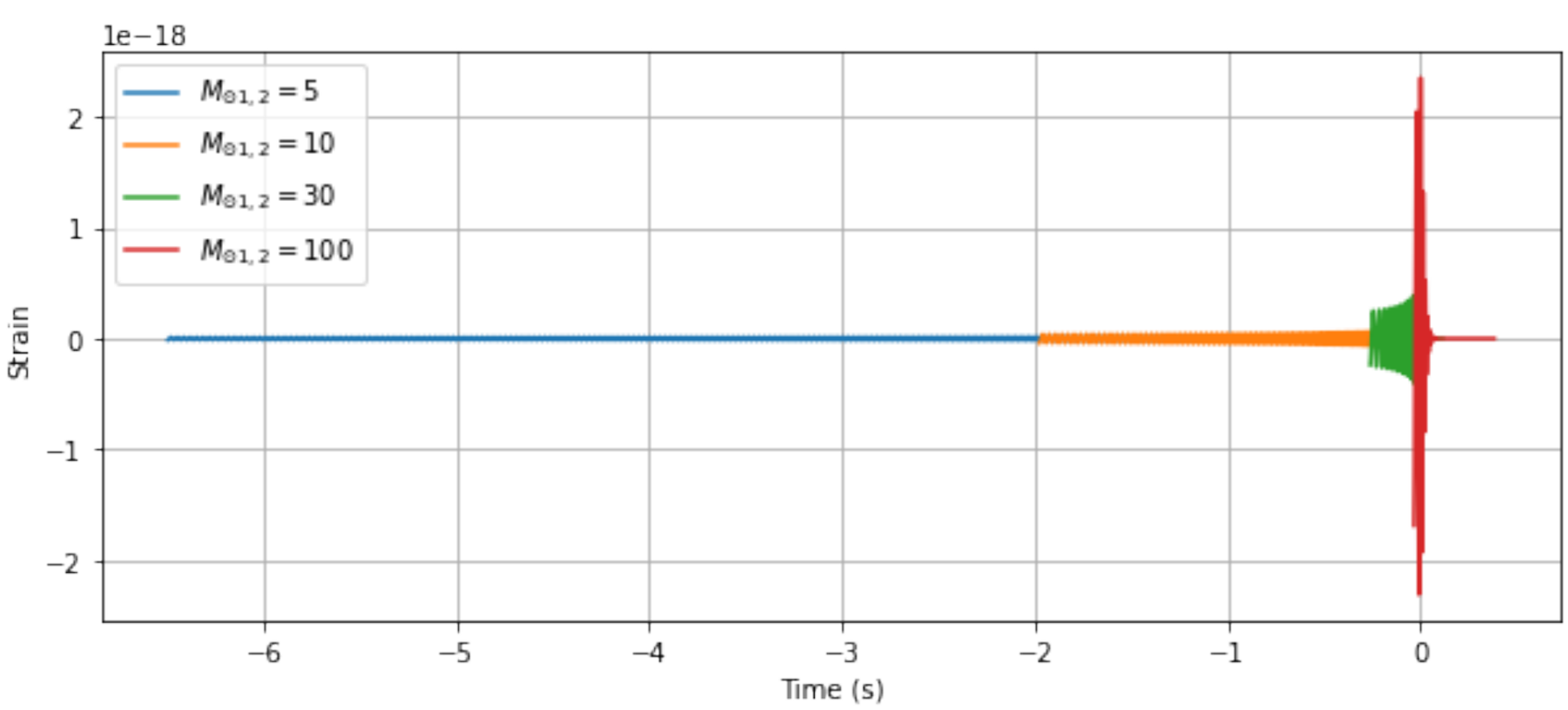
문제 풀이 Hint!!
추천 waveform model:
BBH : SEOBNRv2
BNS : IMRPhenomPv2_NRTidal

```
from pycbc.waveform import td_approximants, fd_approximants
print('Time domain waveform approximants: ',td_approximants())
print('Frequency domain waveform approximants: ', fd_approximants())
```

```
Time domain waveform approximants:  ['TaylorT1', 'TaylorT2', 'TaylorT3', 'SpinTaylorT1', 'SpinTaylorT4',
'SpinTaylorT5', 'PhenSpinTaylor', 'PhenSpinTaylorRD', 'EOBNRv2', 'EOBNRv2HM', 'TEOBResum_ROM', 'SEOBNRv1',
'SEOBNRv2', 'SEOBNRv2_opt', 'SEOBNRv3', 'SEOBNRv3_pert', 'SEOBNRv3_opt', 'SEOBNRv3_opt_rk4', 'SEOBNRv4',
'SEOBNRv4_opt', 'SEOBNRv4P', 'SEOBNRv4PHM', 'SEOBNRv2T', 'SEOBNRv4T', 'SEOBNRv4_ROM_NRTidalv2',
'SEOBNRv4_ROM_NRTidalv2_NSBH', 'HGimri', 'IMRPhenomA', 'IMRPhenomB', 'IMRPhenomC', 'IMRPhenomD',
'IMRPhenomD_NRTidalv2', 'IMRPhenomNSBH', 'IMRPhenomHM', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal',
'IMRPhenomPv2_NRTidalv2', 'TaylorEt', 'TaylorT4', 'EccentricTD', 'SpinDominatedWf', 'NR_hdf5', 'NRSur7dq2',
'NRSur7dq4', 'SEOBNRv4HM', 'NRHybSur3dq8', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM',
'IMRPhenomXP', 'IMRPhenomXPHM', 'TEOBResumS', 'IMRPhenomT', 'IMRPhenomTHM', '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', 'SpinTaylorF2', 'TaylorF2NL',
'PreTaylorF2', ‘SpinTaylorF2_SWAPPER']

Frequency domain waveform approximants:  ['EccentricFD', 'TaylorF2', 'TaylorF2Ecc', 'TaylorF2NLTides',
'TaylorF2RedSpin', 'TaylorF2RedSpinTidal', 'SpinTaylorF2', 'EOBNRv2_ROM', 'EOBNRv2HM_ROM',
'SEOBNRv1_ROM_EffectiveSpin', 'SEOBNRv1_ROM_DoubleSpin', 'SEOBNRv2_ROM_EffectiveSpin', '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_surrogate', 'IMRPhenomA',
'IMRPhenomB', 'IMRPhenomC', 'IMRPhenomD', 'IMRPhenomD_NRTidal', 'IMRPhenomD_NRTidalv2', 'IMRPhenomNSBH',
'IMRPhenomHM', 'IMRPhenomP', 'IMRPhenomPv2', 'IMRPhenomPv2_NRTidal', 'IMRPhenomPv2_NRTidalv2', 'SpinTaylorT4Fourier',
'SpinTaylorT5Fourier', 'NRSur4d2s', 'IMRPhenomXAS', 'IMRPhenomXHM', 'IMRPhenomPv3', 'IMRPhenomPv3HM', 'IMRPhenomXP',
'IMRPhenomXPHM', 'SpinTaylorF2_SWAPPER', 'TaylorF2NL', 'PreTaylorF2', 'multiband', 'TaylorF2_INTERP', 'SpinTaylorT5',
'SEOBNRv1_ROM_EffectiveSpin_INTERP', 'SEOBNRv1_ROM_DoubleSpin_INTERP', 'SEOBNRv2_ROM_EffectiveSpin_INTERP',
'SEOBNRv2_ROM_DoubleSpin_INTERP', 'EOBNRv2_ROM_INTERP', 'EOBNRv2HM_ROM_INTERP', 'SEOBNRv2_ROM_DoubleSpin_HI_INTERP',
'SEOBNRv4_ROM_INTERP', 'SEOBNRv4HM_ROM_INTERP', 'SEOBNRv4', 'SEOBNRv4P', 'IMRPhenomC_INTERP', 'IMRPhenomD_INTERP',
'IMRPhenomPv2_INTERP', 'IMRPhenomD_NRTidal_INTERP', 'IMRPhenomPv2_NRTidal_INTERP', 'IMRPhenomHM_INTERP',
'IMRPhenomPv3HM_INTERP', 'IMRPhenomXHM_INTERP', 'IMRPhenomXPHM_INTERP', 'SpinTaylorF2_INTERP', 'TaylorF2NL_INTERP',
'PreTaylorF2_INTERP', 'SpinTaylorF2_SWAPPER_INTERP']
```

WF w/ different masses

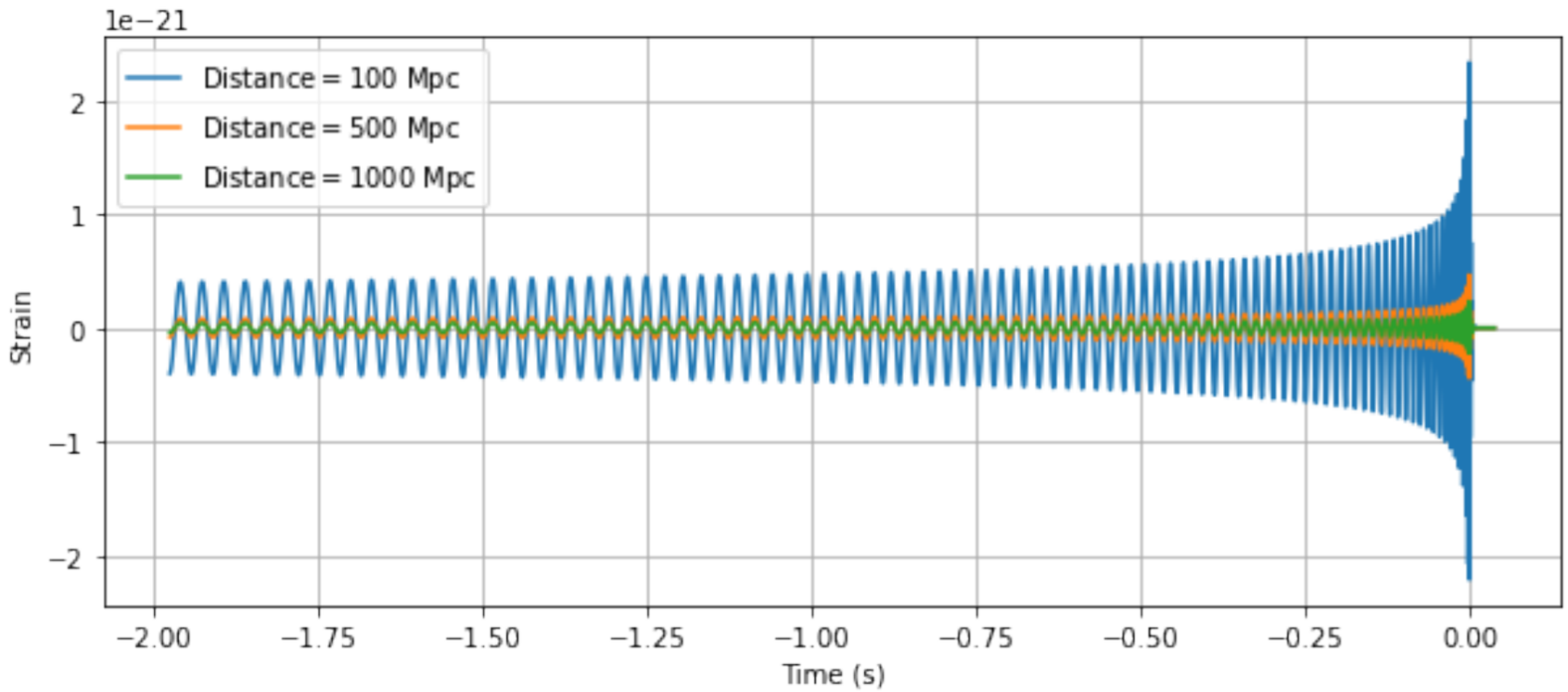
```
pylab.figure(figsize=pylab.figaspect(0.4))
for m in [5, 10, 30, 100]:
    hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                             mass1=m,
                             mass2=m,
                             delta_t=1.0/4096,
                             f_lower=30)

    pylab.plot(hp.sample_times, hp, label='$M_{\odot 1,2}=%s$' % m)
```



WF w/ different distances

```
pylab.figure(figsize=pylab.figaspect(0.4))
for d in [100, 500, 1000]:
    hp, hc = get_td_waveform(approximant="SEOBNRv4_opt",
                             mass1=10,
                             mass2=10,
                             delta_t=1.0/4096,
                             f_lower=30,
                             distance=d)

    pylab.plot(hp.sample_times, hp, label='Distance$=%s$ Mpc' % d)
```

Mpc



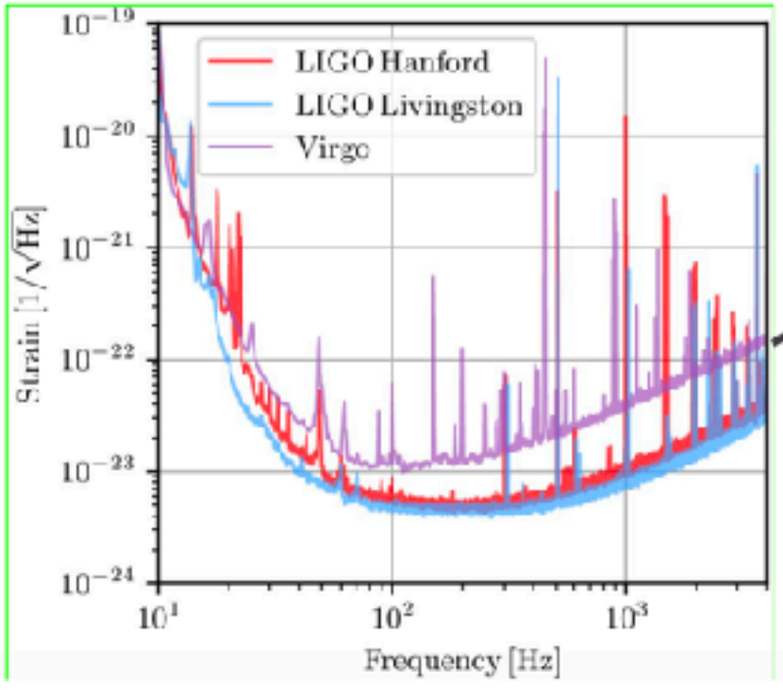
Matched Filtering

Optimal for signals:

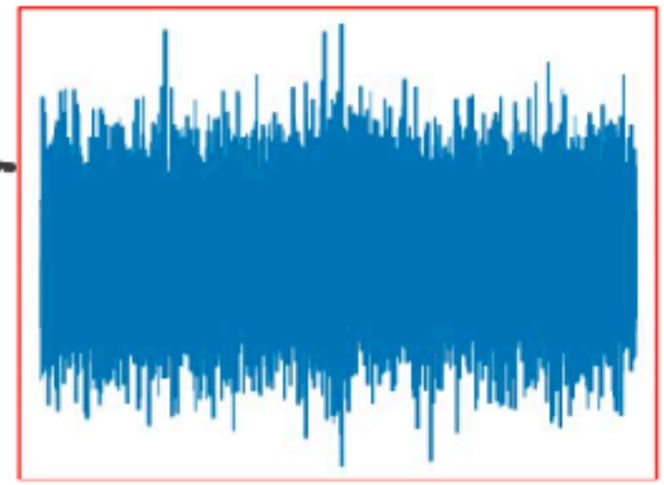
- in stationary Gaussian noise
- with known PSD

(Wainstein and Zubakov, 1962)

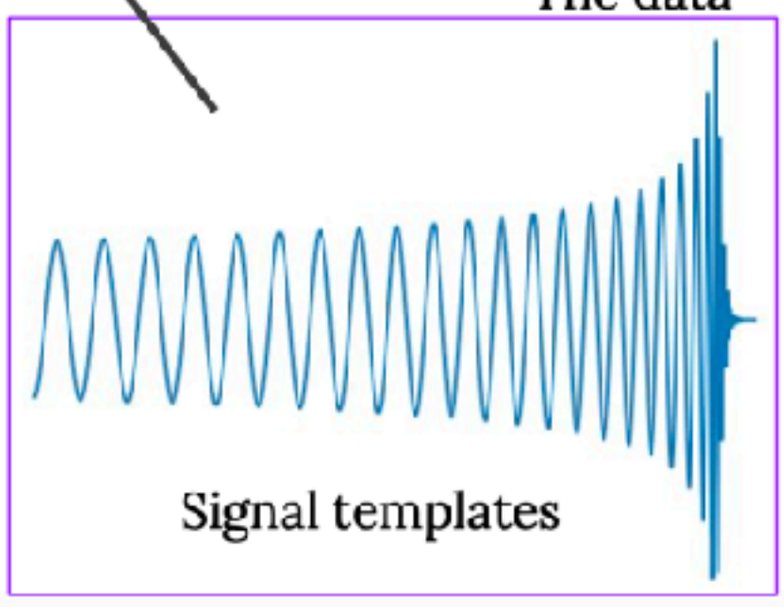
$$(s|h) = 4\Re \int_0^\infty \frac{\tilde{s}(f)\tilde{h}^\star(f)}{S_h(f)} \mathrm{d}f$$



PSD



The data



Signal templates

Cross-corre

```
hp1, _ = get_td_waveform(a
                         m
                         m
                         d
                         f

hp1 = hp1 / max(numpy.corr

sample_rate = 1024 # sampl
data_length = 1024 # secon

# Generate a long stretch
data = numpy.random.normal
times = numpy.arange(len(d
```

Matched Filter

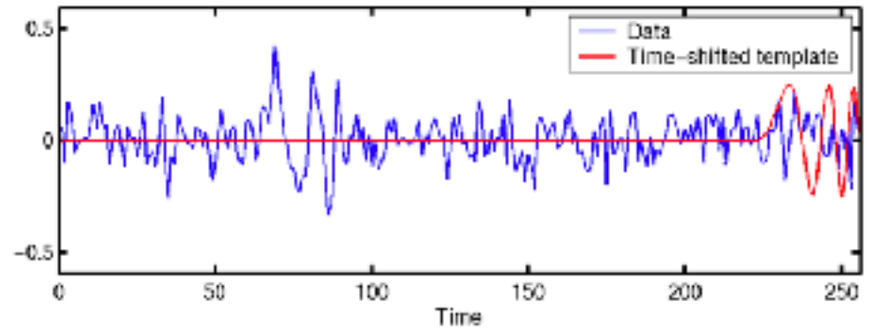
- Correlate data with expected signal (Here, plotting absolute value) G1400343

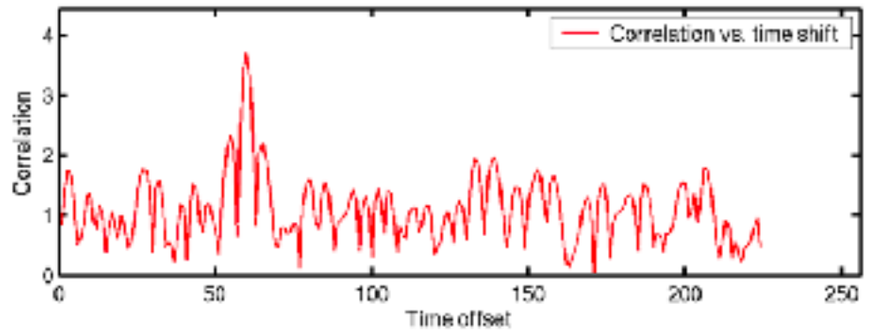
$$C(t) = \int_{-\infty}^{\infty} dt'\, s(t')\, h(t'-t)$$

Time offset

Data

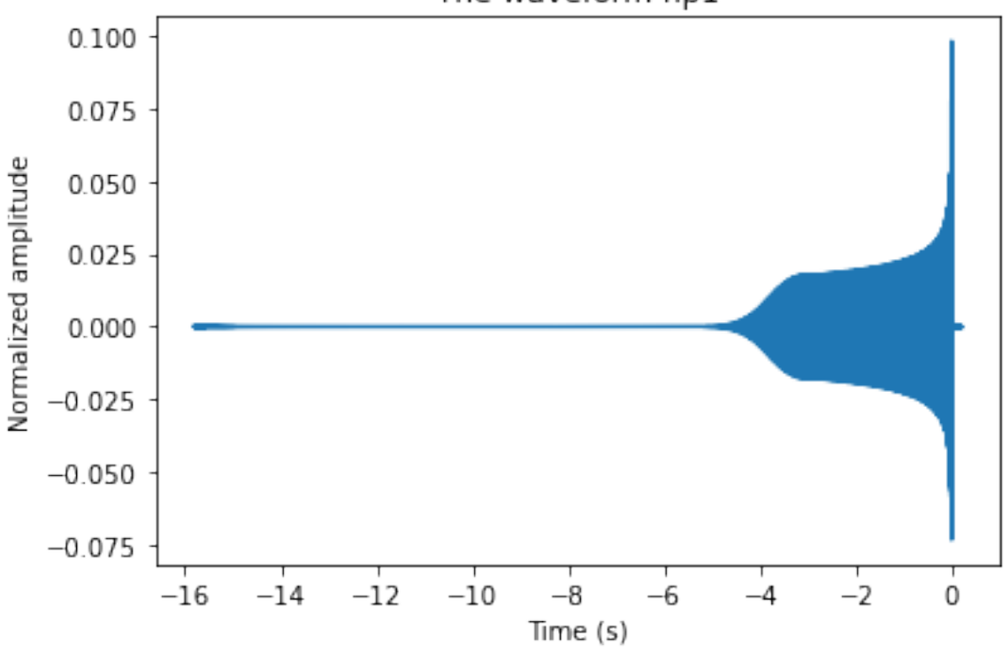
Template with time offset

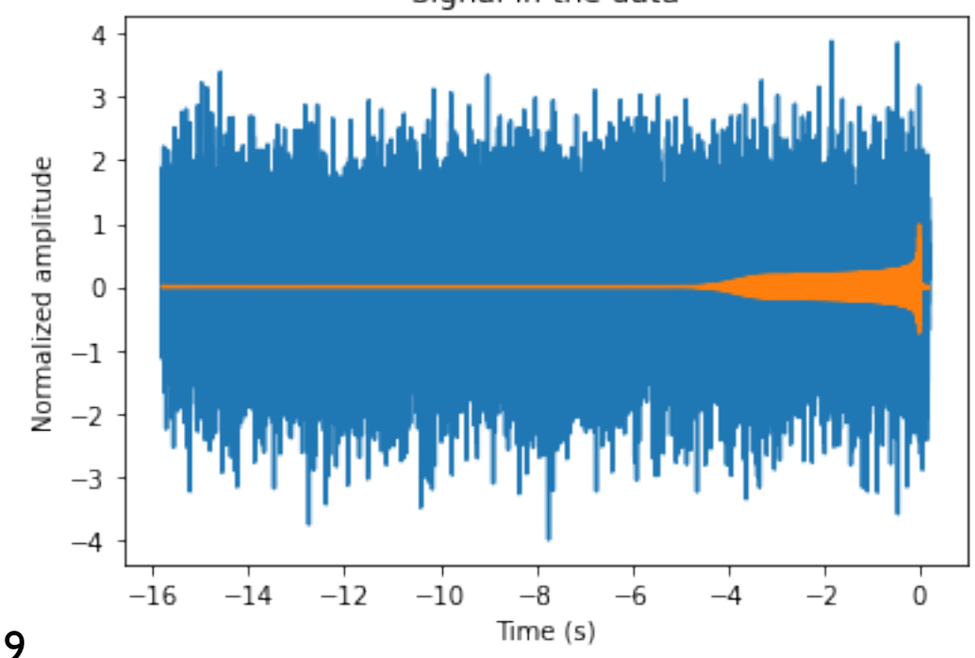




이형원교수님
여름학교 강의중에서

2021 Summer School on Numerical Relativity and Gravitational Waves 48





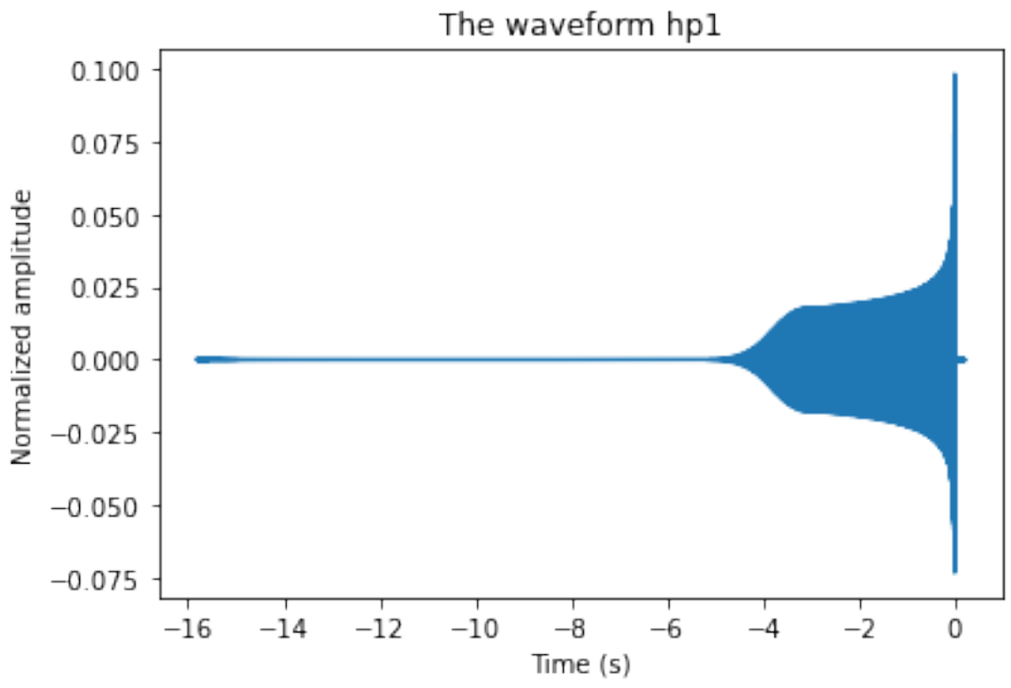
Cross-correlation

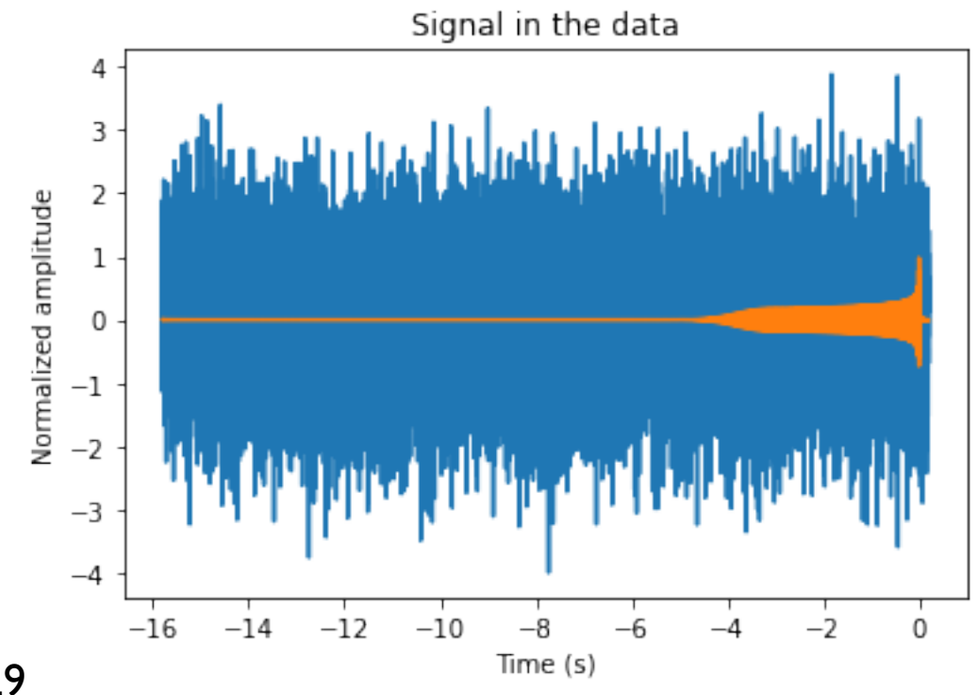
```
hp1, _ = get_td_waveform(approximant=apx,
                         mass1=10,
                         mass2=10,
                         delta_t=1.0/sample_rate,
                         f_lower=25)
```

```
apx = 'IMRPhenomD'
```

```
hp1 = hp1 / max(numpy.correlate(hp1, hp1, mode='full'))**0.5
```

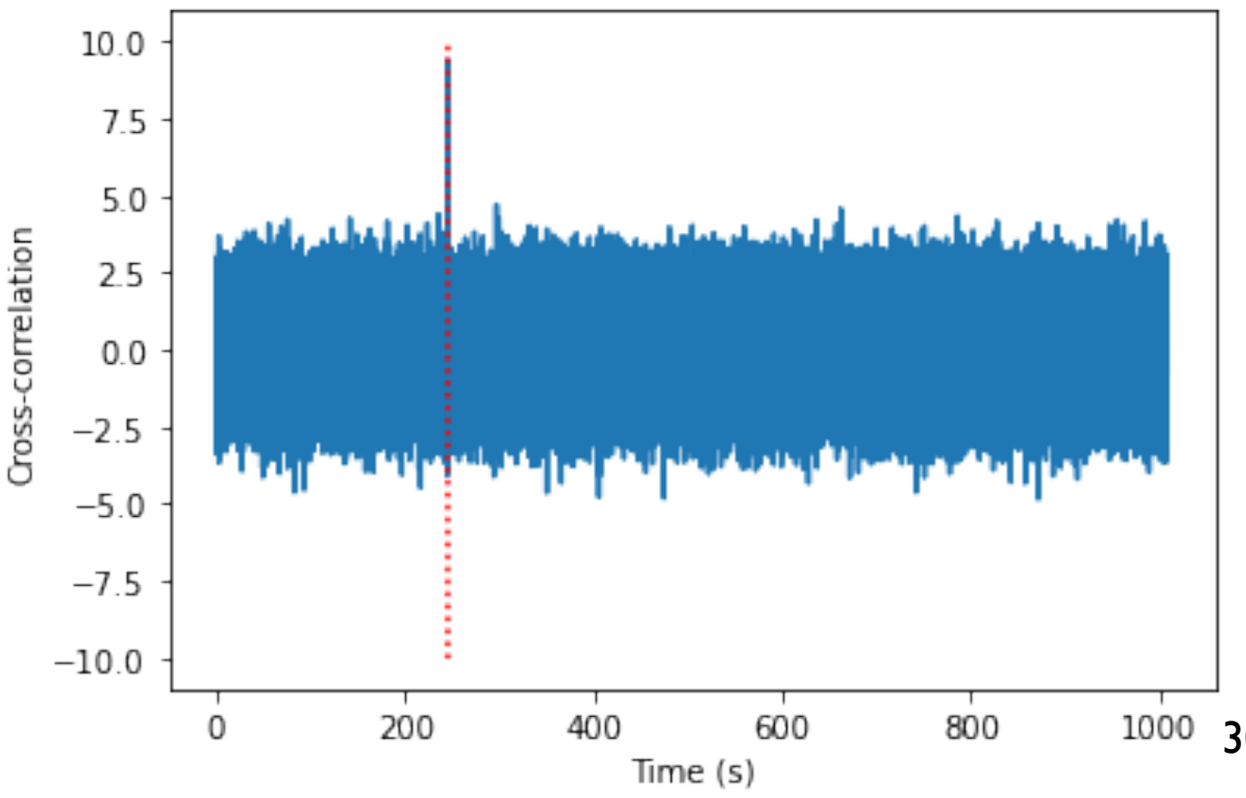
```
sample_rate = 1024 # samples per second
data_length = 1024 # seconds

# Generate a long stretch of white noise: the data series and the time series.
data = numpy.random.normal(size=[sample_rate * data_length])
times = numpy.arange(len(data)) / float(sample_rate)
```





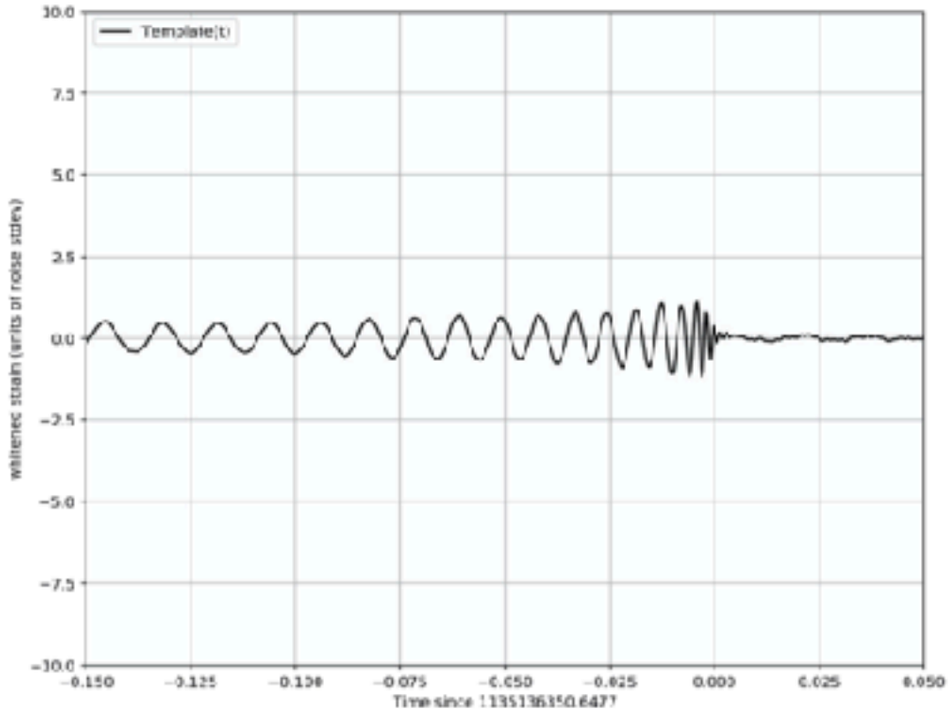
Cross-correlation

```
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')
```

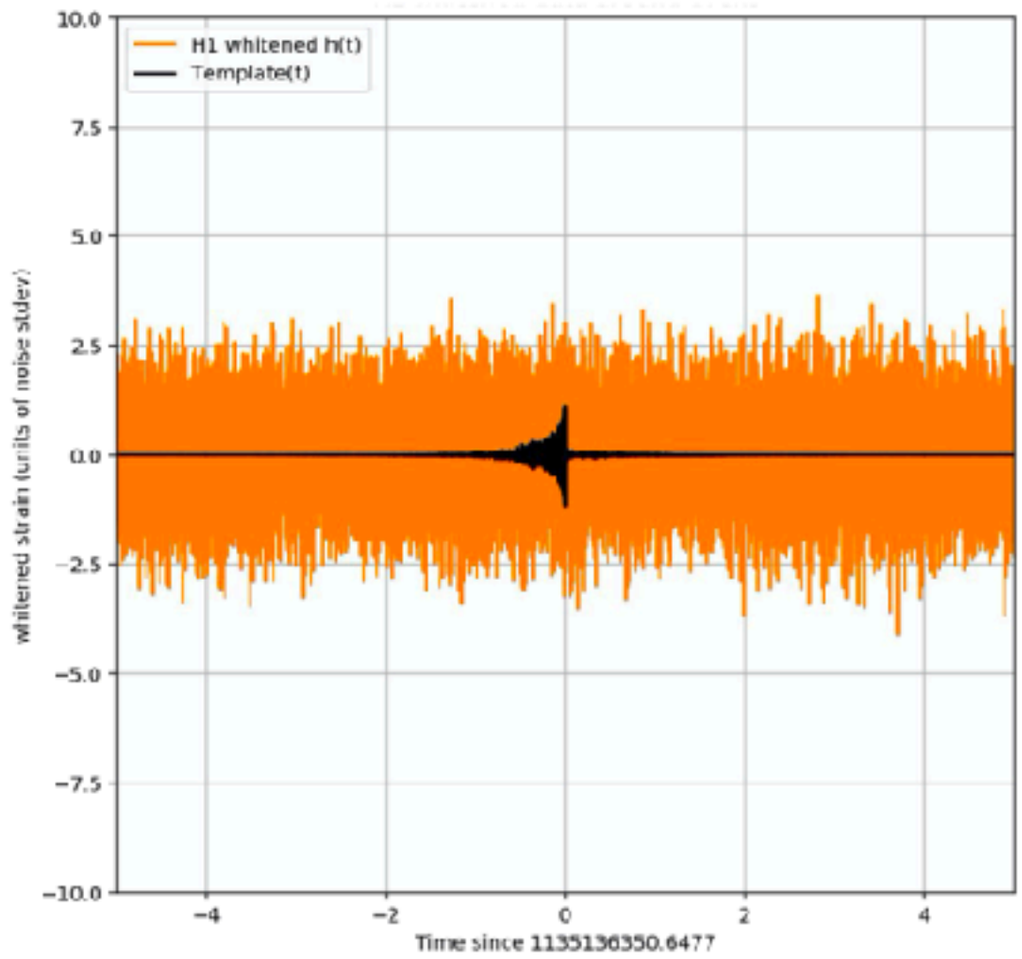


Detection problem

We know what the signal looks like



But it is buried in detector noise



Adapted from GWOSC tutorial

Noise backgrounds

Complicated noise curves

Many lines in the data, not such an issue for transient searches, but can be an issue for continuous wave searches

To an okay approximation, the detector data is colored Gaussian noise - standard Gaussian noise just with certain frequencies louder than others

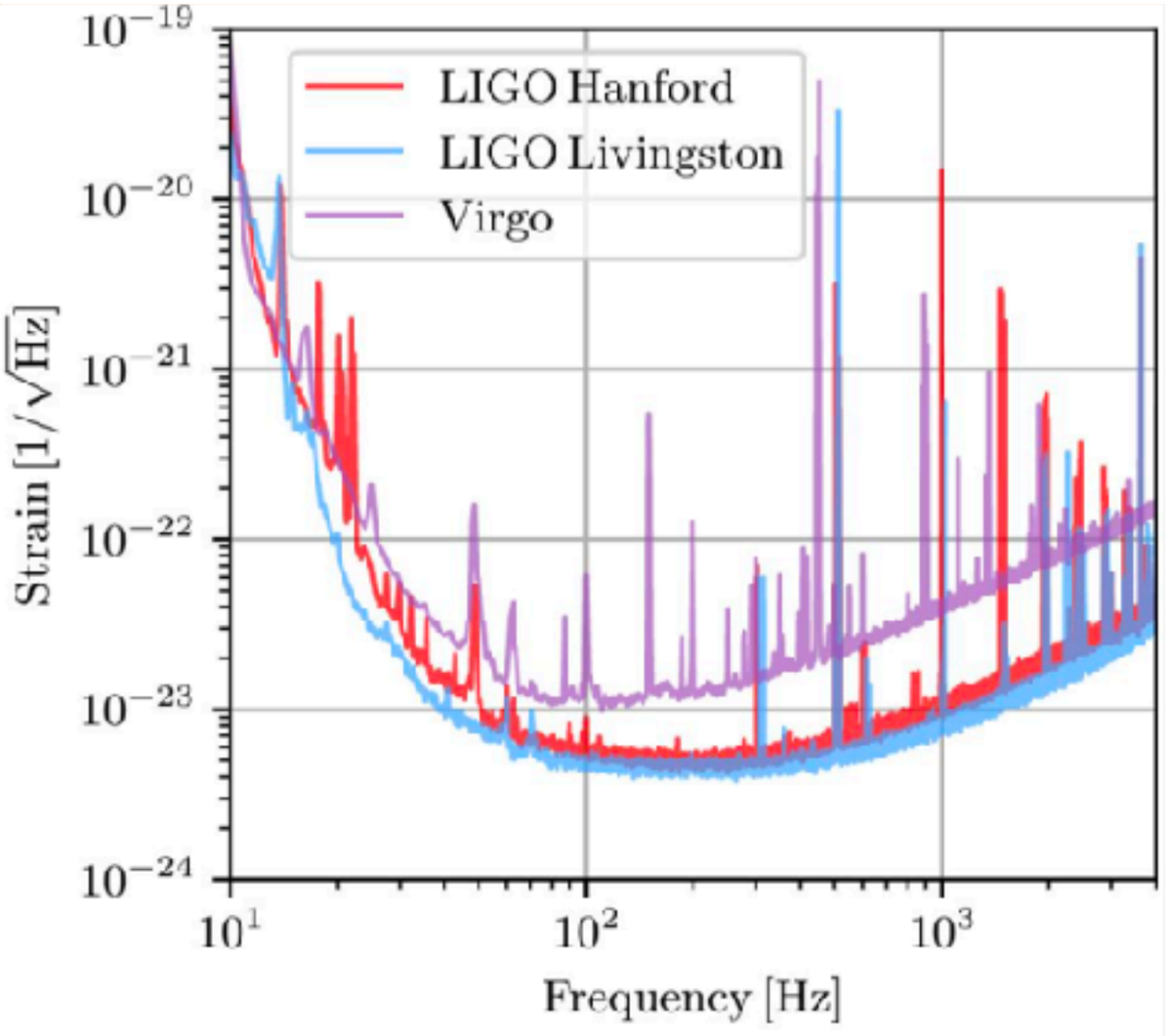


Image from Abbott et al (2020) GWTC-2 2010.14527

Noise backgrounds

Non-stationarity

The detector sensitivity is not constant, this can happen rapidly or slowly

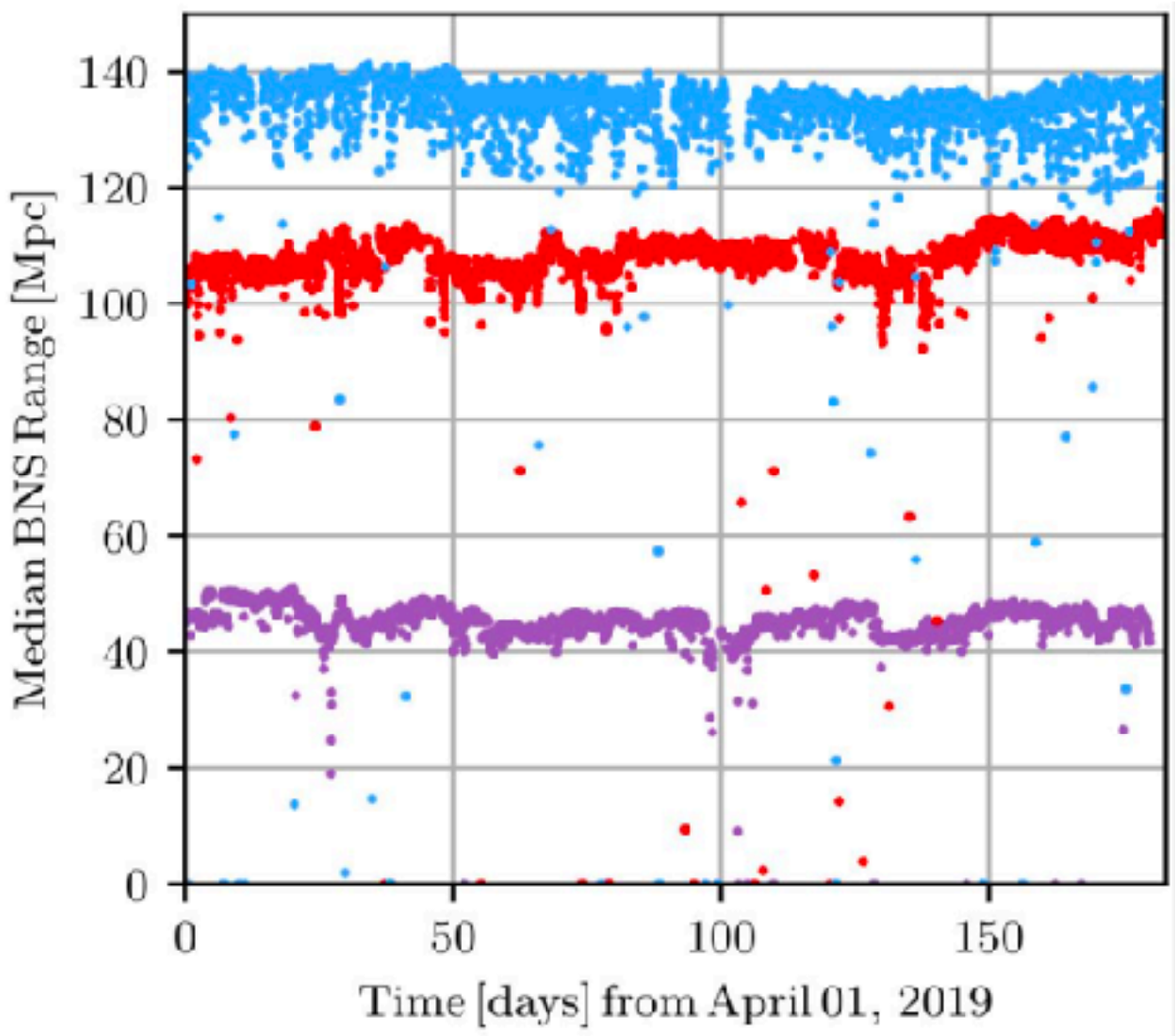
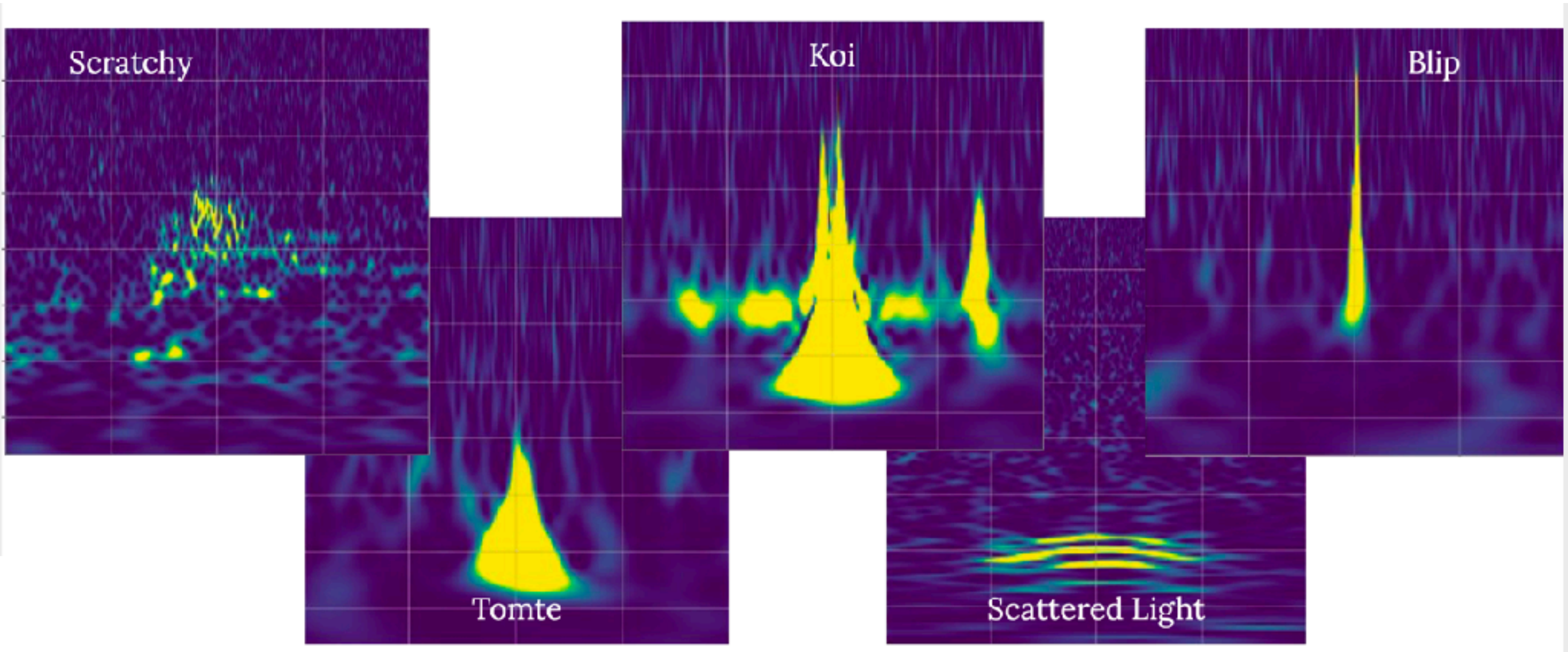


Image from Abbott et al (2020) GWTC-2 2010.14527

Noise backgrounds

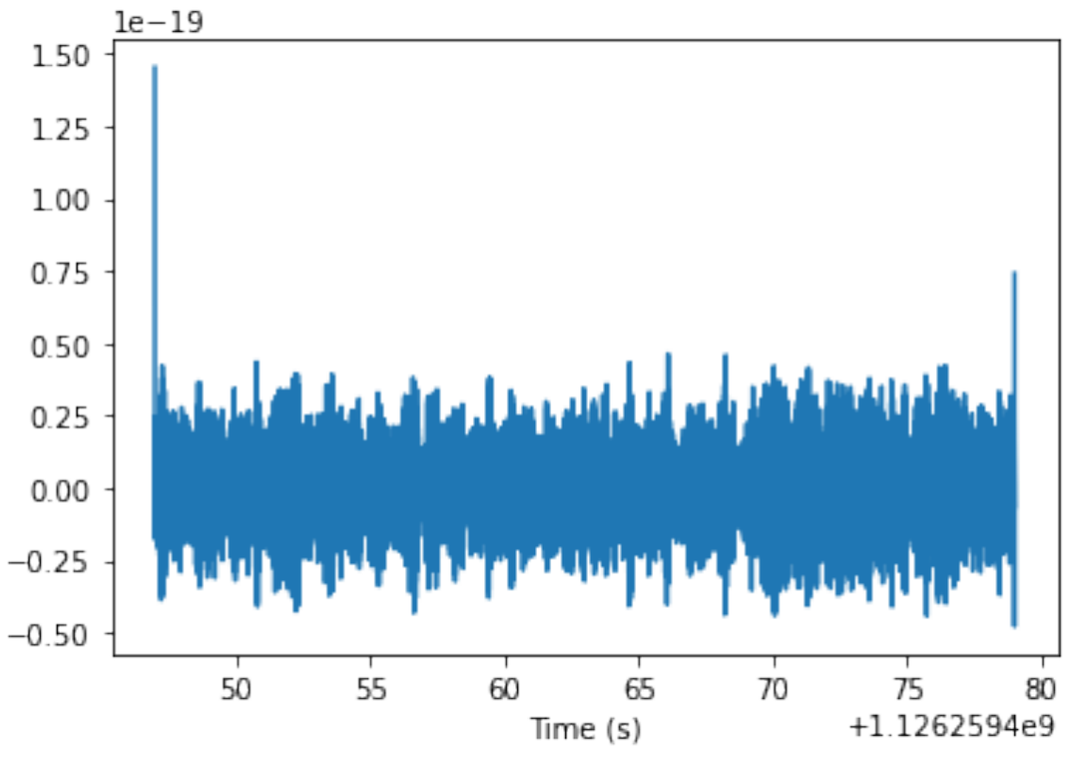
Non-Gaussian Glitches



Matched Filtering (1) - data

```
from pycbc.catalog import Merger
from pycbc.filter import resample_to_delta_t, highpass

# As an example we use the GW150914 data
merger = Merger("GW150914")

# Get the data from the Hanford detector
strain = merger.strain('H1')

# Remove the low frequency content and downsample the data to 2048Hz
strain = highpass(strain, 15.0)
strain = resample_to_delta_t(strain, 1.0/2048)
```

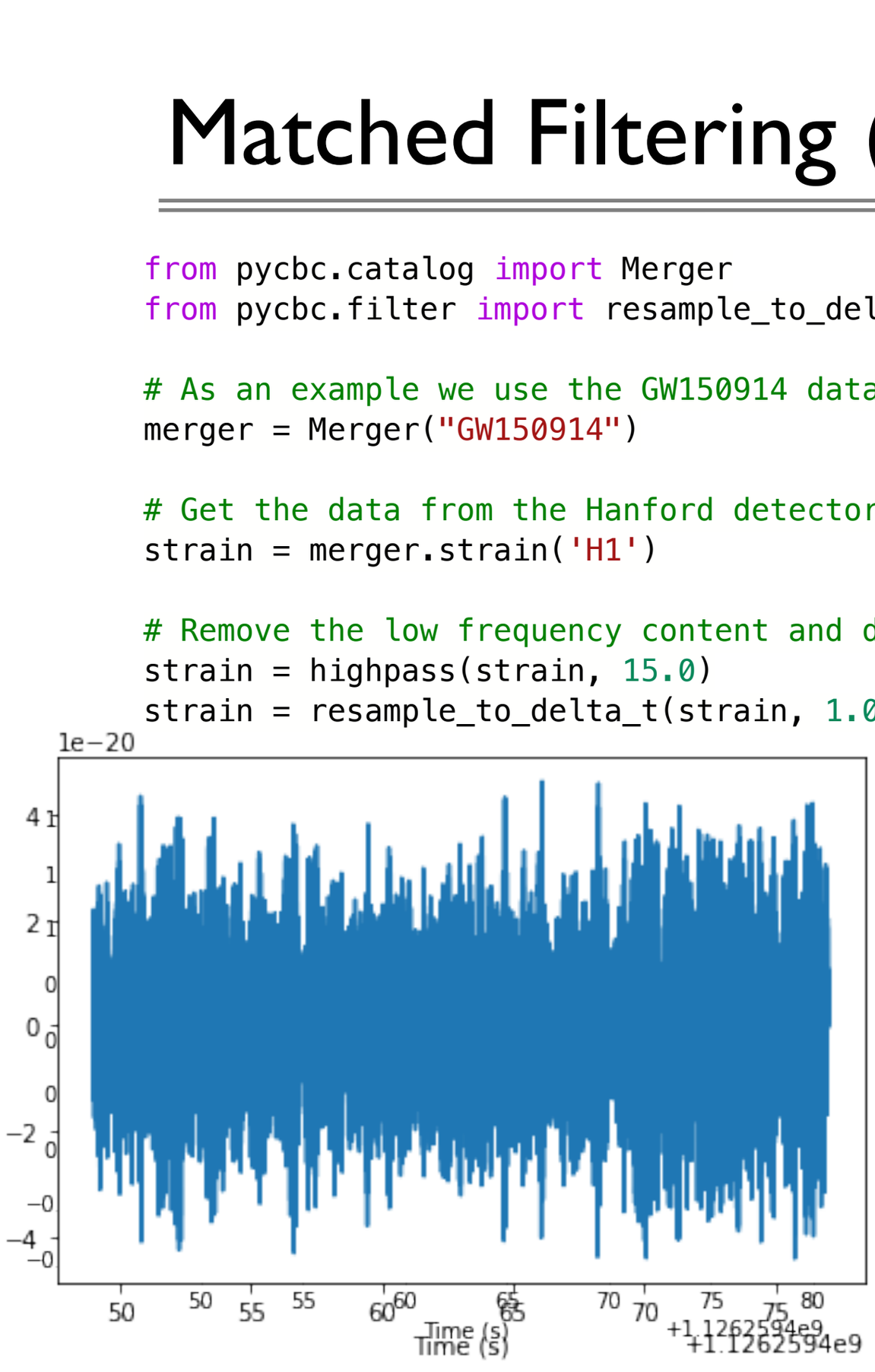
Read local file:
pycbc.frame.read_frame(file, channel_name)



Matched Filtering (1) - data

```
from pycbc.catalog import Merger
from pycbc.filter import resample_to_delta_t, highpass

# As an example we use the GW150914 data
merger = Merger("GW150914")

# Get the data from the Hanford detector
strain = merger.strain('H1')

# Remove the low frequency content and downsample the data to 2048Hz
strain = highpass(strain, 15.0)
strain = resample_to_delta_t(strain, 1.0/2048)
```

Read local file:
pycbc.frame.read_frame(file, channel_name)




```
 Remove 2 seconds of data from both the beginning and end
conditioned = strain.crop(2, 2)

pylab.plot(conditioned.sample_times, conditioned)
pylab.xlabel('Time (s)')
pylab.show()
```

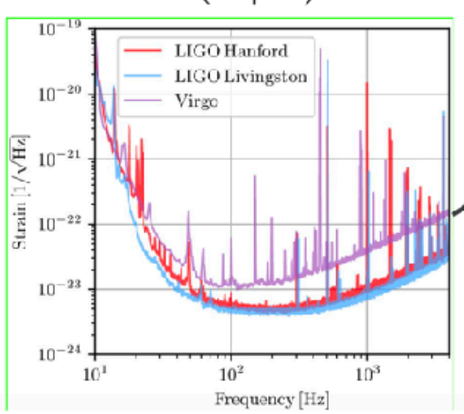
Matched Filtering (2) - PSD

Optimal for signals:

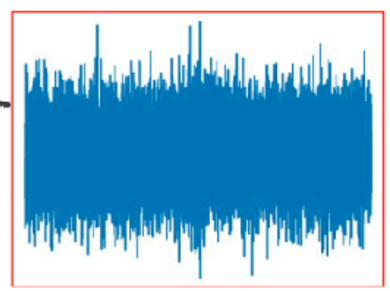
- in stationary Gaussian noise
- with known PSD

(Wainstein and Zubakov, 1962)

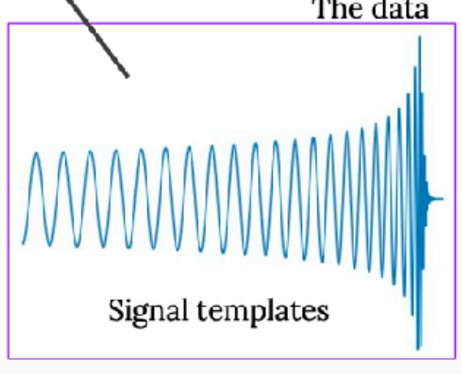
$$(s|h) = 4\Re \int_0^\infty \frac{\tilde{s}(f)\tilde{h}^\star(f)}{S_h(f)} \mathrm{d}f$$



PSD



The data



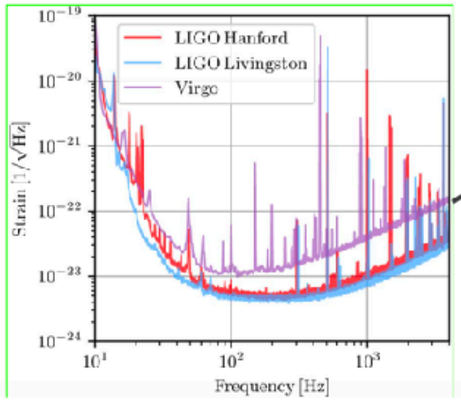
Signal templates

Matched Filtering (2) - PSD

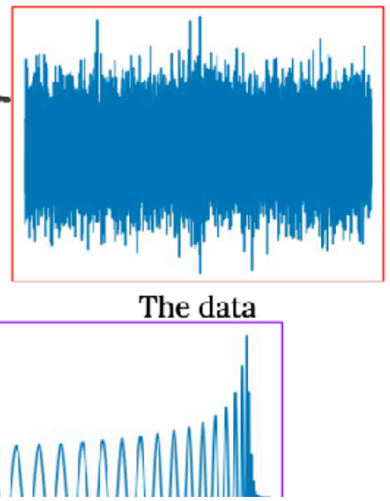
Optimal for signals:
- in stationary Gaussian noise
- with known PSD

(Wainstein and Zubakov, 1962)

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



PSD

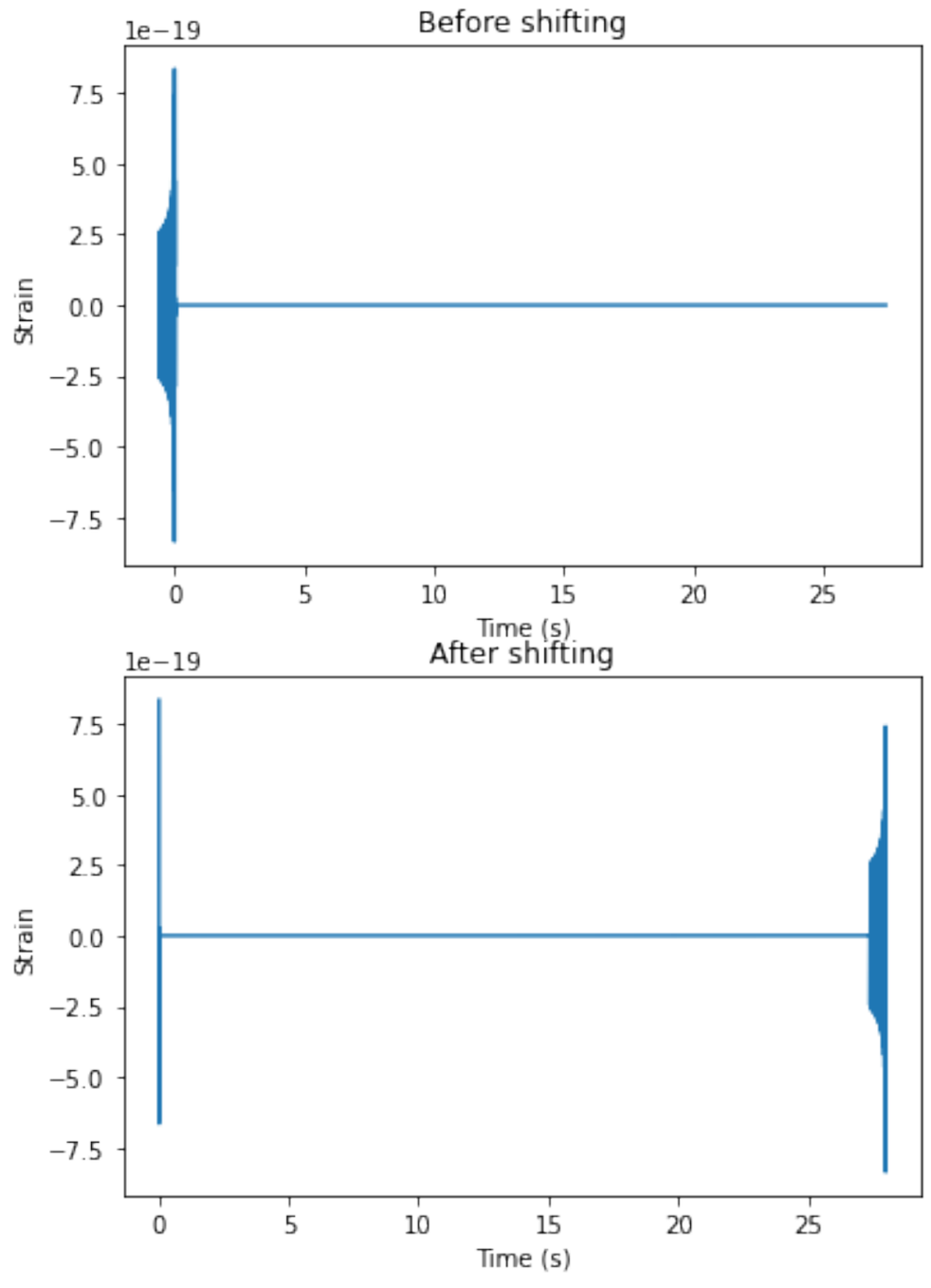


The data

```
from pycbc.psd import interpolate, inverse_spectrum_truncation
# Estimate the power spectral density

# We use 4 second samples of our time series in Welch method.
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 15 Hz, and will have low values
# below this we need to inform the function to not include frequencies
# below this frequency.
psd = inverse_spectrum_truncation(psd, int(4 * conditioned.sample_rate),
                                  low_frequency_cutoff=15)
```

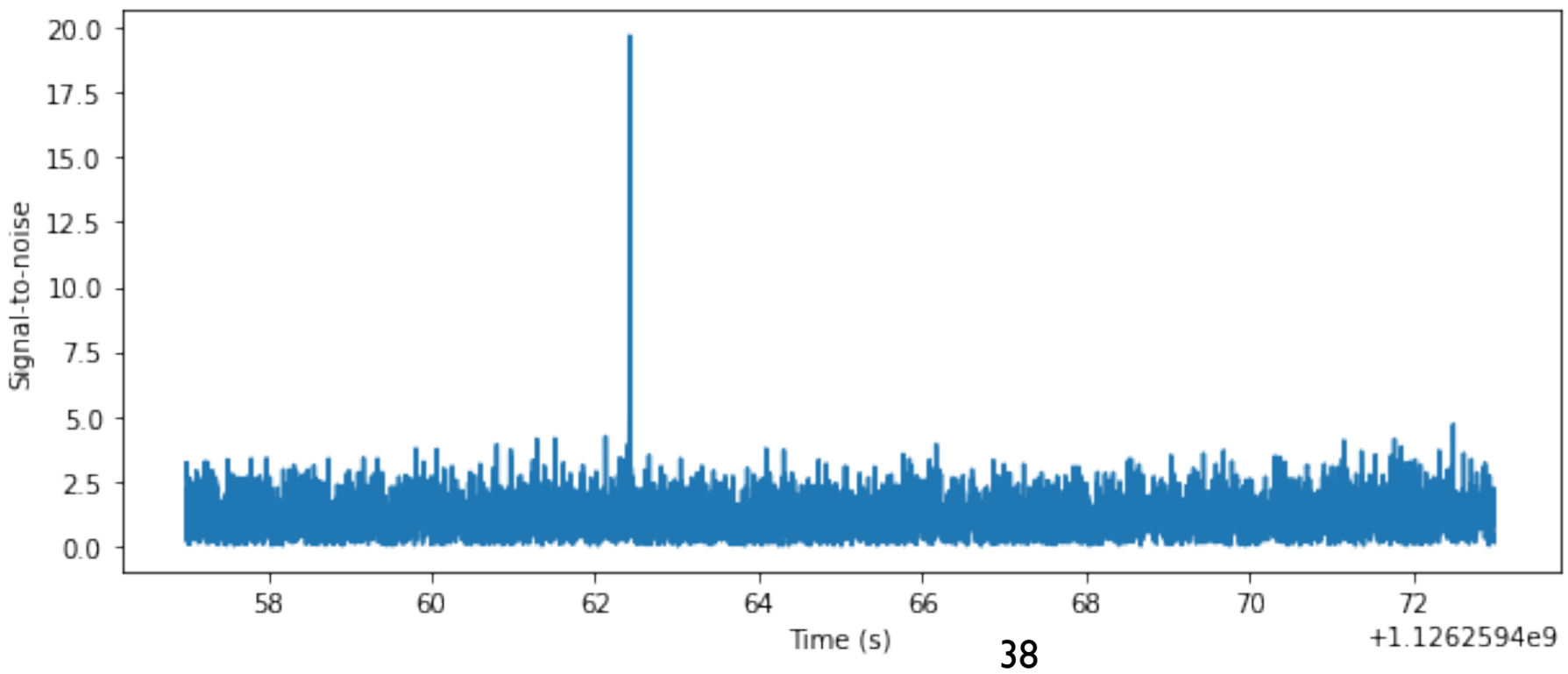
Matched Filtering (3) - waveform

```
from pycbc.waveform import get_td_waveform

m = 36 # 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))

# Let's plot the signal before and after shifting

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')
```



Matched Filtering (3) - snr time series

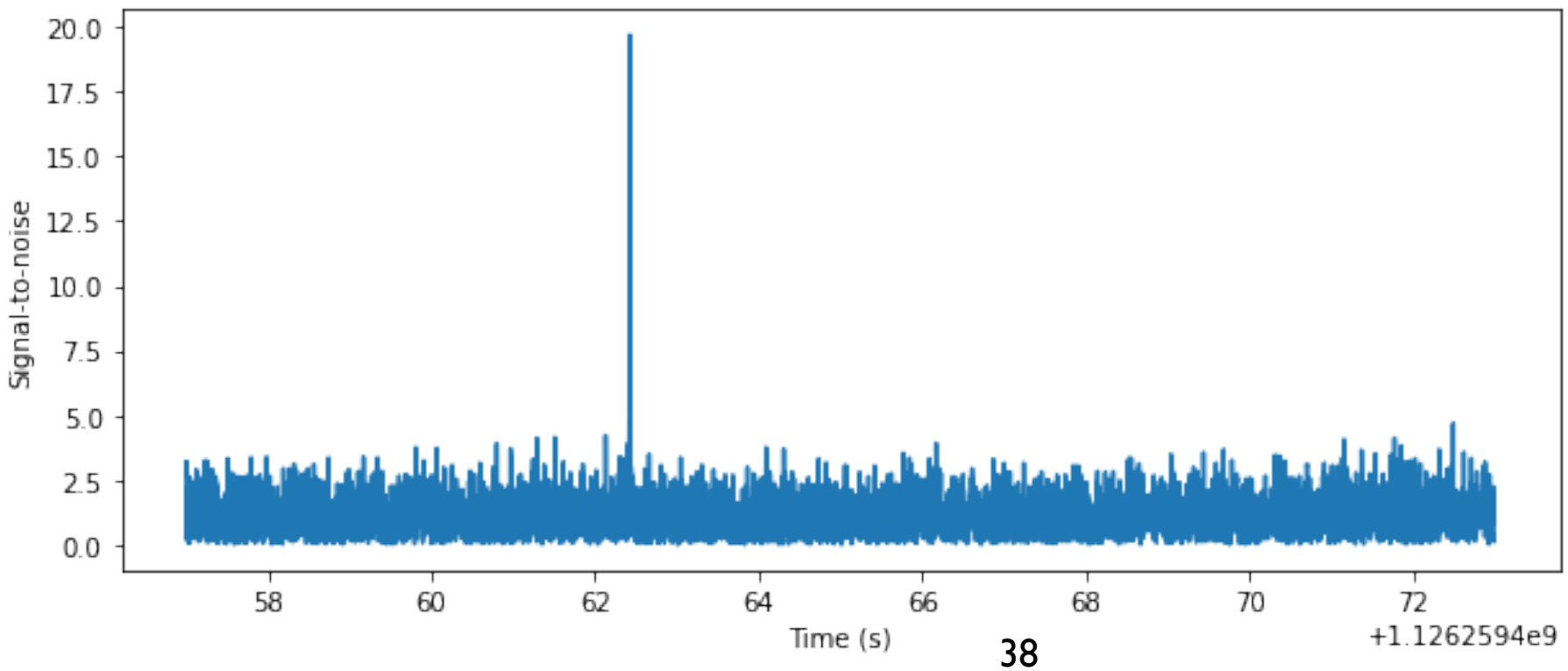
```
from pycbc.filter import matched_filter
import numpy

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 seonds 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)
```



Matched Filtering (3) - snr time series

```
from pycbc.filter import matched_filter
import numpy

snr = matched_filter(template, conditioned,
                     
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)))
```

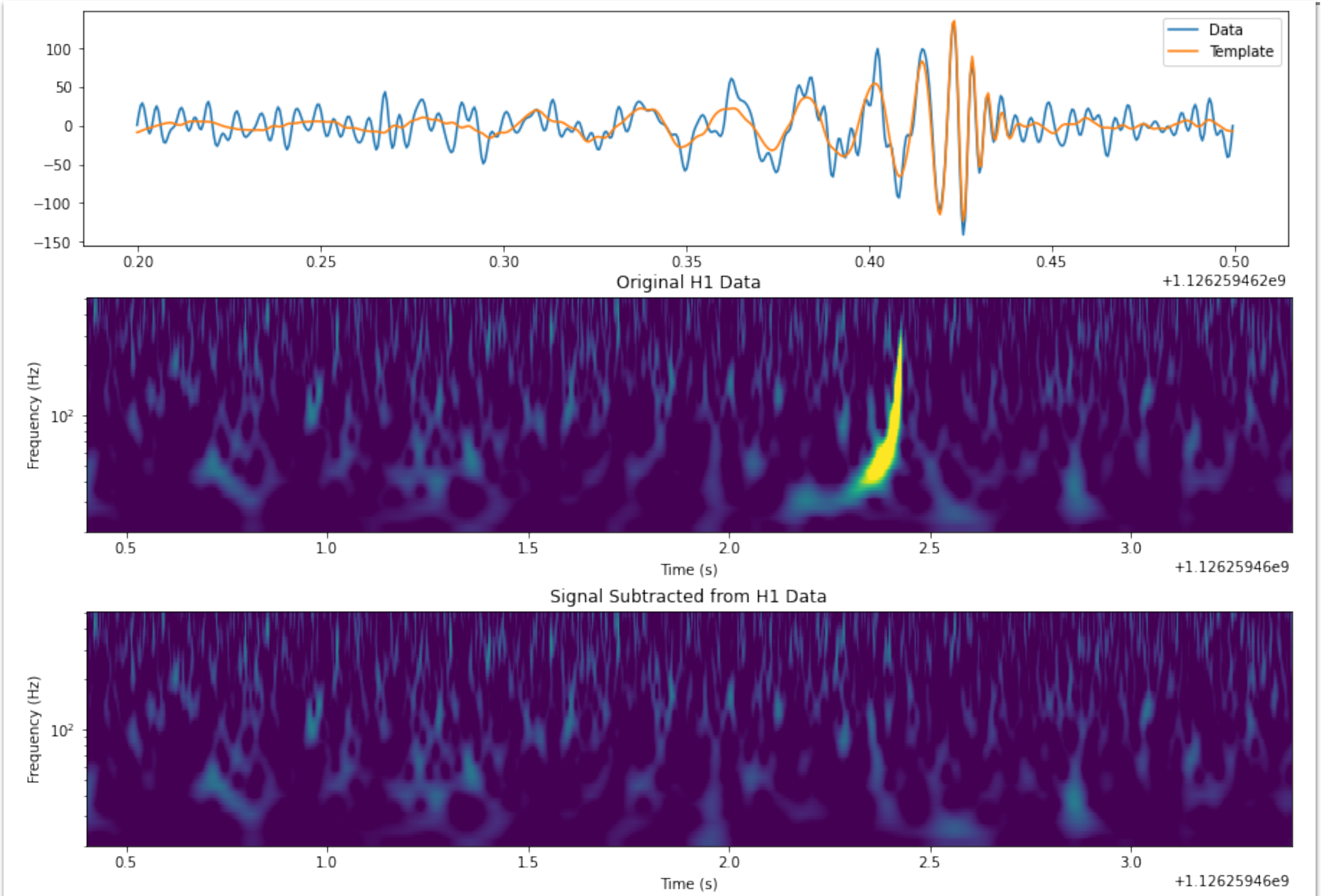
```
We found a signal at 1126259462.4248047s with SNR 19.677089013145903
```



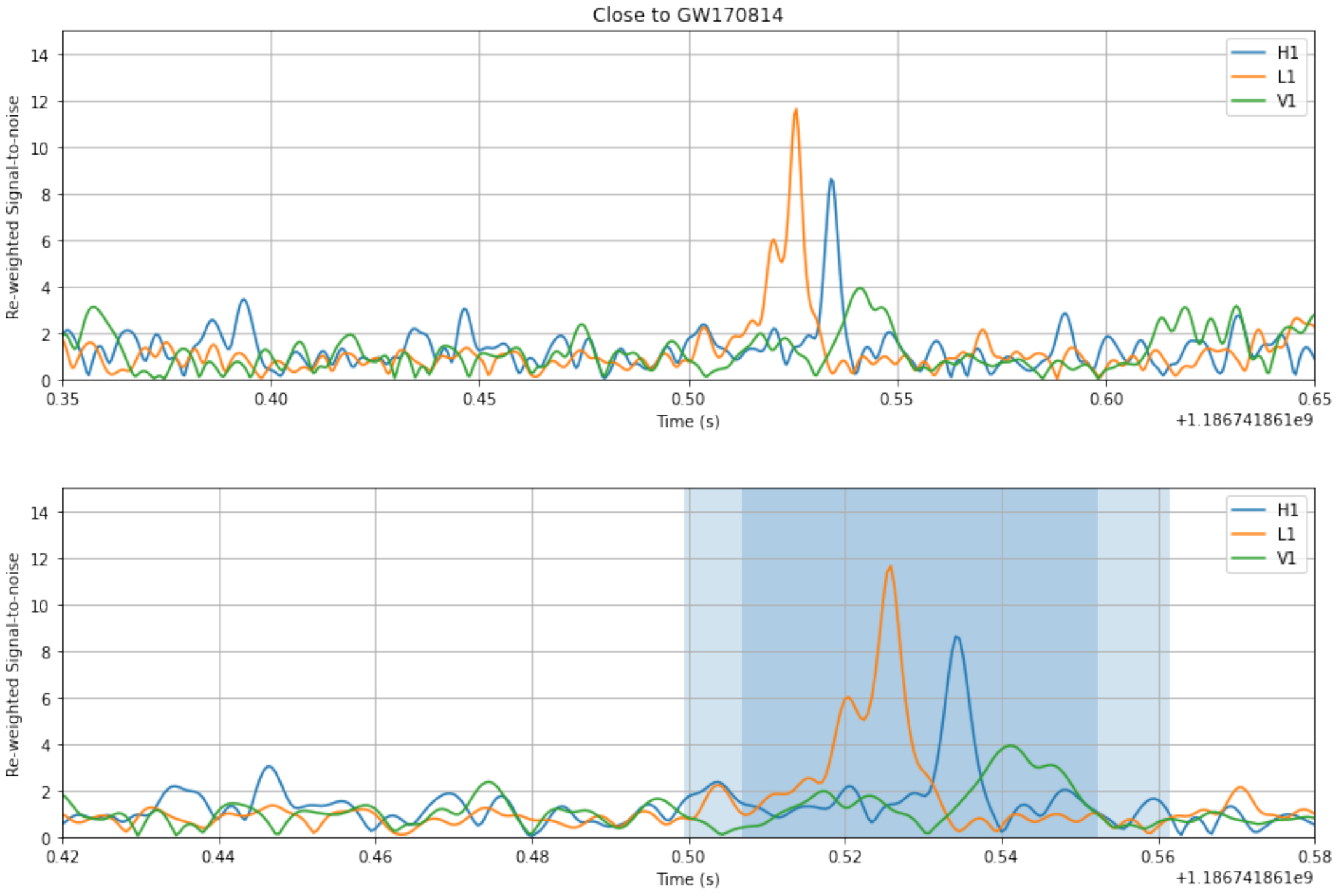
Matched Filtering (4) - snr time series

```
from pycbc.filter import sigma
# 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
```

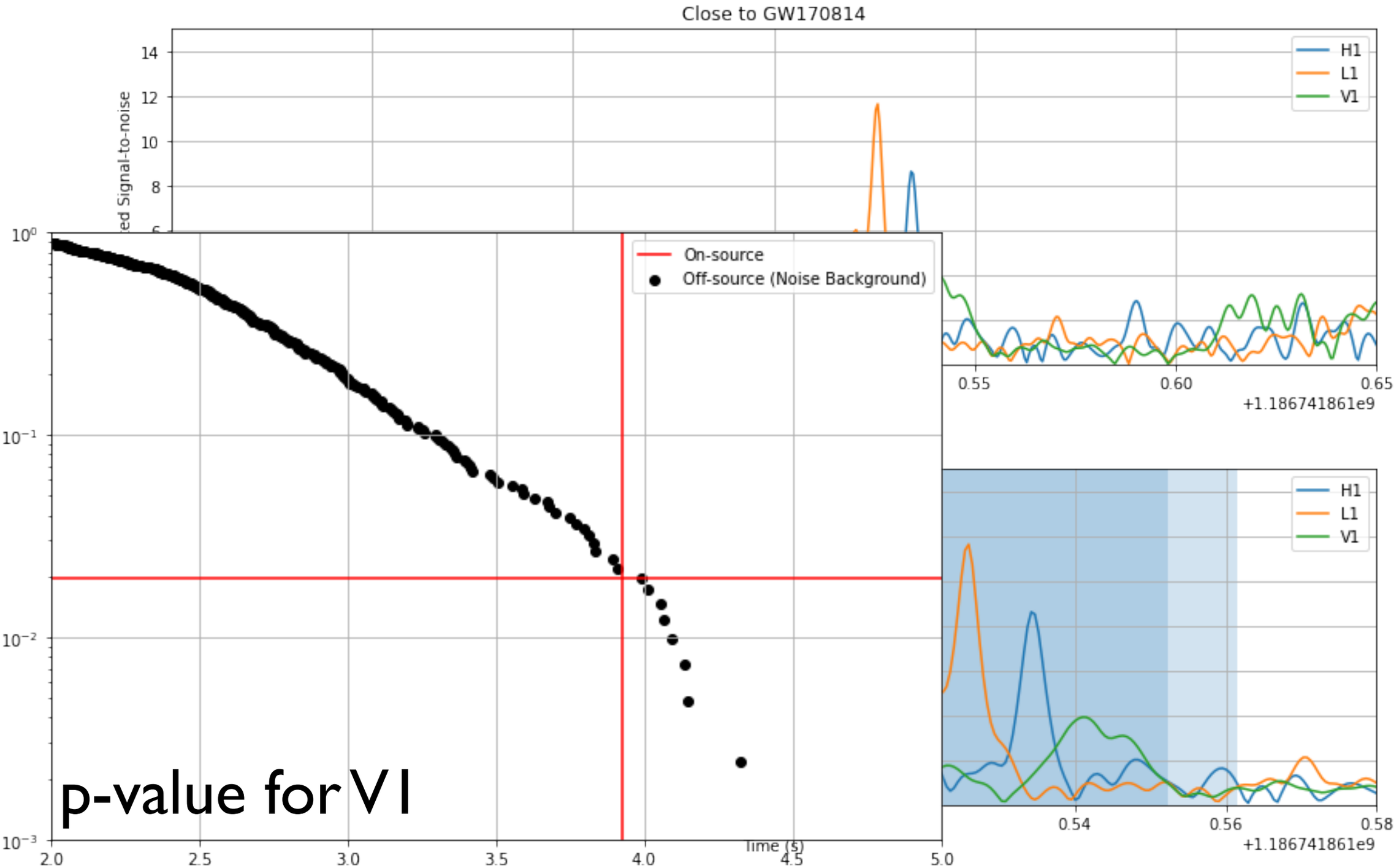
Matched Filtering (4) - snr time series



Signal Consistency and Significance



Signal Consistency and Significance



Bayesian Inference

Bayes' theorem

The likelihood could be the function of errors

Posterior $$p(\theta|d) = \frac{p(d|\theta)}{p(d)} \cdot p(\theta)$$

Prior choices can influence results

$$= \frac{p(d|\theta)}{\int p(d|\theta)p(\theta)d\theta} \cdot p(\theta)$$

The evidence is unimportant for parameter estimation (but not model selection !)

$$p(\theta|d) \sim p(d|\theta)p(\theta)$$

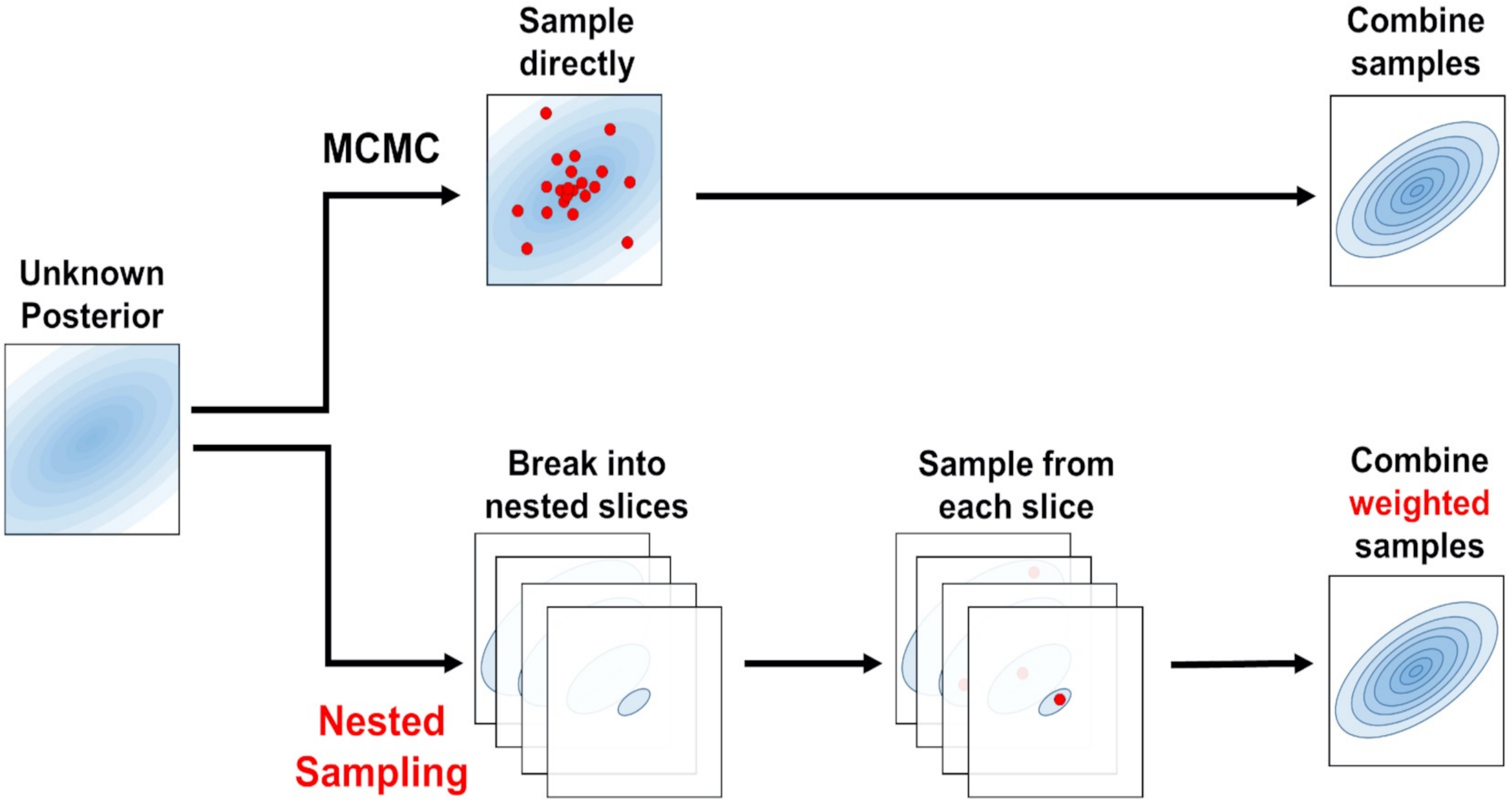
Prior, p(θ) : the distribution of the parameter(s) before any data is observed

Likelihood, p(d|θ) : the distribution of the observed data conditional on its parameters

Posterior, p(θ|d): the distribution of the parameter(s) after taking into account the observed data

Model evidence, p(d):the distribution of the observed data marginalized over the parameter(s)

Figure 1. A schematic representation of the different approaches MCMC methods and nested sampling methods take to ...



Mon Not R Astron Soc, Volume 493, Issue 3, April 2020, Pages 3132–3158, https://doi.org/10.1093/mnras/staa278

Posterior samples from PE

```
import h5py
import pandas as pd
import corner


 label = 'GW150914'

 # if you do not have wget installed, simply download manually
 # https://dcc.ligo.org/LIGO-P1800370/public/GW150914_GWTC-1.hdf5
 # from your browser
 ! wget https://dcc.ligo.org/LIGO-P1800370/public/{label}_GWTC-1.hdf5
```

```
posterior_file = './'+label+'_GWTC-1.hdf5'
posterior = h5py.File(posterior_file, 'r')


print('This file contains four datasets: ',posterior.keys())
```

```
This file contains four datasets:  <KeysViewHDF5
['IMRPhenomPv2_posterior', 'Overall_posterior', 'SEOBNRv3_posterior',
‘prior']>
```

Posterior samples from PE

```
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')
```

- `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.

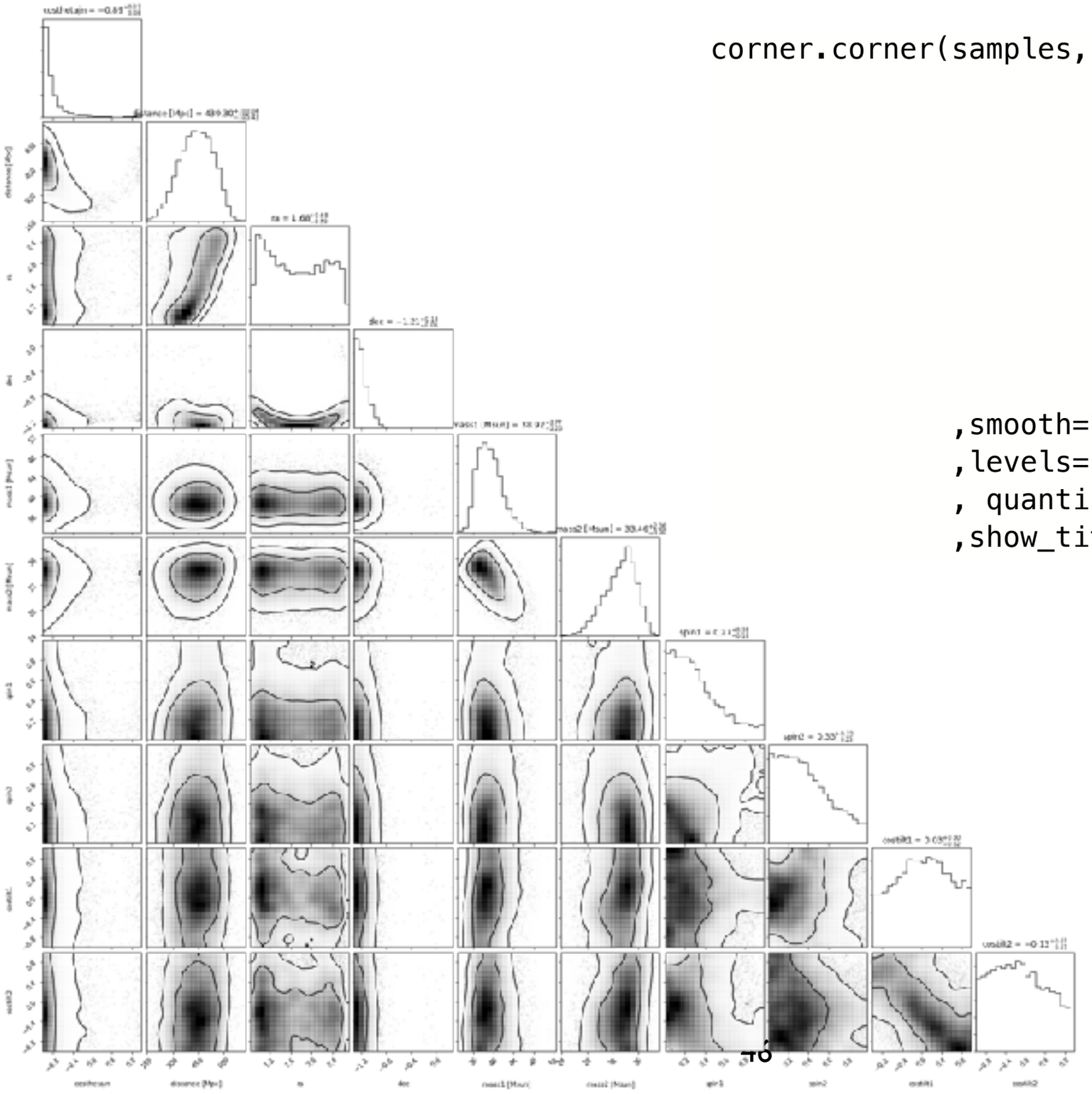
Posterior samples from PE

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

	costheta_jn	luminosity_distance_Mpc	right_ascension	declination	m1_detector_frame_Msun	m2_detector_frame_Msun	spin1	spin2	costilt1	costilt2
0	-0.976633	517.176717	1.456176	-1.257815	39.037380	37.044563	0.417147	0.857740	-0.280624	0.403853
1	-0.700404	401.626864	2.658802	-0.874661	34.620096	34.184416	0.125709	0.260679	-0.757349	-0.312285
2	-0.840752	369.579071	1.106548	-1.136396	37.894343	33.970520	0.581047	0.926893	0.649781	-0.510843
3	-0.583657	386.935268	2.077180	-1.245351	36.412973	35.684463	0.235808	0.094391	0.116578	-0.720505
4	-0.928271	345.104345	0.993604	-1.069243	39.477251	31.645008	0.511521	0.868009	-0.438237	0.269333
...	...	...	...	...	...	...	...	...	...	...
8345	-0.691637	306.985025	1.485646	-1.269228	37.561962	33.355792	0.484003	0.627191	0.194507	-0.408345
8346	-0.834615	452.649414	2.065352	-1.265618	37.824298	36.674075	0.589654	0.650758	-0.737792	0.875384
8347	-0.911463	448.930876	1.536913	-1.257956	38.063291	35.757913	0.708407	0.714805	0.852085	-0.797475
8348	-0.856914	561.020036	2.367289	-1.211824	44.884396	31.592433	0.389284	0.521304	-0.251461	0.830526
8349	-0.919556	519.641782	1.916675	-1.250801	37.275183	35.445032	0.391824	0.516908	-0.705305	0.600727

8350 rows × 10 columns

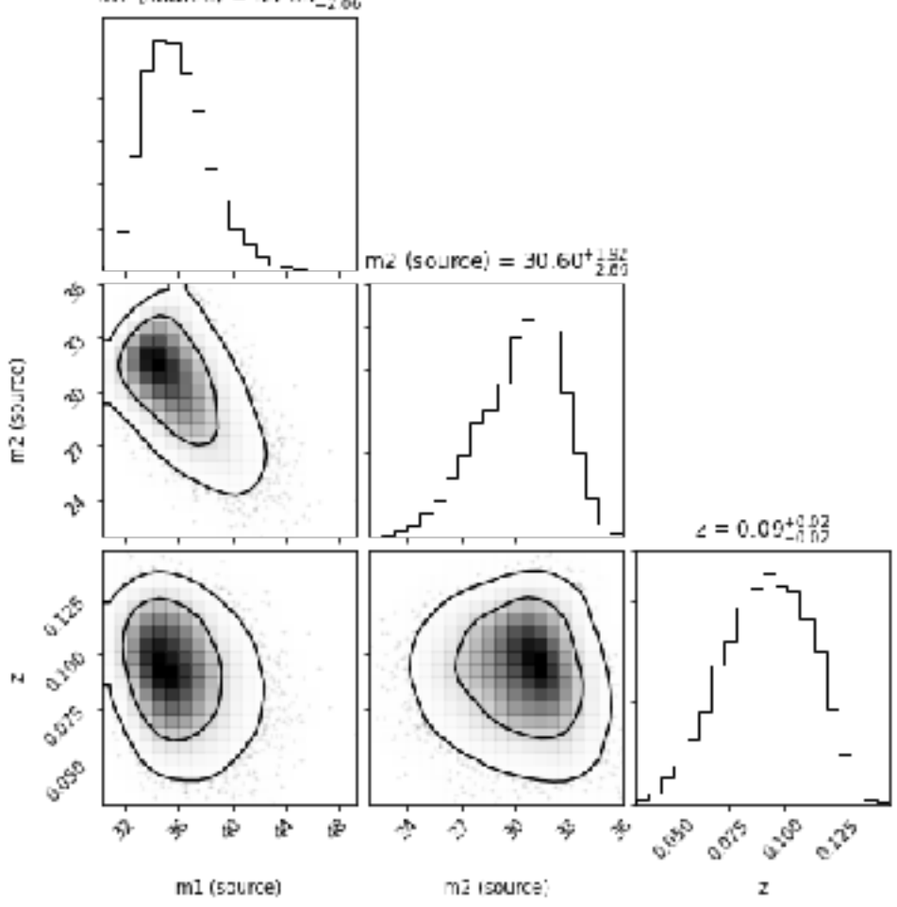
Posterior samples from PE



```
corner.corner(samples,labels=['costhetajn',
                              'distance [Mpc]',
                              'ra',
                              'dec',
                              'mass1 [Msun]',
                              'mass2 [Msun]',
                              'spin1',
                              'spin2',
                              'costilt1',
                              'costilt2']
,smooth=1.0
,levels=(0.68,0.95)
, quantile=(0.16,0.84)
,show_titles=True);
```

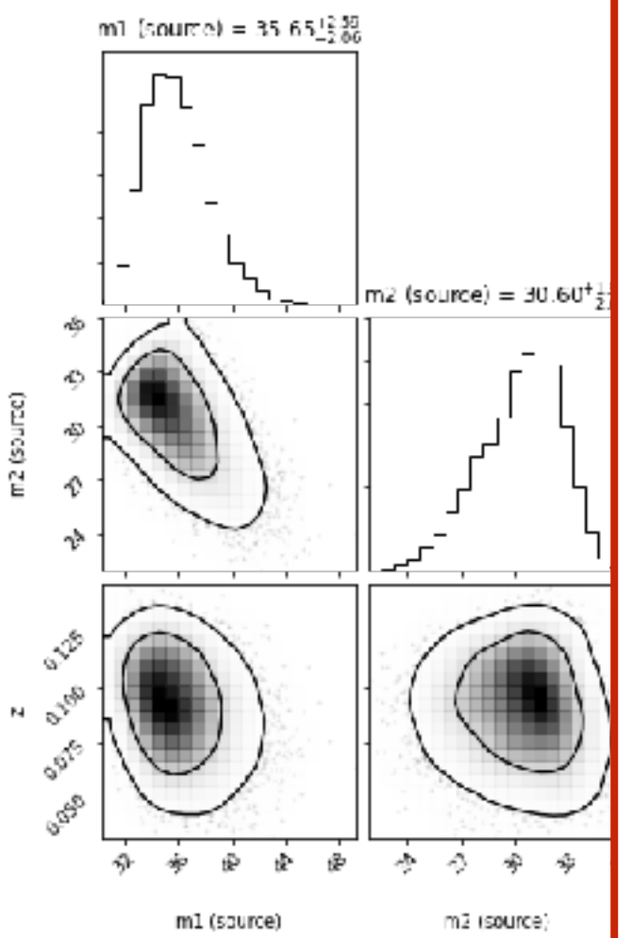
Computing new quantities from posterior samples

```
import astropy.units as u
from astropy.cosmology import Planck15, z_at_value

z = np.array([z_at_value(Planck15.luminosity_distance, dist * u.Mpc) for dist in
samples['luminosity_distance_Mpc']])

samples['m1_source_frame_Msun']=samples['m1_detector_frame_Msun']/(1.0+z)
samples['m2_source_frame_Msun']=samples['m2_detector_frame_Msun']/(1.0+z)
samples['redshift']=z
```



Computing new quantities from posterior samples

```
import astropy.units as u
from astropy.cosmology import Planck15, z_at_value

z = np.array([z_at_value(Planck15.luminosity_distance, dist * u.Mpc) for dist in
samples['luminosity_distance_Mpc']])

samples['m1_source_frame_Msun']=samples['m1_detector_frame_Msun']/(1.0+z)
samples['m2_source_fram
samples['redshift']=z
```




```
import bilby
# calculate the detector frame chirp mass
mchirp = ((samples['m1_detector_frame_Msun'] *
samples['m2_detector_frame_Msun'])**(3./5))/\
        (samples['m1_detector_frame_Msun'] +
samples['m2_detector_frame_Msun'])**(1./5)
# initialize a SampleSummary object to describe the chirp mass posterior
samples
chirp_mass_samples_summary =
bilby.core.utils.SamplesSummary(samples=mchirp, average='median')
print('The median chirp mass = {}
Msun'.format(chirp_mass_samples_summary.median))
print('The 90% confidence interval for the chirp mass is {} - {}
Msun'.format(chirp_mass_samples_summary.lower_absolute_credible_interval,
chirp_mass_samples_summary.upper_absolute_credible_interval))
```

```
The median chirp mass = 31.23055308109465 Msun
The 90% confidence interval for the chirp mass is
29.655877108464615 - 32.97324559242388 Msun
```

PE w/ Bilby - 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_duratio
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)
```

PE w/ Bilby - data

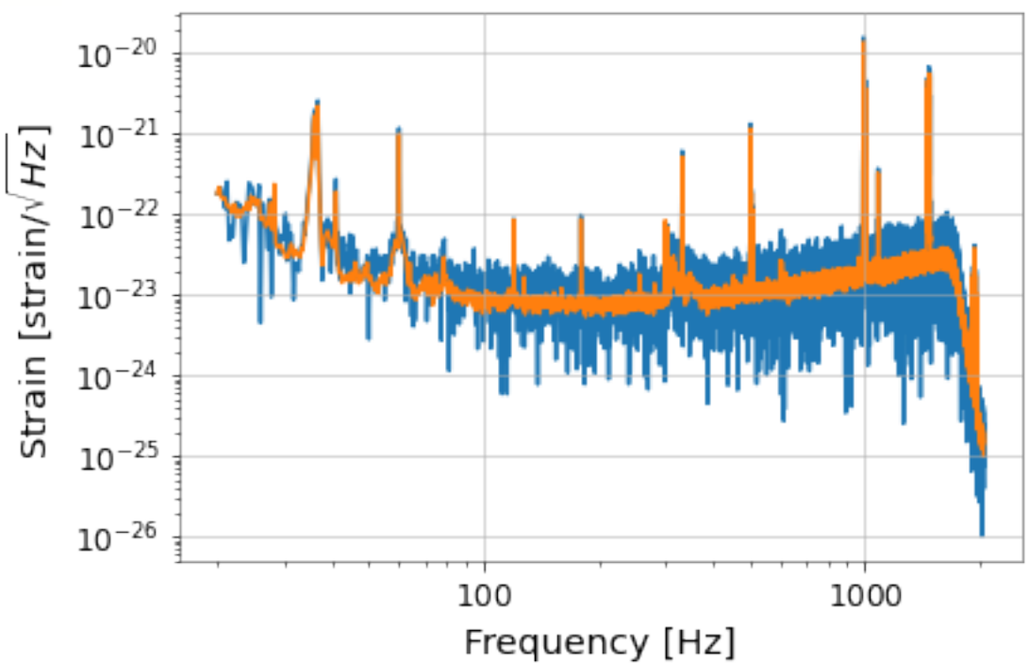
* Set up empty interferometers

```
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")
```

PE w/ Bilby - data

```
    *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()
```



PE w/ Bilby - priors

```
prior = bilby.core.prior.PriorDict()
prior['chirp_mass'] = Uniform(name='chirp_mass', minimum=30.0,maximum=32.5)
prior['mass_ratio'] = Uniform(name='mass_ratio', minimum=0.5, maximum=1)
prior['phase'] = Uniform(name="phase", minimum=0, maximum=2*np.pi)
prior['geocent_time'] = Uniform(name="geocent_time", minimum=time_of_event-0.1,
maximum=time_of_event+0.1)
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['dec'] =  -1.2232
prior['ra'] =  2.19432
prior['theta_jn'] =  1.89694
prior['psi'] =  0.532268
prior['luminosity_distance'] = 412.066
```

PE w/ Bilby - priors

```
prior = bilby.core.prior.PriorDict()
prior['chirp_mass'] = Uniform(name='chirp_mass', minimum=30.0,maximum=32.5)
prior['mass_ratio'] = Uniform(name='mass_ratio', minimum=0.5, maximum=1)
prior['phase'] = Uniform(name="phase", minimum=0, maximum=2*np.pi)
prior['geocent_time'] = Uniform(name="geocent_time", minimum=time_of_event-0.1,
maximum=time_of_event+0.1)
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['dec'] =  -1.2232
prior['ra'] =  2.19432
prior['theta_jn'] =  1.89694
prior['psi'] =  0.532268
prior['luminosity_distance'] =
```

Essence of the Bayesian Idea

- Bayes' rule(theorem)

$$p(M|D) = \frac{p(D|M)\,p(M)}{p(D)} \qquad p(M,D) = p(M|D)p(D) = p(D|M)p(M)$$

- Improved belief is the product of initial belief and the probability that initial belief generate the observed data

$$p(M,\boldsymbol{\theta}|D,I) = \frac{p(D|M,\boldsymbol{\theta},I)\,p(M,\boldsymbol{\theta}|I)}{p(D|I)}$$

$$p(M,\boldsymbol{\theta}|I) = p(\boldsymbol{\theta}|M,I)\,p(M|I)$$

이형원교수님
여름학교 강의중에서

PE w/ Bilby - Likelihood

```
# First, put our "data" created above into a list of intererometers (the
order is arbitrary)
interferometers = [H1, L1]

# Next create a dictionary of arguments which we pass into the
LALSimulation waveform - we specify the waveform approximant here
waveform_arguments = dict(
    waveform_approximant='IMRPhenomPv2', reference_frequency=100.,
catch_waveform_errors=True)

# Next, create a waveform_generator object. This wraps up some of the jobs
of converting between parameters etc
waveform_generator = bilby.gw.WaveformGenerator(
    frequency_domain_source_model=bilby.gw.source.lal_binary_black_hole,
    waveform_arguments=waveform_arguments,
    parameter_conversion=convert_to_lal_binary_black_hole_parameters)

# 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)
```

PE w/ Bilby - run

```
result_short = bilby.run_sampler(
    likelihood, prior, sampler='dynesty', outdir='short', label="GW150914",
    conversion_function=bilby.gw.conversion.generate_all_bbh_parameters,
    sample="unif", nlive=500, dlogz=3)
    # Arguments are used to make things fast - not recommended for general use
```

dynesty: https://arxiv.org/abs/1904.02180
https://dynesty.readthedocs.io/en/latest/dynamic.html

Samplers:
 Nested Sampling: dynesty, nestle, cpnest
MCMC : bilby_mcmc, emcee, ptemcee, pymc3

PE w/ Bilby - run

```
04:51 bilby INFO    : Running for label 'GW150914', output will be saved to 'short'
04:51 bilby INFO    : Using lal version 7.1.2
04:51 bilby INFO    : Using lal git version Branch: None;Tag: lalsuite-v6.82;Id: cf792129c2473f42ce6c6ee21d8234254cefd337;;Builder: Unknown User <>;Repository status: UNCLEAN: Modified
working tree
04:51 bilby INFO    : Using lalsimulation version 2.5.1
04:51 bilby INFO    : Using lalsimulation git version Branch: None;Tag: lalsuite-v6.82;Id: cf792129c2473f42ce6c6ee21d8234254cefd337;;Builder: Unknown User <>;Repository status: UNCLEAN:
Modified working tree
04:51 bilby INFO    : Search parameters:
04:51 bilby INFO    :   chirp_mass = Uniform(minimum=30.0, maximum=32.5, name='chirp_mass', latex_label='$\\mathcal{M}$', unit=None, boundary=None)
04:51 bilby INFO    :   mass_ratio = Uniform(minimum=0.5, maximum=1, name='mass_ratio', latex_label='$q$', unit=None, boundary=None)
04:51 bilby INFO    :   time_jitter = Uniform(minimum=-0.000244140625, maximum=0.000244140625, name=None, latex_label=None, unit=None, boundary='periodic')
04:51 bilby INFO    :   phase = 0.0
04:51 bilby INFO    :   geocent_time = 1126259460.3999023
04:51 bilby INFO    :   a_1 = 0.0
04:51 bilby INFO    :   a_2 = 0.0
04:51 bilby INFO    :   tilt_1 = 0.0
04:51 bilby INFO    :   tilt_2 = 0.0
04:51 bilby INFO    :   phi_12 = 0.0
04:51 bilby INFO    :   phi_jl = 0.0
04:51 bilby INFO    :   dec = -1.2232
04:51 bilby INFO    :   ra = 2.19432
04:51 bilby INFO    :   theta_jn = 1.89694
04:51 bilby INFO    :   psi = 0.532268
04:51 bilby INFO    :   luminosity_distance = 412.066
04:51 bilby INFO    : Generating frequency domain strain from given time domain strain.
04:51 bilby INFO    : Applying a tukey window with alpha=0.1, roll off=0.2
04:51 bilby INFO    : Single likelihood evaluation took 1.482e-02 s
0it [00:00, ?it/s]04:51 bilby INFO    : Using sampler Dynesty with kwargs {'bound': 'multi', 'sample': 'unif', 'verbose': True, 'periodic': None, 'reflective': None,
'check_point_delta_t': 600, 'nlive': 500, 'first_update': None, 'walks': 100, 'npdim': None, 'rstate': None, 'queue_size': 1, 'pool': None, 'use_pool': None, 'live_points': None,
'logl_args': None, 'logl_kwargs': None, 'ptform_args': None, 'ptform_kwargs': None, 'enlarge': 1.5, 'bootstrap': None, 'vol_dec': 0.5, 'vol_check': 8.0, 'facc': 0.2, 'slices': 5,
'update_interval': 300, 'print_func': <bound method Dynesty._print_func of <bilby.core.sampler.dynesty.Dynesty object at 0x7fe0fd1bbed0>>, 'dlogz': 3, 'maxiter': None, 'maxcall': None,
'logl_max': inf, 'add_live': True, 'print_progress': True, 'save_bounds': False, 'n_effective': None, 'maxmcmc': 5000, 'nact': 5}
04:51 bilby INFO    : Checkpoint every check_point_delta_t = 600s
04:51 bilby INFO    : Using dynesty version 1.1
04:51 bilby INFO    : Resume file short/GW150914_resume.pickle does not exist.
04:51 bilby INFO    : Generating initial points from the prior
980it [00:51,  6.17it/s, bound:0 nc: 15 ncall:3.3e+03 eff:30.0% logz-ratio=266.09+/-0.08 dlogz:3.012>3]04:52 bilby INFO    : Written checkpoint file short/GW150914_resume.pickle
04:52 bilby INFO    : Writing 190 current samples to short/GW150914_samples.dat
982it [00:53, 18.23it/s, bound:0 nc:  1 ncall:3.3e+03 eff:45.3% logz-ratio=268.04+/-0.13 dlogz:0.005>3]04:52 bilby INFO    : Sampling time: 0:00:41.658335
04:52 bilby INFO    : Reconstructing marginalised parameters.

100%|██████████| 1482/1482 [00:46<00:00, 31.90it/s]04:53 bilby INFO    : Generating sky frame parameters.
100%|██████████| 1482/1482 [00:00<00:00, 2545.58it/s]
04:53 bilby INFO    : Computing SNRs for every sample.
100%|██████████| 1482/1482 [00:21<00:00, 70.41it/s]
04:54 bilby INFO    : Summary of results:
nsamples: 1482
ln_noise_evidence: -8534.562
ln_evidence: -8266.517 +/-  0.128
ln_bayes_factor: 268.045 +/-  0.128
```

PE w/ Bilby - results

```
1 result_short.posterior
```

	chirp_mass	mass_ratio	time_jitter	phase	geocent_time	a_1	a_2	tilt_1	tilt_2	phi_12	phi_jl	dec	ra	theta_jn	psi	luminosity_distance
0	31.304732	0.642621	0.000201	4.248570	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
1	30.431251	0.865419	-0.000241	5.339548	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
2	32.351273	0.823104	0.000208	1.453658	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
3	30.489016	0.749595	0.000115	1.913821	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
4	30.647587	0.683362	-0.000116	4.753118	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1477	31.527348	0.986097	-0.000080	5.085407	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
1478	31.527348	0.986097	-0.000080	2.036470	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
1479	31.527348	0.986097	-0.000080	1.935416	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
1480	31.527348	0.986097	-0.000080	2.048065	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066
1481	31.527348	0.986097	-0.000080	2.041310	1.126259e+09	0.0	0.0	0.0	0.0	0.0	0.0	-1.2232	2.19432	1.89694	0.532268	412.066

1482 rows × 50 columns

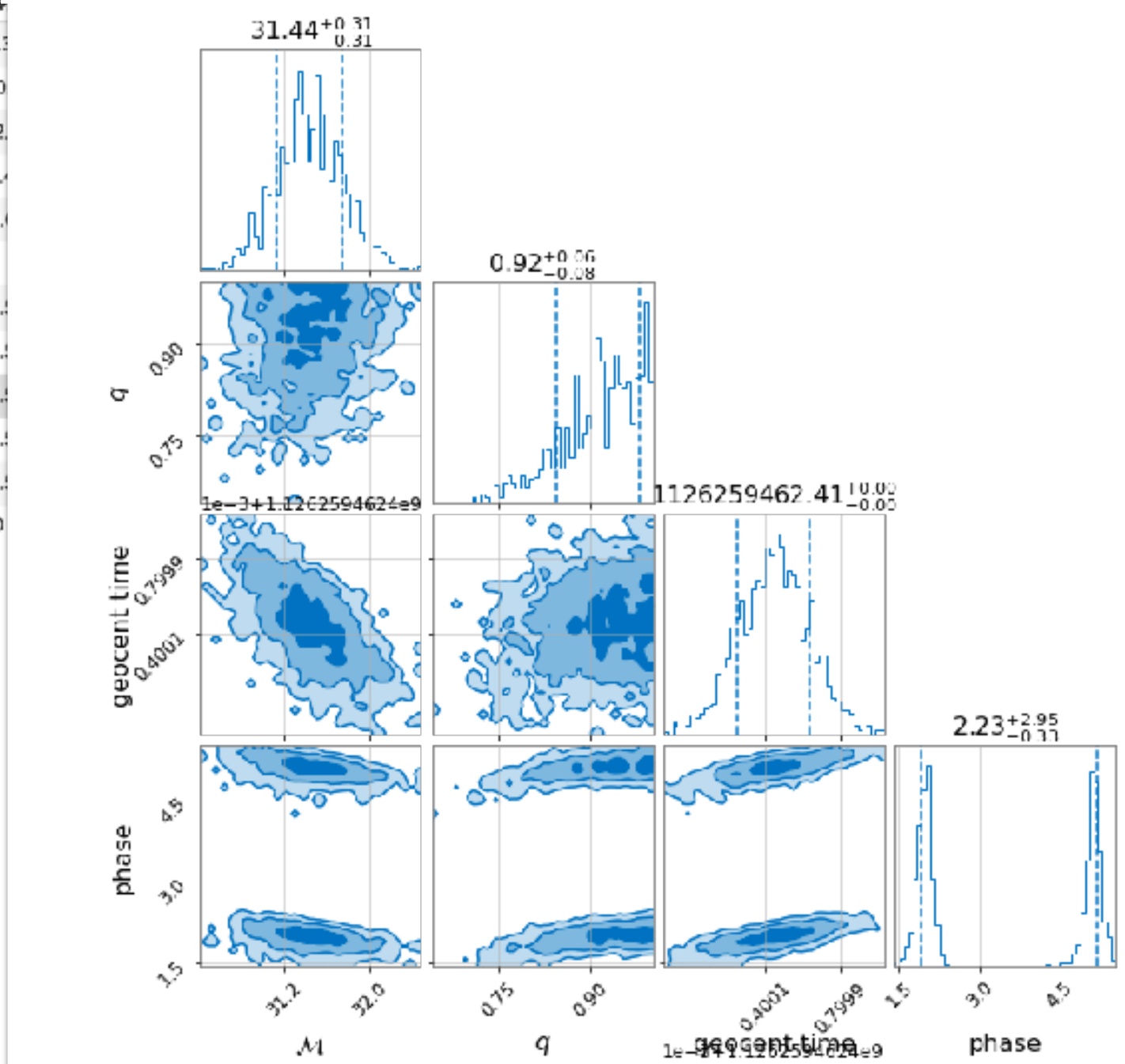
PE w/ Bilby - results

```
result_short.plot_corner(parameters=["chirp_mass", "mass_ratio",
"geocent_time", "phase"], prior=True)
```

```
1 result
```

	chirp
0	31.3
1	30
2	32.
3	30.
4	30.
...	
1477	31.
1478	31.
1479	31.
1480	31.
1481	31.

1482 rows × 50



luminosity_distance
412.066
412.066
412.066
412.066
412.066
...
412.066
412.066
412.066
412.066
412.066

PE w/ Bilby - results

```
result_short.plot_corner(parameters=["chirp_mass", "mass_ratio",
"geocent_time", "phase"], prior=True)
```

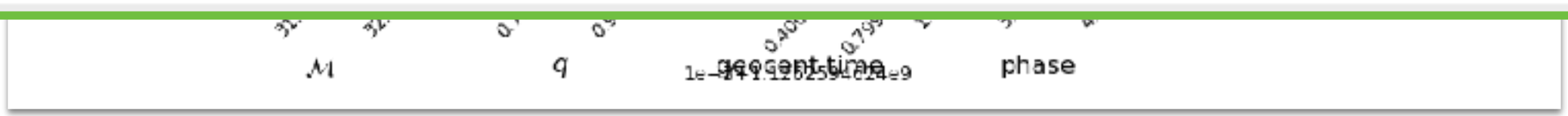
Bilby example

이형원교수님
여름학교 강의중에서

Example Gaussian

```
#!/usr/bin/env python3
"""
An example of how to use bilby to perform parameter estimation for
non-gravitational wave data consisting of a Gaussian with a mean and variance
"""
import bilby
import numpy as np

# A few simple setup steps
label = 'gaussian_example'
outdir = 'outdir'

# Here is minimum requirement for a Likelihood class to run with bilby. In this
# case, we setup a GaussianLikelihood, which needs to have a log_likelihood
# method. Note, in this case we will NOT make use of the `bilby`
# waveform_generator to make the signal.

# Making simulated data: in this case, we consider just a Gaussian

data = np.random.normal(3, 4, 100)


class SimpleGaussianLikelihood(bilby.Likelihood):
    def __init__(self, data):
        """
        A very simple Gaussian likelihood

        Parameters
        ----------
        data: array_like
            The data to analyse
        """
        super().__init__(parameters={'mu': None, 'sigma': None})
        self.data = data
        self.N = len(data)

    def log_likelihood(self):
        mu = self.parameters['mu']
        sigma = self.parameters['sigma']
        res = self.data - mu
        return -0.5 * (np.sum((res / sigma)**2) +
                       self.N * np.log(2 * np.pi * sigma**2))


likelihood = SimpleGaussianLikelihood(data)
priors = dict(mu=bilby.core.prior.Uniform(0, 5, 'mu'),
              sigma=bilby.core.prior.Uniform(0, 10, 'sigma'))

# And run sampler
result = bilby.run_sampler(
    likelihood=likelihood, priors=priors, sampler='dnest4', npoints=50000,
    walks=100, outdir=outdir, label=label)
result.plot_corner()
```

- Data : $N(3,4)$



PE w/ Bilby - BNS

```
priors = bilby.gw.prior.BNSPriorDict()
for key in ['psi', 'geocent_time', 'ra', 'dec', 'chi_1', 'chi_2',
            'theta_jn', 'luminosity_distance', 'phase']:
    priors[key] = injection_parameters[key]
priors.pop('mass_1')
priors.pop('mass_2')
priors.pop('lambda_1')
priors.pop('lambda_2')
priors.pop('mass_ratio')
priors['chirp_mass'] = bilby.core.prior.Gaussian(1.215, 0.1, name='chirp_mass',
unit='$M_{\\odot}$')
priors['symmetric_mass_ratio'] = bilby.core.prior.Uniform(0.1, 0.25,
name='symmetric_mass_ratio')
priors['eos_spectral_gamma_0'] = bilby.core.prior.Uniform(0.2, 2.0, name='gamma0',
latex_label='$\\gamma_0')
priors['eos_spectral_gamma_1'] = bilby.core.prior.Uniform(-1.6, 1.7, name='gamma1',
latex_label='$\\gamma_1')
priors['eos_spectral_gamma_2'] = bilby.core.prior.Uniform(-0.6, 0.6, name='gamma2',
latex_label='$\\gamma_2')
priors['eos_spectral_gamma_3'] = bilby.core.prior.Uniform(-0.02, 0.02,
name='gamma3', latex_label=‘$\\gamma_3’)

priors['eos_check'] = bilby.gw.prior.EOSCheck()
```

PE w/ Bilby - BNS

```
# Initialise the likelihood by passing in the interferometer data (IFOs)
# and the waveform generator
likelihood = bilby.gw.GravitationalWaveTransient(
    interferometers=interferometers, waveform_generator=waveform_generator,
    time_marginalization=False, phase_marginalization=False,
    distance_marginalization=False, priors=priors)

# Run sampler.  In this case we're going to use the `dynesty` sampler
result = bilby.run_sampler(
    likelihood=likelihood, priors=priors, sampler='dynesty', npoints=1000,
    injection_parameters=injection_parameters, outdir=outdir, label=label,
    conversion_function=bilby.gw.conversion.generate_all_bns_parameters,
    resume=True)
```

Jump to Tutorials w/ Colab.

1. https://colab.research.google.com/
2. Search “gw-odw/odw-2021” in GitHub Tab.
   - https://github.com/gw-odw/odw-2021
3. An example of GW150914

★ Save a copy in your google drive

- Go ‘file’ tab > ‘Save a copy in Drive’

★ Email me (ymkim715@gmail.com, or ymkim715@unist.ac.kr) if you have a question after the summer school.

★ Or contact GWOSC team (gwosc@igwn.org)


Thank you for your attention.