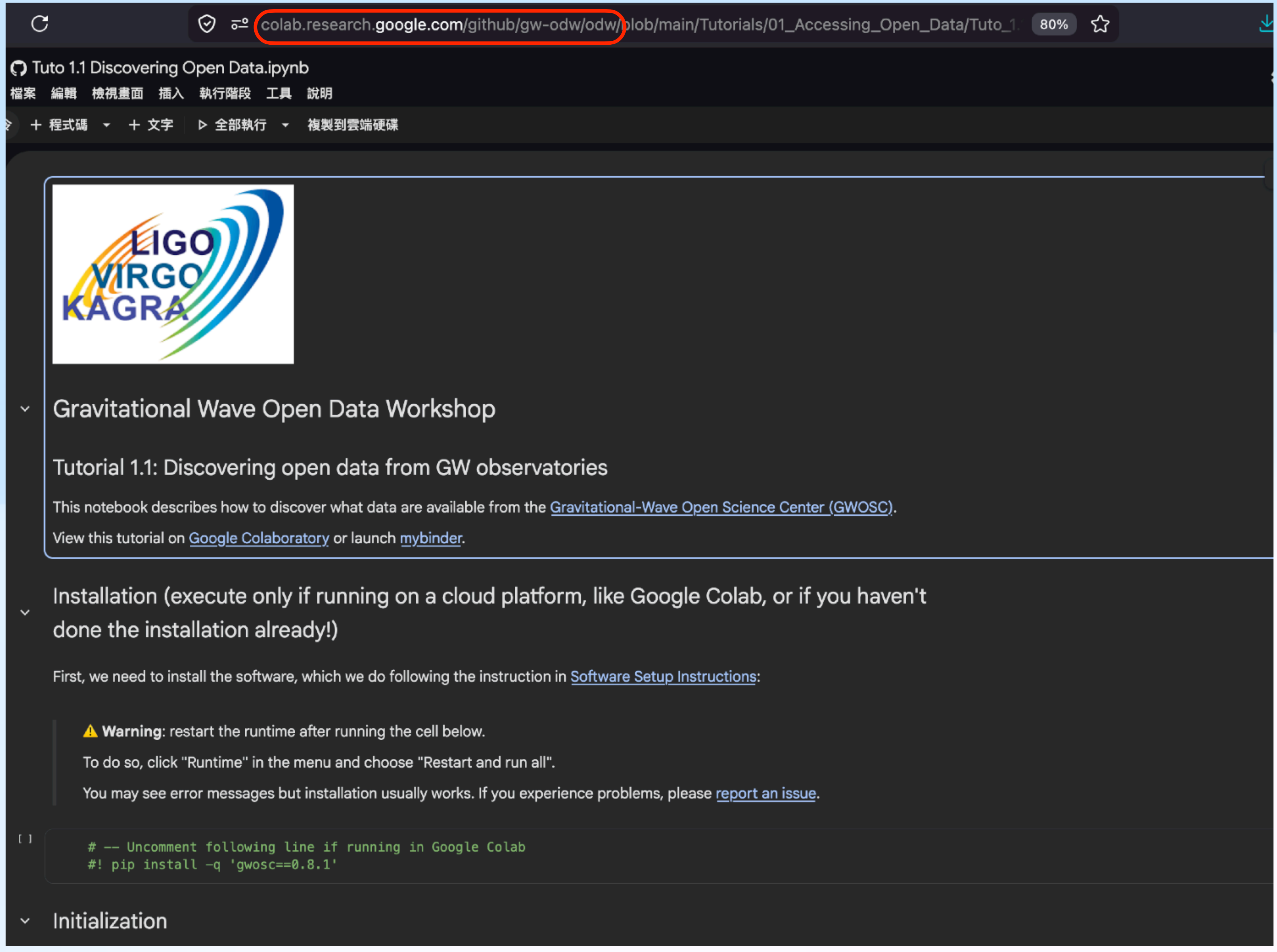
Gravitational Wave Searches for Compact Binary Coalescences (CBC)

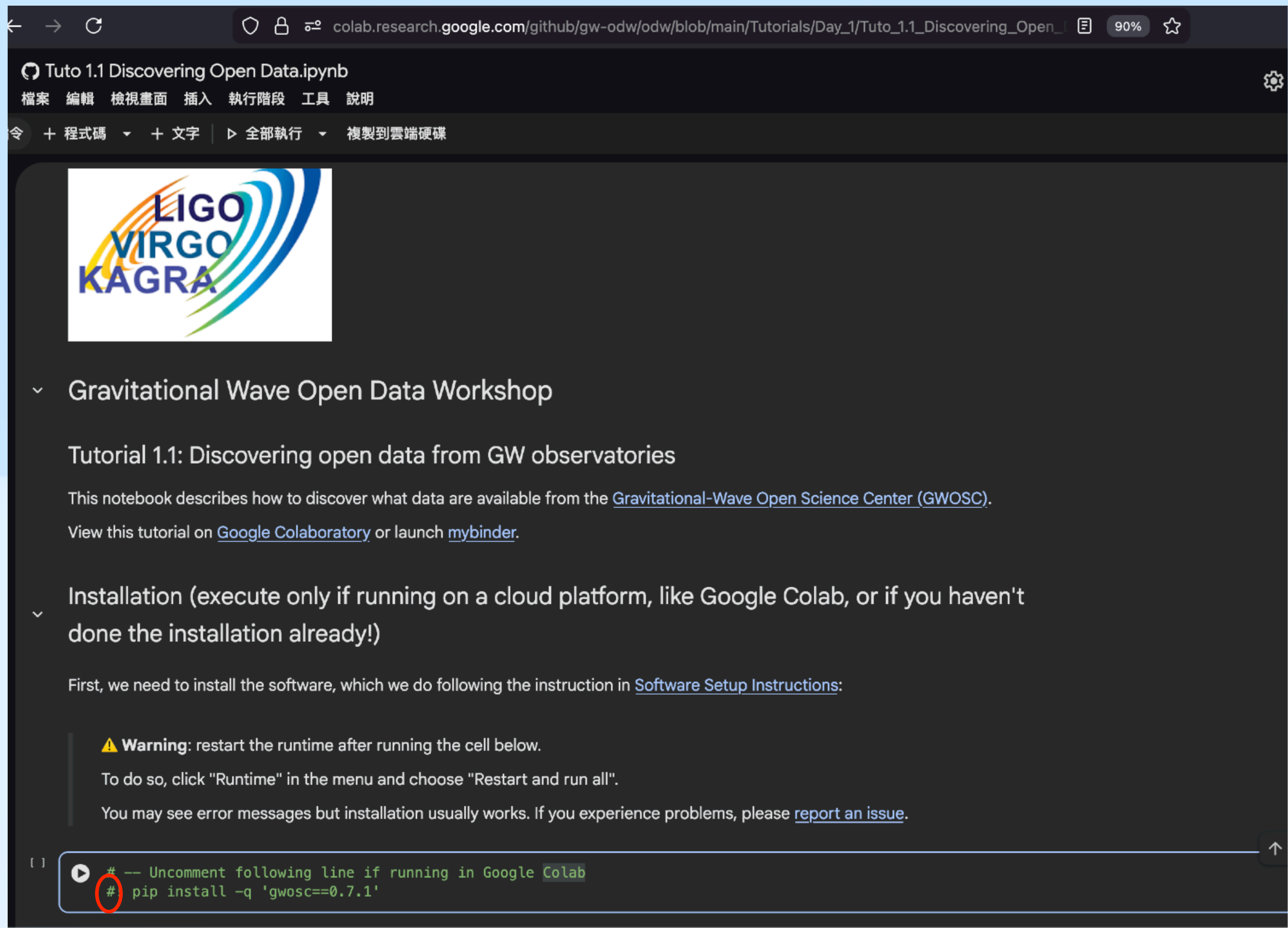
Lupin C. C. Lin of National Cheng Kung University

@Study hub of 2026 GWOSC GW open data workshop

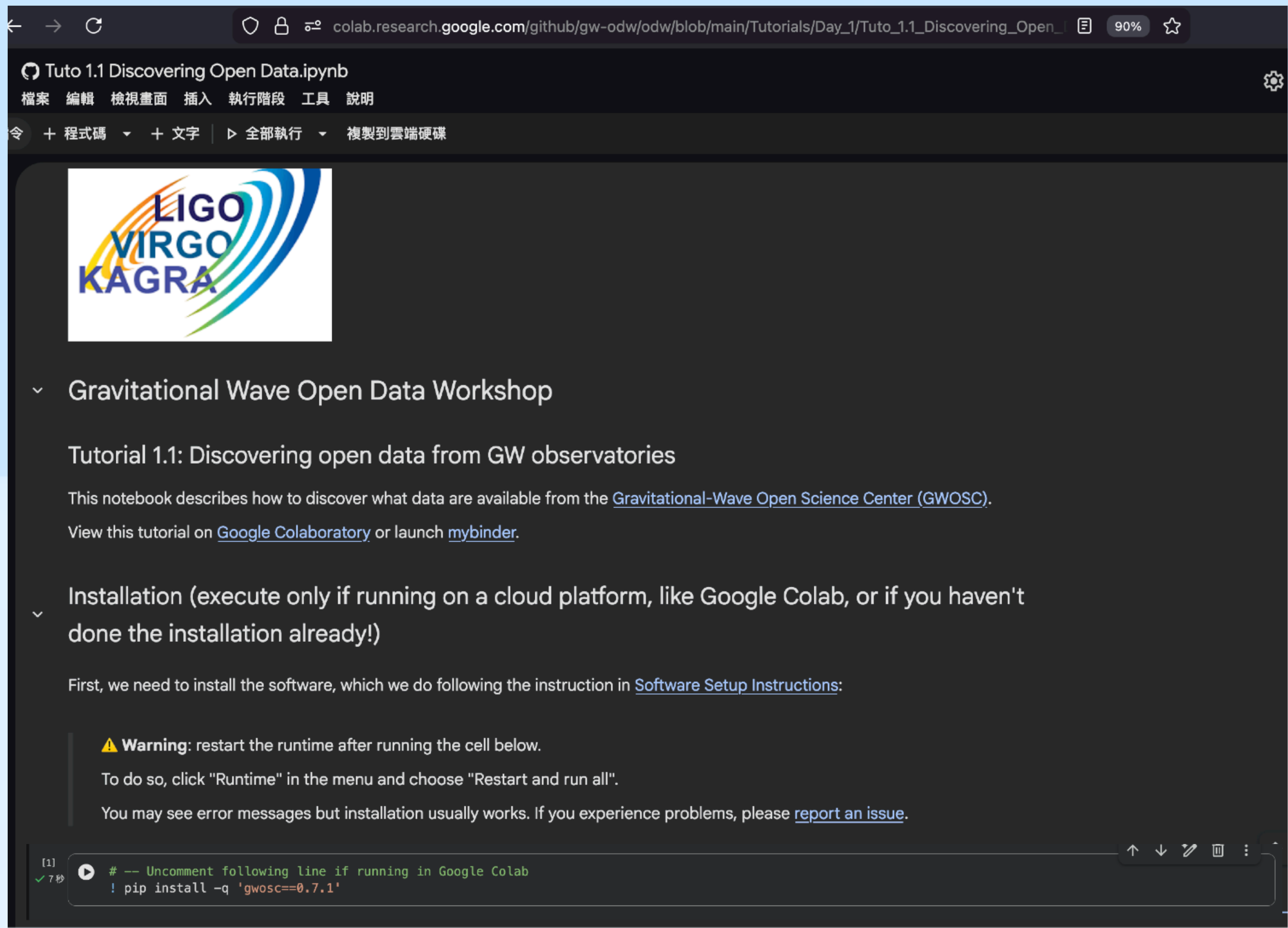
Tutorial Resources



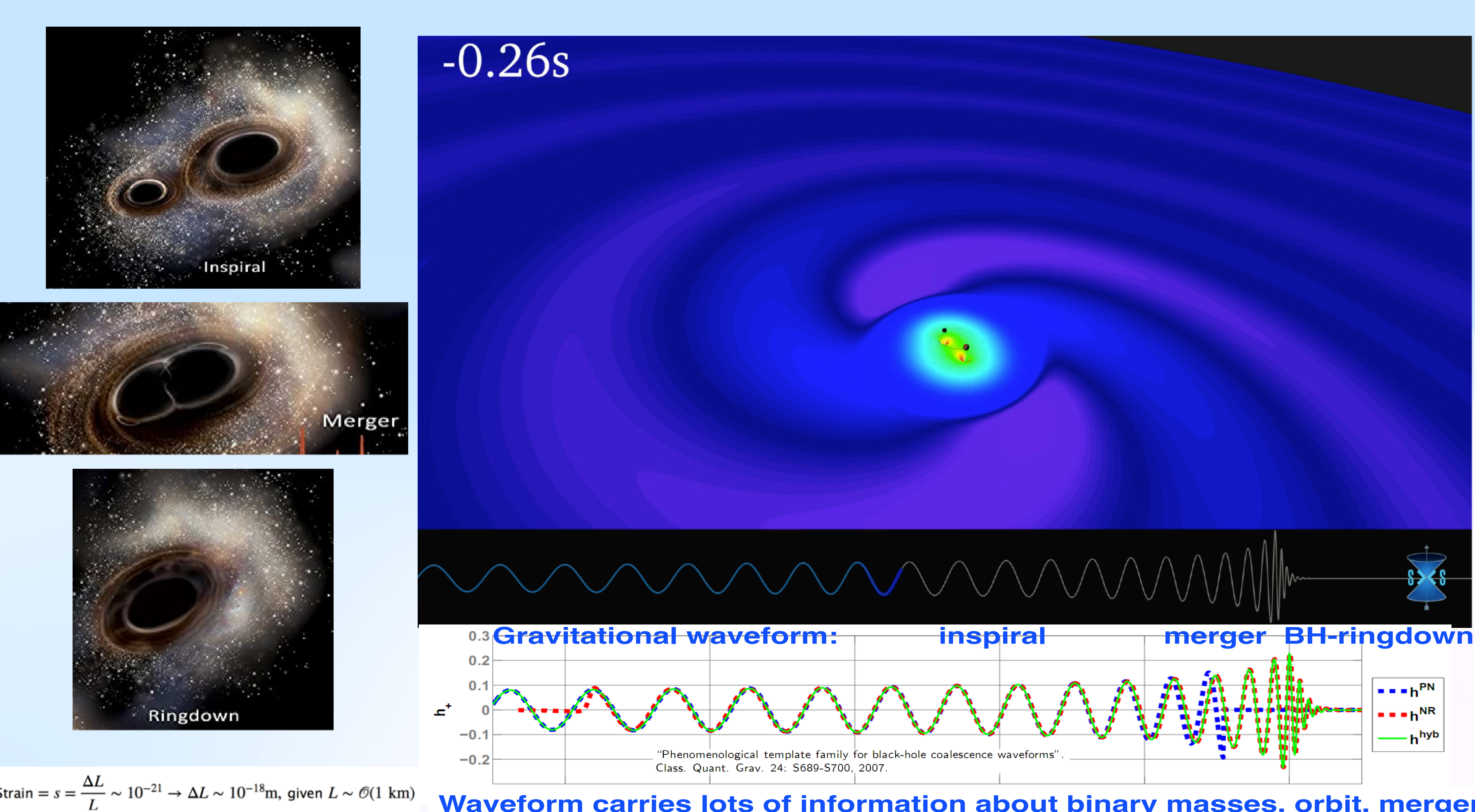
Tutorial Resources



Tutorial Resources

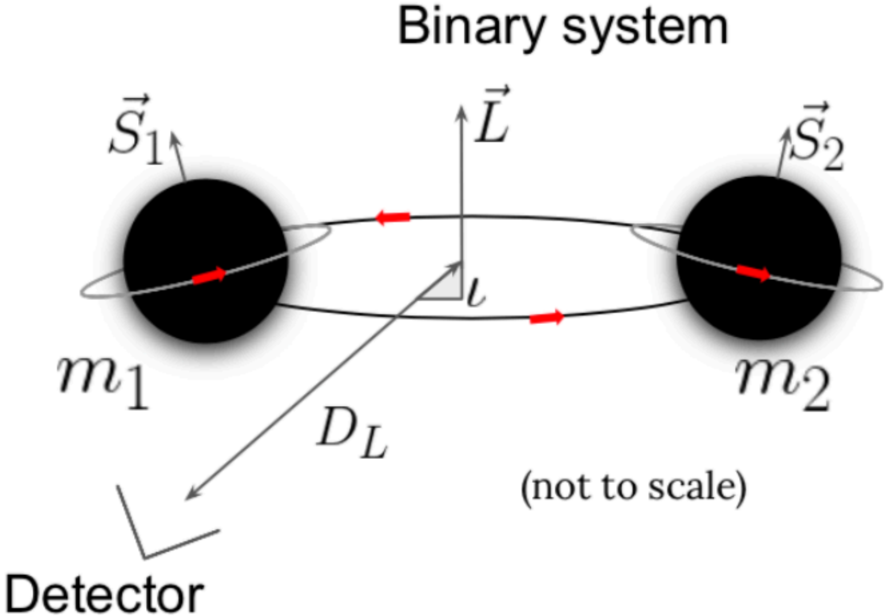


To search for GW signals from compact binary coalescence



Modelling Compact Binary Mergers

The signal from a binary system made up of black holes is described by 15 parameters



(8) **Intrinsic parameters**

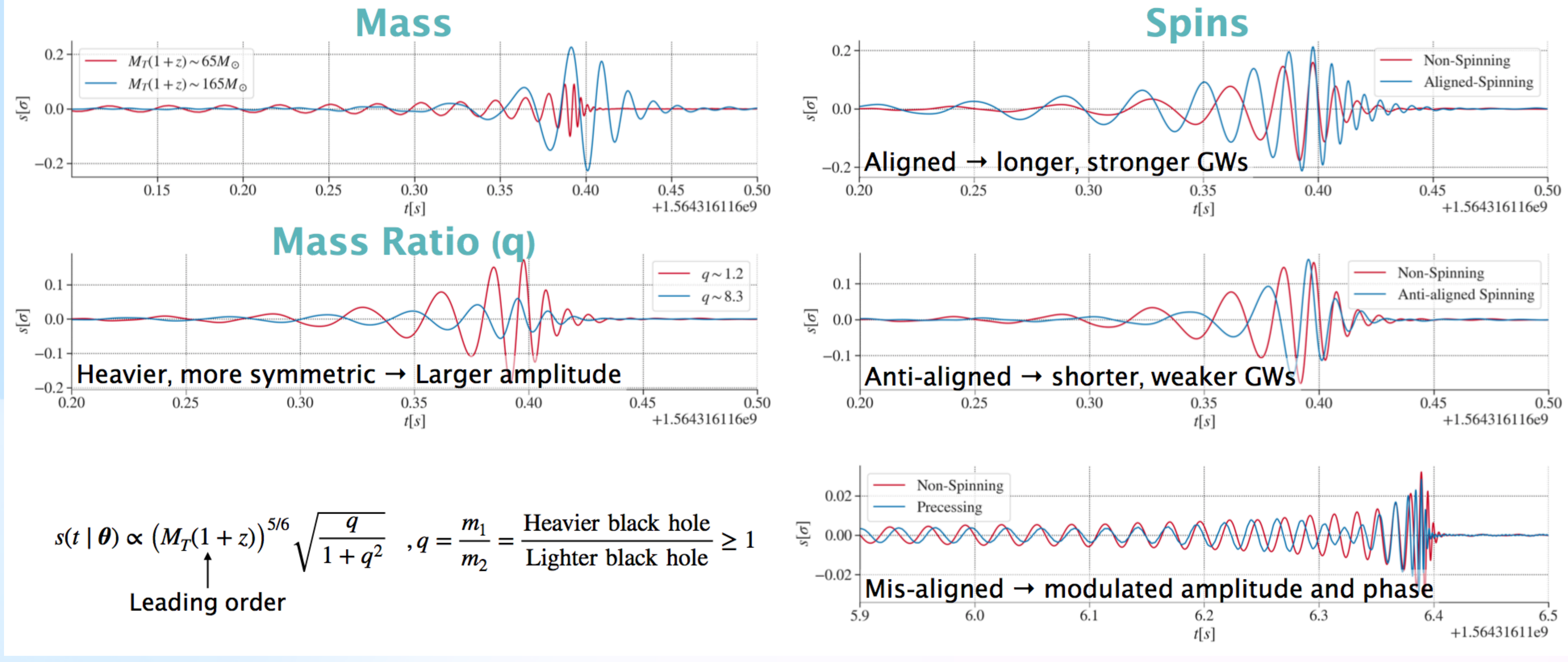
- Two component masses: m_1, m_2
- Six spin Components: χ_1, χ_2

(7) **Extrinsic parameters**

- Sky Location: (α, δ)
- Luminosity distance: D_L (or equivalently the redshift z)
- Binary orientation parameters: (ι, φ)
- Polarisation angle: ψ
- Merger time: t_c

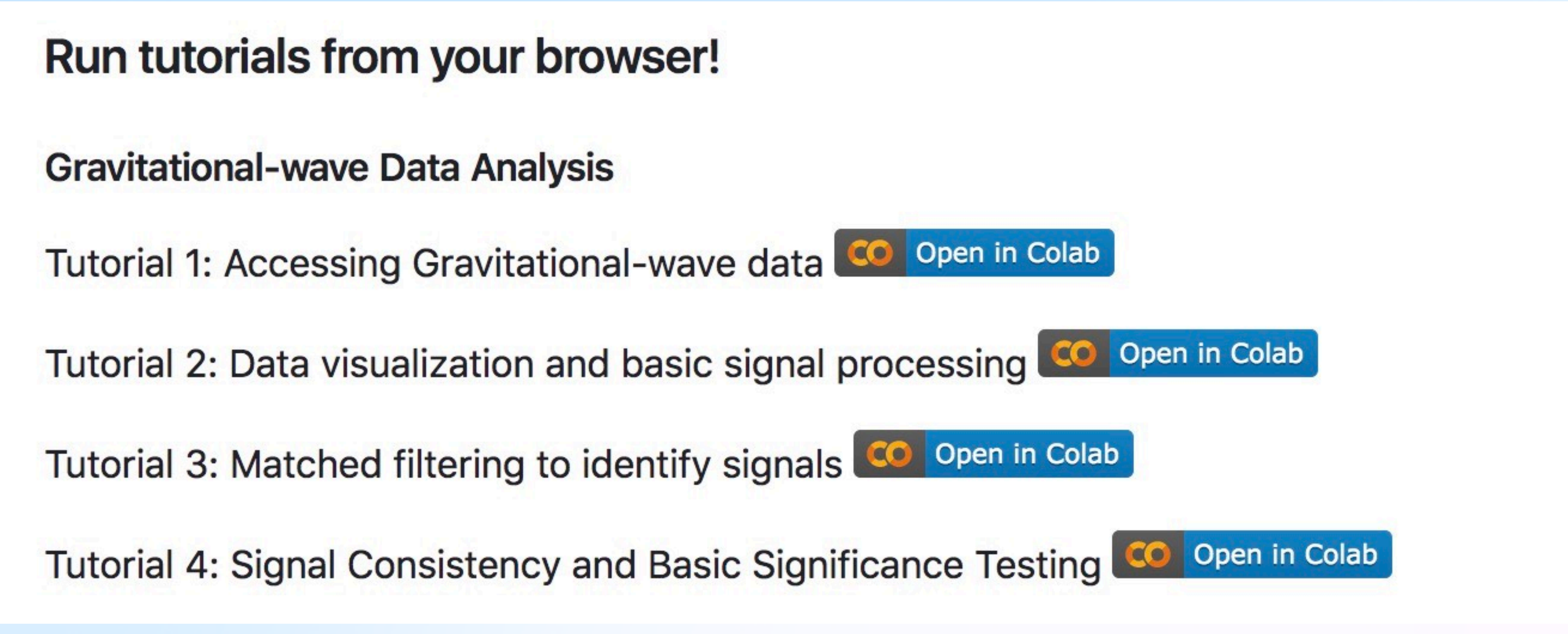
More parameters required if matter or new physics is included

Phenomenology of Black Hole Binaries



To generate the GW waveform with PyCBC

PyCBC (Python-based toolkit for Compact Binary Coalescence) :
PyCBC is the result a set of core libraries and application suites used to study gravitational-wave data and astrophysics. It contains algorithms that can detect CBC and measure the astrophysical parameters of detected sources. PyCBC was used in the first direct detection of gravitational waves (GW150914) by LIGO and is used in the ongoing analysis of LIGO and Virgo data.
PyCBC-Tutorials: https://github.com/gwastro/PyCBC-Tutorials



Reference: https://pycbc.org/pycbc/latest/html/

Approximants of GW waveform in PyCBC

```
from pycbc.waveform import td_approximants, fd_approximants

print(td_approximants())
print(fd_approximants())
```

td: time-domain

fd: frequency-domain

['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', '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', 'SpinTaylorF2', 'TaylorF2NL', 'PreTaylorF2', 'SpinTaylorF2_SWAPPER']

Reference: https://reurl.cc/vk8bKe

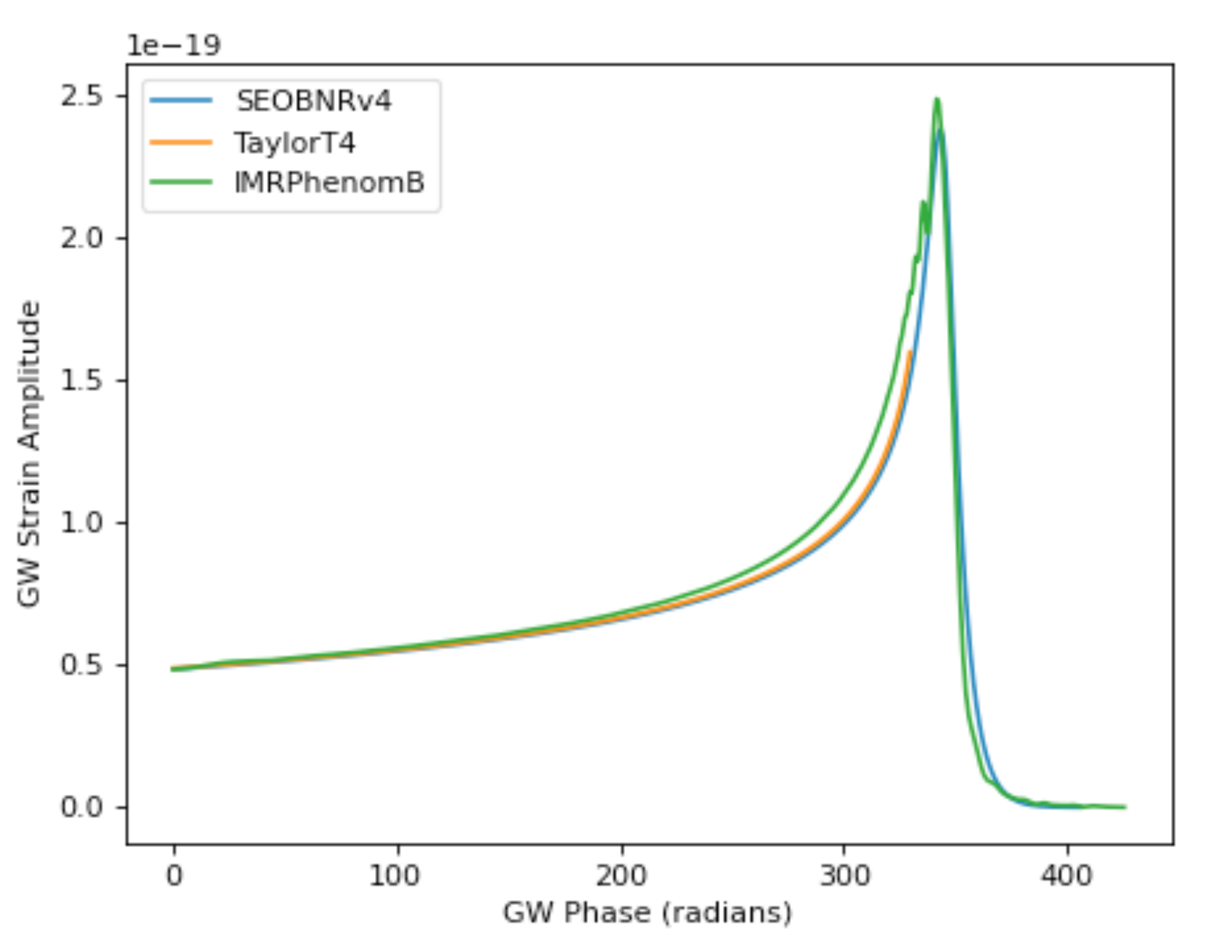
Plotting GW phase and amplitude of TD waveform

```
import matplotlib.pyplot as pp
from pycbc import waveform


for apx in ['SEOBNRv4', 'TaylorT4', 'IMRPhenomB']:
    hp, hc = waveform.get_td_waveform(approximant=apx,
                                      mass1=10,
                                      mass2=10,
                                      delta_t=1.0/4096,
                                      f_lower=40)

    hp, hc = hp.trim_zeros(), hc.trim_zeros()
    amp = waveform.utils.amplitude_from_polarizations(hp, hc)
    phase = waveform.utils.phase_from_polarizations(hp, hc)

    pp.plot(phase, amp, label=apx)
```



Approximants to obtain CBC waveform

1. **Taylor** : The two-body dynamics in general relativity has been solved using the post-Newtonian (PN) approximation. The evolution of the orbital phase and the emitted gravitational radiation are now known to a rather high order corresponding to the characteristic velocity of the binary. (https://arxiv.org/abs/0907.0700)

*n*PN approximation refers to the terms of fractional order $(v/c)^{2n} \sim ((GM/c^2r))^n$ in the equations of motion.

In the adiabatic approximation, the theory allows the phasing to be specified by a pair of differential equations: $\dot{\phi}(t)=v^3/M$ and $\dot{v}(t)=-F(v)/E'(v)$

$\phi(t)$: orbital phase, M: total mass of binary, F: gravitational wave luminosity, E': the derivative of the binding energy with respect to v.

Different Taylor families:

T1: the choice to leave the PN expansions of the luminosity F(v) and E'(v).

T2: expanding the ratio of the polynomials F (v)/E'(v) in these equations to consistent PN order and integrating them one obtains a pair of parametric equations for φ(v) and t(v). (TaylorT2 is the most computationally expensive.)

T3: the phasing could be written as an explicit function of time φ(t).

T4: to expand the rational polynomial F (v)/E'(v) in v to the consistent PN order. (It is a straightforward extension of TaylorT1, and at 3.5PN order by coincidence is found to be in better agreement with numerical simulations of the inspiral phase.)

F2: The analogue of the TaylorT2 in the frequency domain follows by explicitly truncating the energy and flux functions to consistent post-Newtonian orders, and this leads us to a Fourier domain waveform. (The most often employed PN-approximant.)

EccentricTD: Based on Taylor T4 approximant and including orbital eccentricity effects.

Approximants of GW waveform in PyCBC

2. NR (Numerical Relativity): To solve the Einstein equations for two body problem with Numerical Relativity simulations. Highly computational cost is required to generate NR waveforms. (https://arxiv.org/abs/1703.01076) (https://arxiv.org/abs/1705.07089)

Different NR families:

NR_hdf5: to generate waveform with NR simulation provided in a HDF file.
NRSur7dq2: to generate waveform of non-eccentric BBH systems with NR surrogate (Sur) model including all 7 dimensions of the parameter space.

3. EOB (Effective One-Body model): For a simulated waveform, the PN inspiral information is re-summed and calibrated to NR, and the merger-ringdown part is obtained from a phenomenological fit to NR result. (https://arxiv.org/abs/1106.1021) (https://arxiv.org/abs/2004.09442)

Prefix and Suffix in different EOB families:

EOBNRv2: A EOB waveform that incorporates information from several non-spinning BBH simulations, with black hole ring-down quasi-normal modes attached to provide a complete BBH waveform. (The EOBNRv2 waveform is believed to be sufficiently accurate to search for non-spinning BBH signals in the aLIGO sensitive band (10-1000 Hz).
SEOBNRv2_opt: Optimized (opt) Spin-aligned (S) EOBNR model v2.
SEOBNRv3_opt_NK4: To use Runge–Kutta 4th-order method (RK4) Optimized Spin (S) precessing EOBNR model v3 to generate waveform.
SEOBNRv4PHM: The first multipolar precessing (P) waveform model to take into account high-order modes (HM) in the co-precessing frame in the EOB formalism for the entire coalescence stage of binary black holes.

Approximants of GW waveform in PyCBC

4. **IMRPhenom** (Inspiral–Merger–Ringdown Phenomenological waveform model): To generate GW waveform by tuning an extended Post-Newtonian (PN) inspiral expansion and separating analytical functions for the merger and ring-down waveforms to NR simulations, and then to smoothly combine the waveforms in these three regions.

Prefix and Suffix in different IMRPhenom families: (https://arxiv.org/abs/1708.00404) (https://arxiv.org/abs/1905.06011)

IMRPhenomA: only for non-spinning binaries

IMRPhenomB/C/D: spinning, but no precessing (A,B,C should not ne used unless specific reasons)

IMRPhenomP: precessing (P) binaries, based on model C.

IMRPhenomPV2: precessing (P) binaries, based on model D.

IMRPhenomHM: spinning, non-precessing, based on model D and work with higher-multipole (HM) model.

IMRPhenomPV3HM: precessing (P) with higher-multipole (HM) model, based on HM.

IMRPhenomX: updated models for spin-aligned binaries with/without (XHM/XAS) higher-multipole modes and for precessing with/without (XPHM/XP) higher-multipole modes.

IMRPhenomPV2_NRTidal: precessing, and NR-tuned tidal effect (Tidal) to model D phasing.

IMRPhenomNSBH: single spin, non-precessing NS-BH binaries, based on counting amplitude from model C and counting phase from D_NRTidalV2.

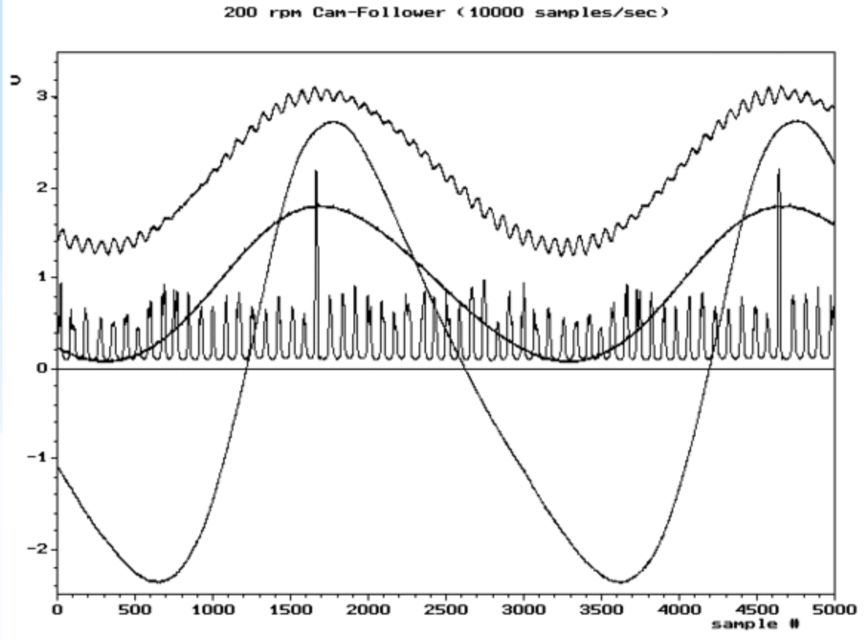
5. **HGimri** (Huerta's and Gair's intermediate-mass-ratio inspirals model): The GW waveform for the circular and equatorial inspirals of a neutron star or a stellar-mass black hole falls into an intermediate-mass black hole. (https://arxiv.org/abs/1009.1985)

To generate the GW waveform with LALSuite

LALSuite (LIGO scientific collaboration Algorithm Library Suite) :
LALsuite contains numerous GW analysis libraries primarily written in C and generally supported for Python packages.
It includes:

- FFT, statistics and time-domain filtering
- Numerical and signal injection routines
- Gravitational waveform and noise generation
  - Burst GW data analysis
  - Inspiral and ringdown CBC GW data analysis
  - Pulsar and CW data analysis
  - Bayesian inference data analysis

Reference: https://lscsoft.docs.ligo.org/lalsuite/lalsuite/index.html



Nyquist frequency: In signal processing, it is a characteristic of a sampler. The Nyquist frequency (f_n) is the frequency whose cycle-length is twice the interval between samples.

$$f_n = \frac{1}{2} f_s$$

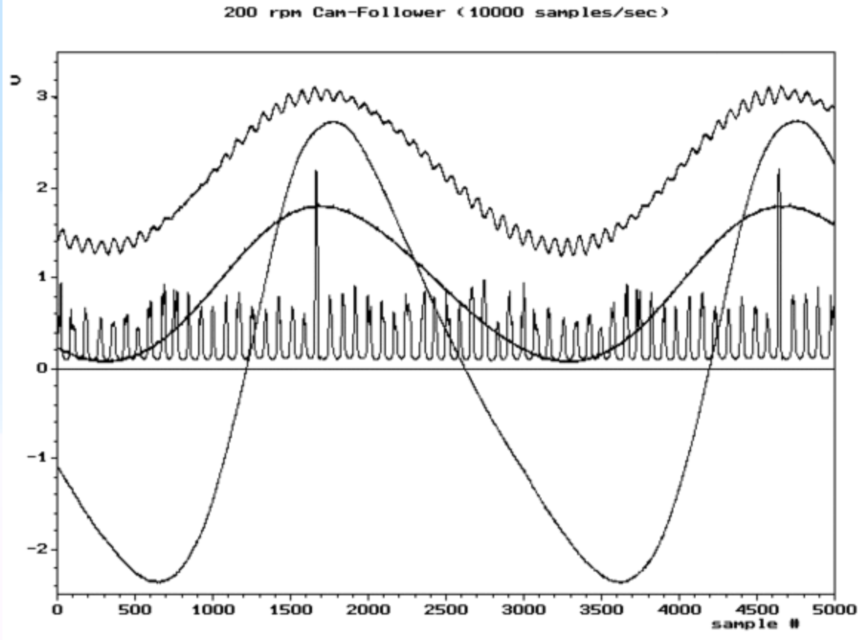
When the highest frequency (bandwidth) of a signal is less than the Nyquist frequency of the sampler, **the resulting discrete-time sequence is said to be free of the distortion known as aliasing.**

To generate the GW waveform with LALSuite

LALSuite (LIGO scientific collaboration Algorithm Library Suite) :
LALsuite contains numerous GW analysis libraries primarily written in C and generally supported for Python packages.
It includes:

- FFT, statistics and time-domain filtering
- Numerical and signal injection routines
- Gravitational waveform and noise generation
  - Burst GW data analysis
  - Inspiral and ringdown CBC GW data analysis
  - Pulsar and CW data analysis
  - Bayesian inference data analysis

Reference: https://lscsoft.docs.ligo.org/lalsuite/lalsuite/index.html



Nyquist frequency: In signal processing, it is a characteristic of a sampler. The Nyquist frequency (f_n) is the frequency whose cycle-length is twice the interval between samples.

$$f_n = \frac{1}{2} f_s$$

When the highest frequency (bandwidth) of a signal is less than the Nyquist frequency of the sampler, **the resulting discrete-time sequence is said to be free of the distortion known as aliasing.**

GW Detector Data

GW interferometers record data as a **discretely sampled time series**

$d = \{d(t_1), \ldots, d(t_n)\}$ at sampling frequency $f_s = 16$ kHz

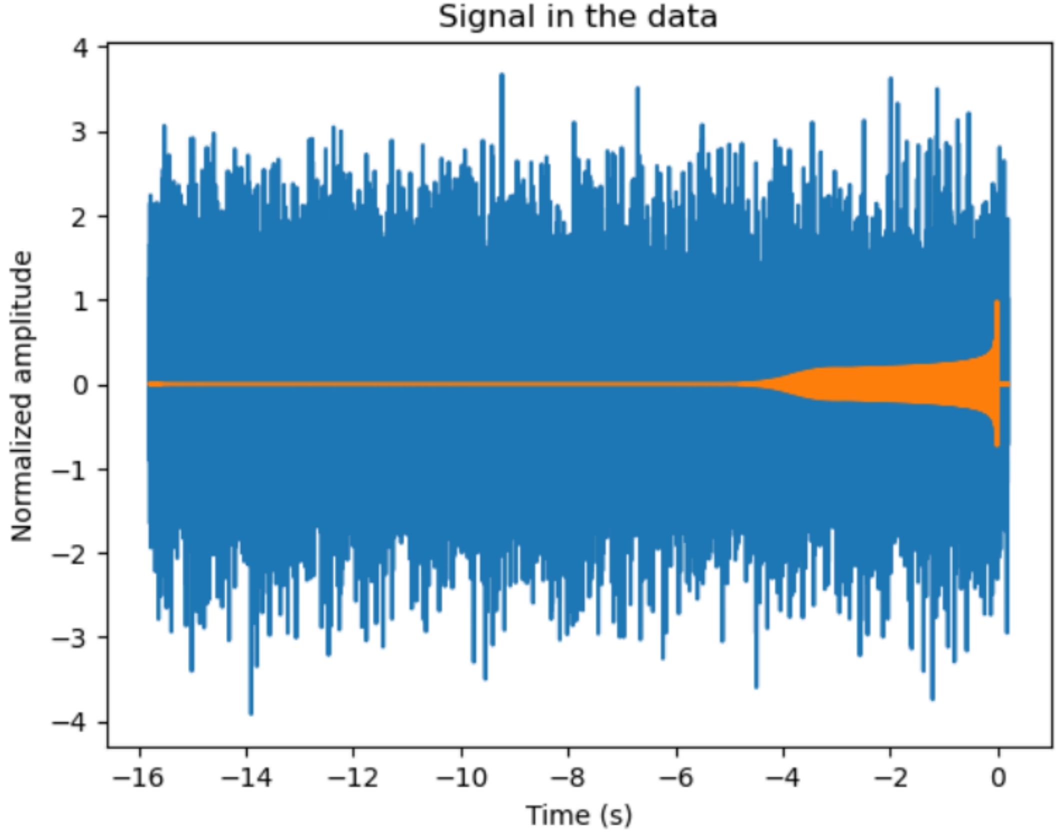
$\rightarrow N_{\text{samples}} = T \times f_s$, where T = data segment duration

$$\boldsymbol{d} = \boldsymbol{s}\left(\boldsymbol{\theta}\right) + \boldsymbol{n}$$

Strain data

Signal
Deterministic for CBC

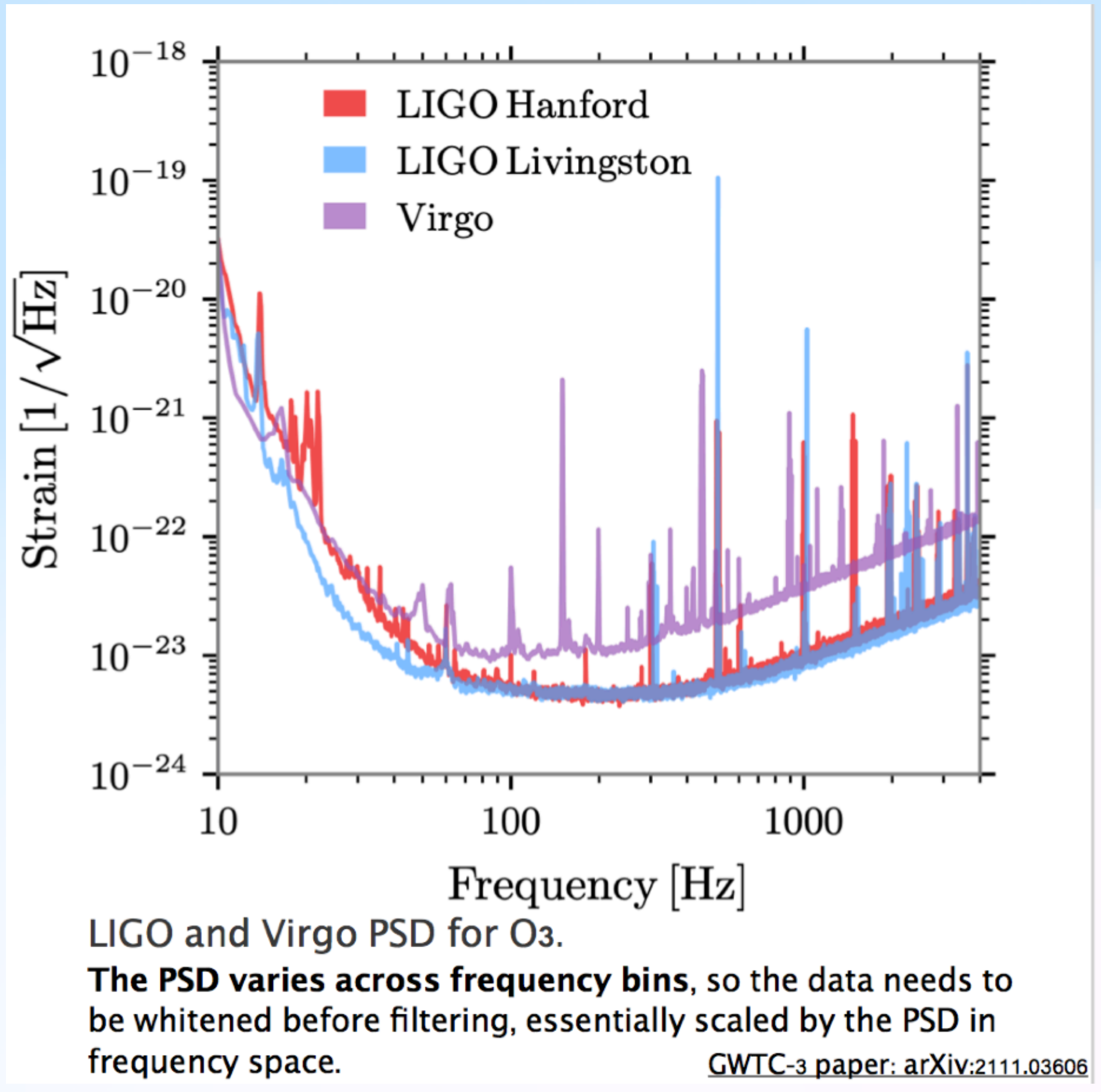
Noise
Stochastic



$|\boldsymbol{n}| \gg \left|\boldsymbol{s}\left(\boldsymbol{\theta}\right)\right| \rightarrow$ Needle in a haystack

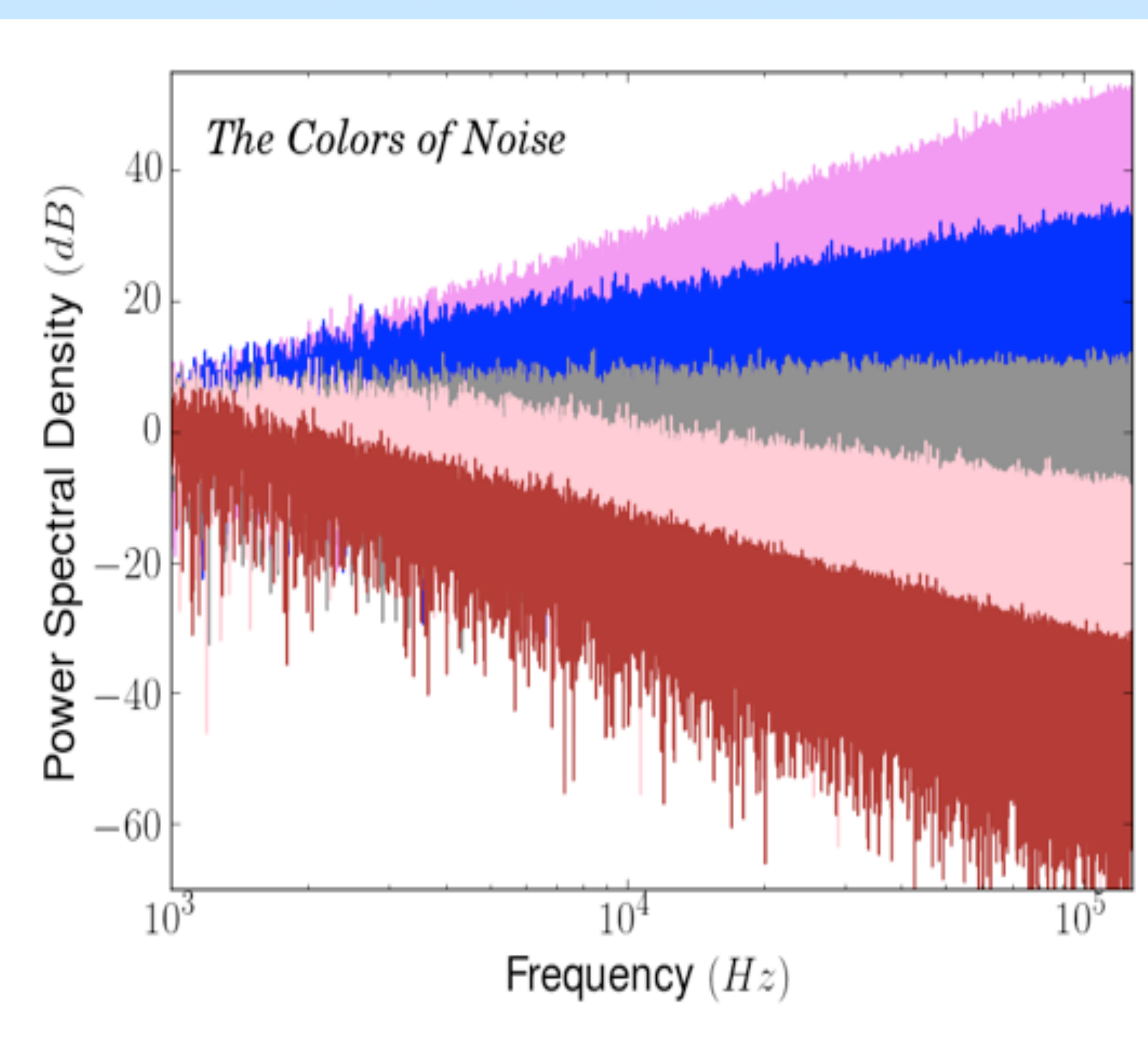
Complicated Noise of the Data

1. Detector noice is not white - not distributed evenly across frequencies.
2. Noise properties can vary over long time scale. (non-stationary)



LIGO and Virgo PSD for O3.
The PSD varies across frequency bins, so the data needs to be whitened before filtering, essentially scaled by the PSD in frequency space. GWTC-3 paper: arXiv:2111.03606

Color noise



The spectrum of white noise has equal power within any equal interval of frequencies and assumed to have a flat power spectrum over the visible range.

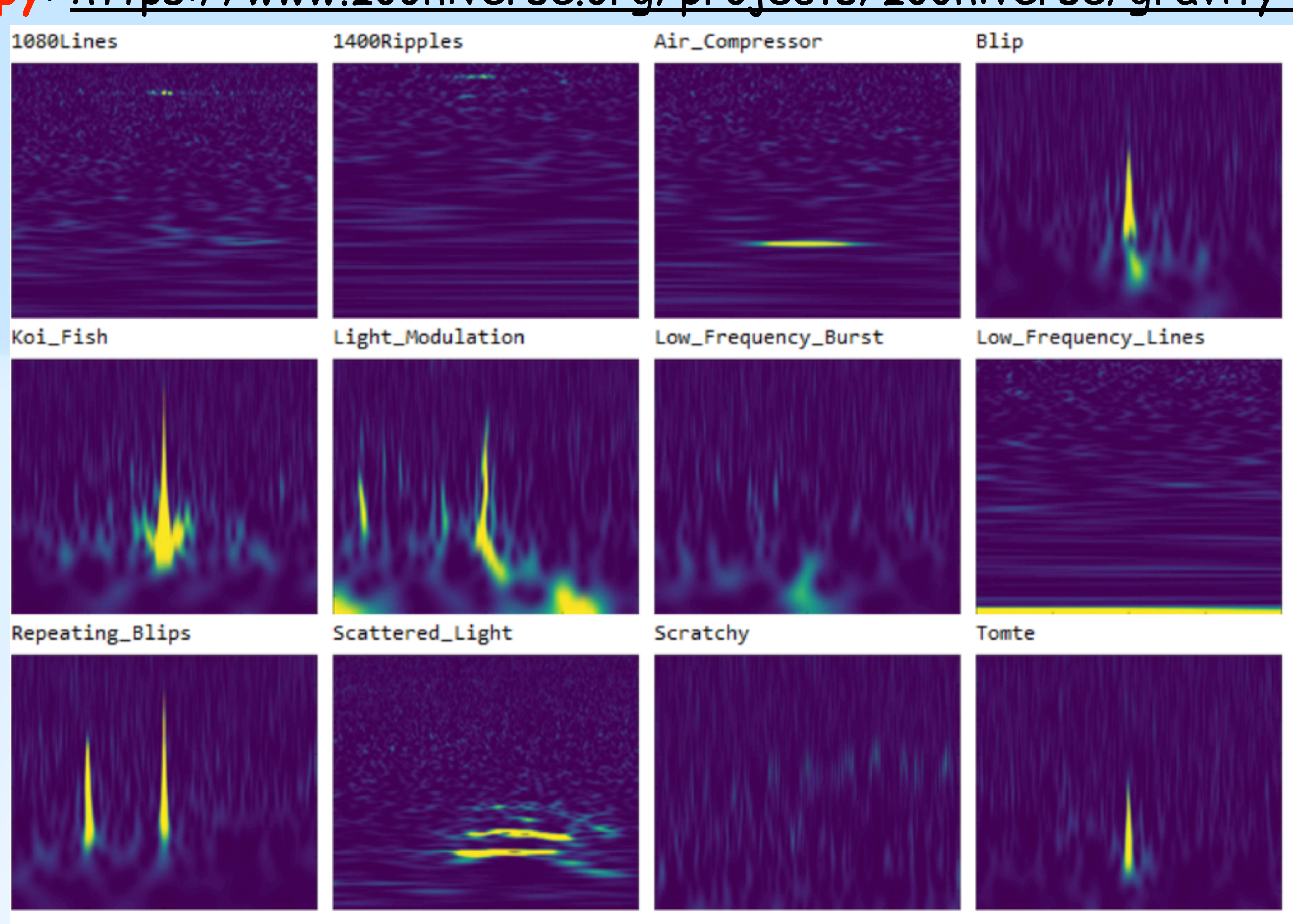
Other color names, such as pink, red, and blue were then given to noise with other spectral profiles, often in reference to the color of light with similar spectra. Many of these definitions assume a signal with components at all frequencies, with a power spectral density per unit of bandwidth proportional to $1/f^{\beta}$ and hence they are examples of **power-law noise**. For instance, the spectral density of white noise is flat ($\beta = 0$), while flicker or pink noise has $\beta = 1$, Brownian/red noise has $\beta = 2$, blue noise has $\beta = -1$, and violent noise has $\beta = -2$, respectively.

Grey noise is random white noise, giving the listener the perception that it is equally loud at all frequencies. This is in contrast to standard white noise which has equal strength over a linear scale of frequencies but is not perceived as being equally loud.

Non-Gaussian Noise

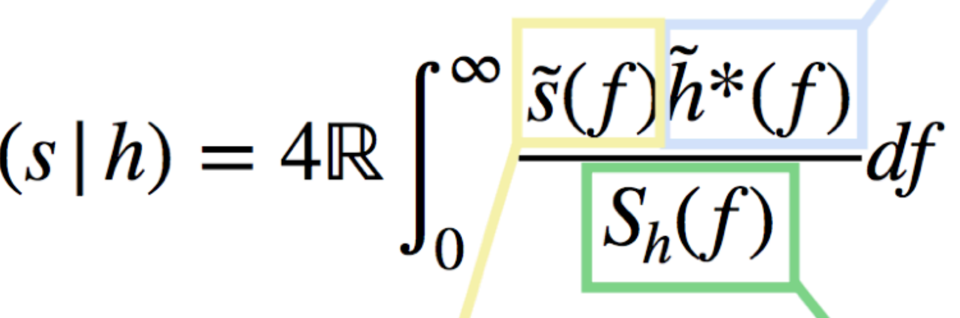
1. Noise properties immediately change over a short time scale. (non-Gaussian)
2. Short duration non-Gaussian artifacts in the data are denoted as glitches.

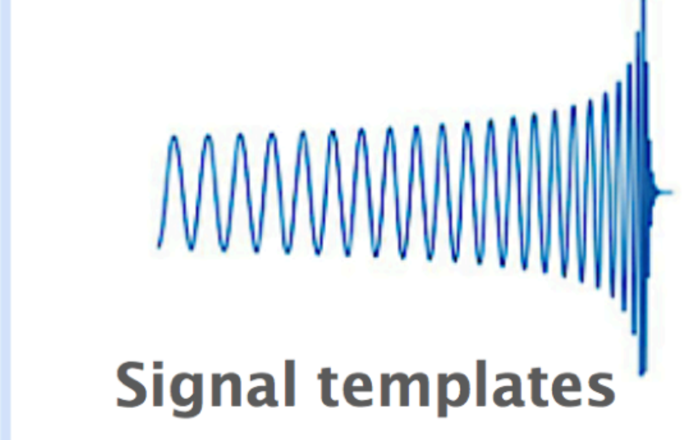
Gravity spy: https://www.zooniverse.org/projects/zooniverse/gravity-spy

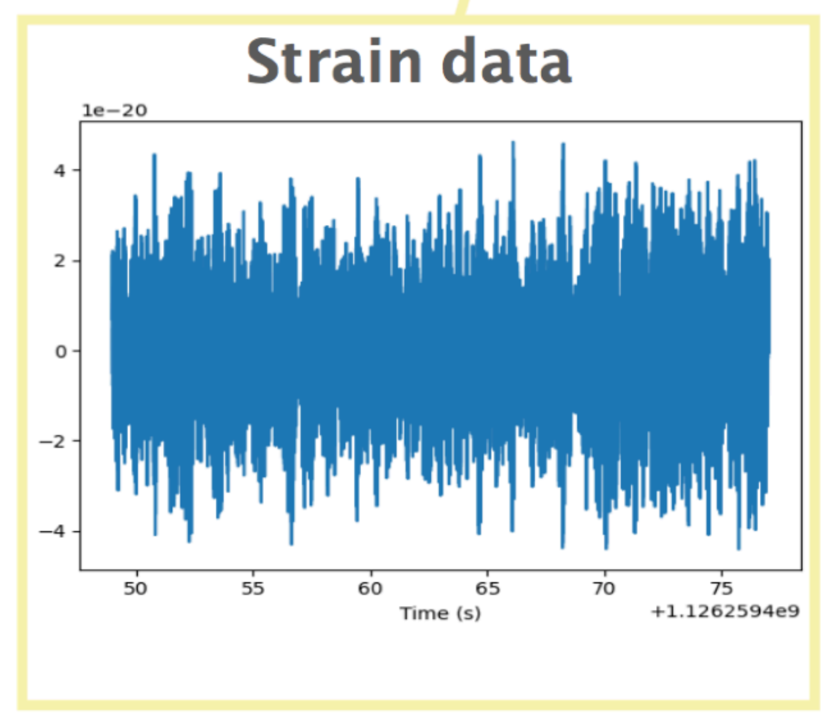


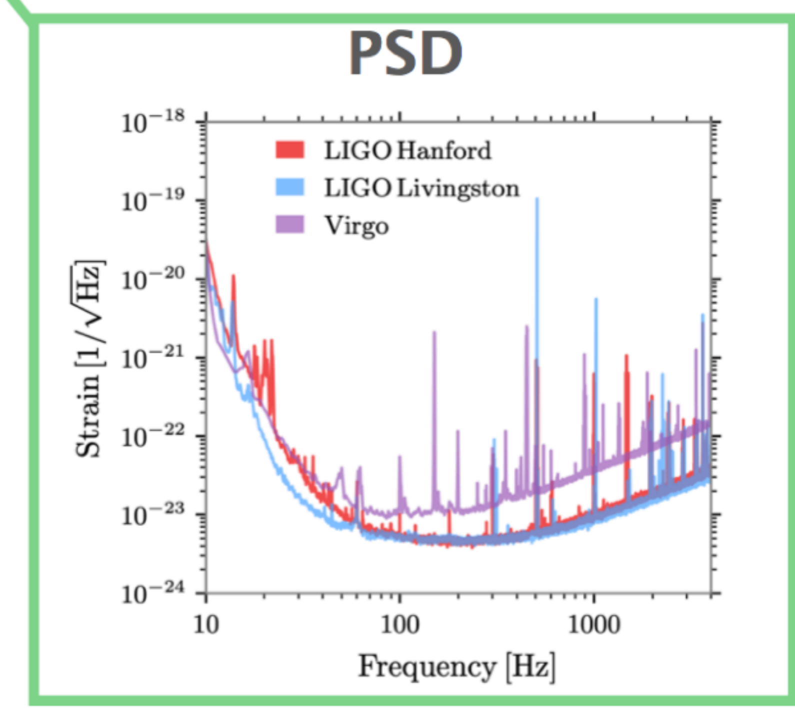
Matched Filtering

If we know the template of the GW waveform, we can simply cross-correlate the data to determine the SNR.









Cross-correlation method

In signal processing, cross-correlation is a measure of similarity of two waveforms as a function of a time lag applied to one of them. The cross-correlation is similar in nature to the convolution of two functions.

$$G(\tau)=(h\star F)(\tau)\overset{\circ}{=}\int_{-\infty}^{\infty}\overline{h(t)}\,F(t+\tau)dt \rightarrow G[i,j]=\sum_{u=-k}^{k}\sum_{v=-k}^{k}h[u,v]F[i+u,j+v]$$ denoted by $G=h\otimes F$

$h[u,v]$ can be treated as the prescription for weights in the linear combination.

In an autocorrelation, there will always be a peak at a lag of zero, and its size will be the signal energy.

```
>>> np.correlate([1, 2, 3], [0, 1, 0.5])
array([3.5])
>>> np.correlate([1, 2, 3], [0, 1, 0.5], "same")
array([2. ,  3.5,  3. ])
>>> np.correlate([1, 2, 3], [0, 1, 0.5], "full")
array([0.5,  2. ,  3.5,  3. ,  0. ])
```

```
        1   2   3
0   1   0.5
```

```
    1   2   3
0   1   0.5
```

```
1   2   3
0   1   0.5
```

```
1   2   3
    0   1   0.5
```

```
1   2   3
        0   1   0.5
```

Matched Filtering

Step 1: Whiten the detector data: $\boldsymbol{d} \rightarrow \frac{\tilde{d}(f_i)}{\sqrt{S_n(f_i)}}$

Whitening normalises the power at all frequencies so that any excess power at any frequency becomes obvious.

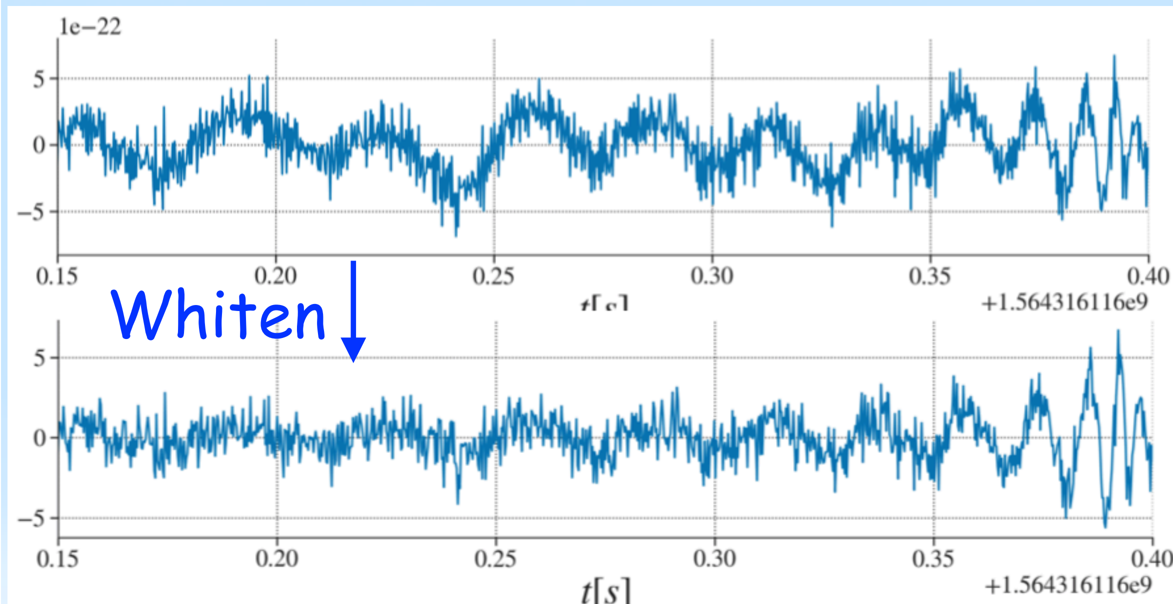
Step 2: Whiten the template: $\boldsymbol{h} \rightarrow \frac{\tilde{h}(f_i \mid \boldsymbol{\theta}')}{\sqrt{S_n(f_i)}}$

Adjust the template's amplitude at each frequency to account for the detector's noise level

Step 3: Calculate the optimal signal-to-noise ratio (SNR) of the template

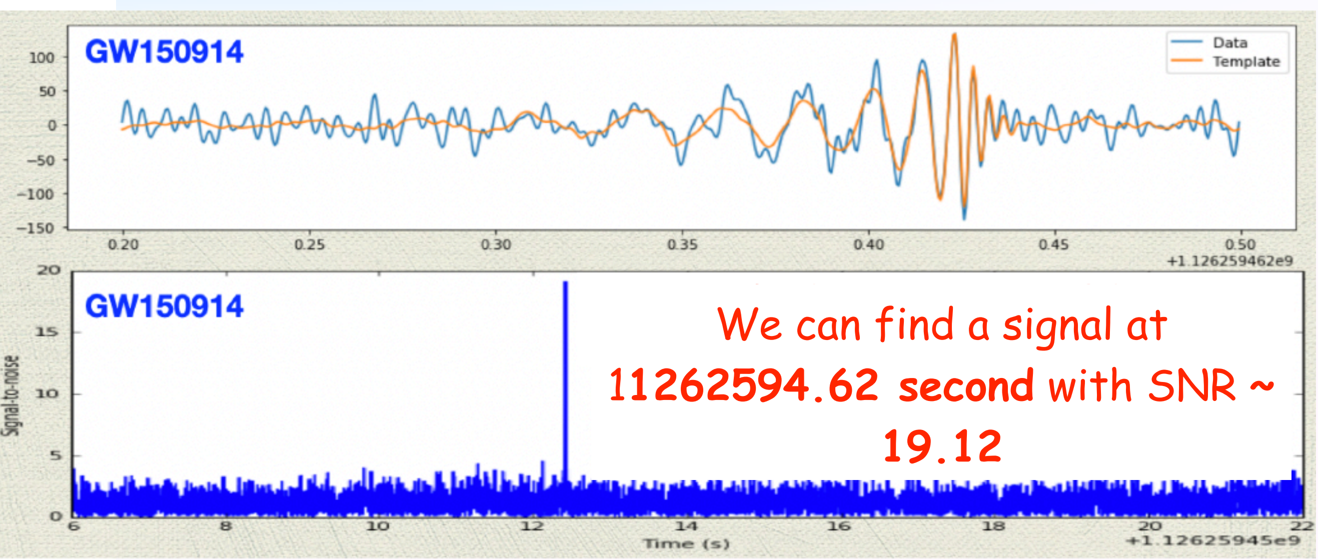
$$\rho^2_{\text{opt}} = (h \mid h) = 4\Re \sum_{f_i} \frac{\tilde{h}^*(f_i \mid \boldsymbol{\theta})\tilde{h}(f_i \mid \boldsymbol{\theta}')}{S_n(f_i)} \Delta f,$$

Δf = frequency resolution



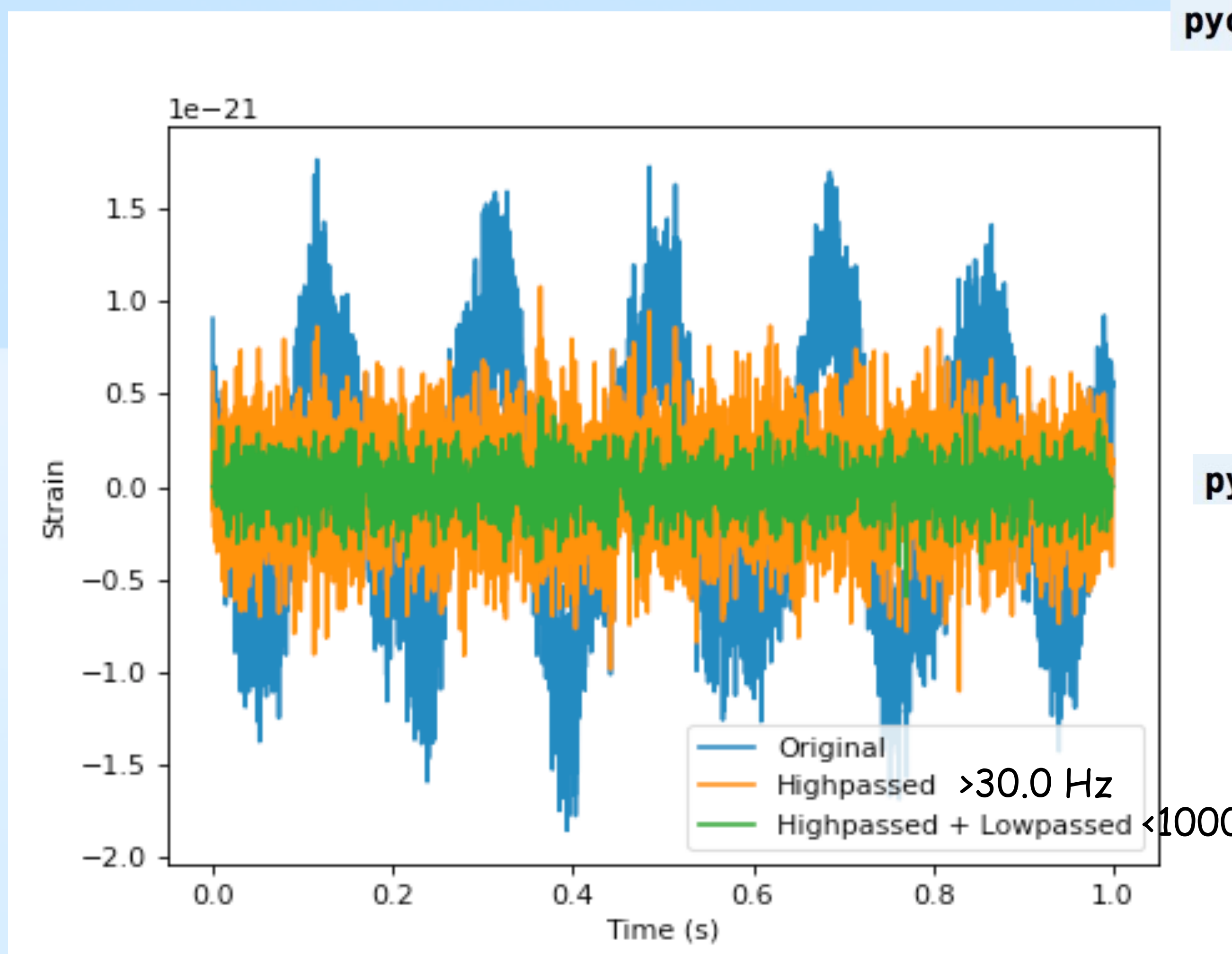
Step 4: Cross correlate the whitened data and whitened normalised template

$$\rho = \frac{(d \mid h)}{\sqrt{(h \mid h)}} \rightarrow \text{matched-filter SNR}$$



Band Pass

A bandpass filter is a device that passes frequencies within a certain range and rejects frequencies outside that range. To compare the data and signal on equal footing, and to concentrate on the frequency range that is important, we whitened both the template and the data, and then bandpass both the data and template between 30-300 Hz.



`pycbc.filter.resample.highpass(timeseries, frequency, filter_order=8, attenuation=0.1)`

Return a new time series that is highpassed above the frequency.

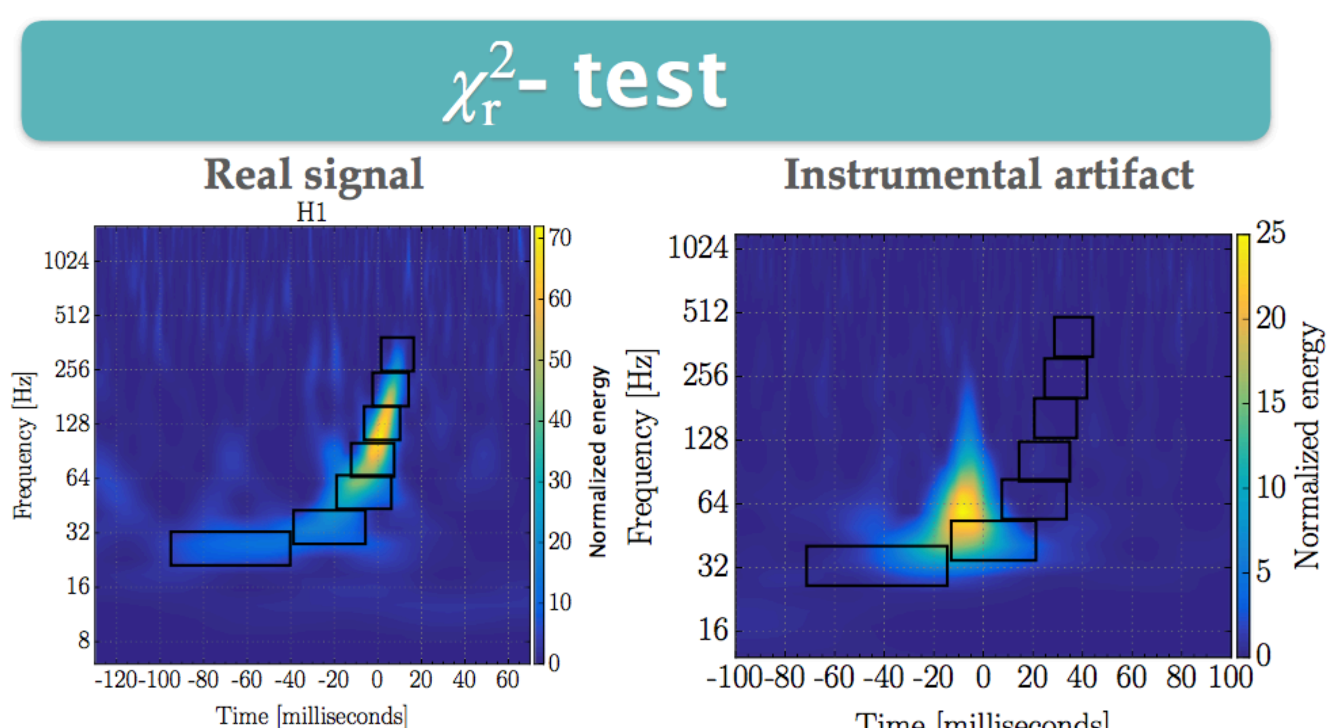
- filter_order ({8, int}, optional) – The order of the filter to use when high-passing the time series.
- attenuation ({0.1, float}, optional) – The attenuation of the filter.

`pycbc.filter.resample.highpass_fir(timeseries, frequency, order, beta=5.0)`

Highpass filter the time series using an FIR filtered generated from the ideal response passed through a kaiser window (beta = 5.0).

- order (int) – Number of corrupted samples on each side of the time series.
- beta (float) – Beta parameter of the kaiser window that sets the side lobe attenuation.

Signal Consistency Tests



- Step-1: Divide the template into p frequency bands of equal expected power.
- Step-2: Calculate $\chi_r^2 = \frac{p}{2p-2} \sum_{l=1}^{p} \left(\rho_l^2 - \frac{\rho^2}{p} \right)^2$
- Trigger consistent with template $\chi_r^2 \rightarrow 1$.
- Use χ_r^2-output to calculate $\varrho = f(\rho, \chi_r^2) \rightarrow$ amended ρ

Allen PRD 71 (2005) 062001 SB, …, IH, SP et al. PRD 87 (2013) 024003

Calculation of the matched -filtered SNR:

(Usman et al., 2016)

$$\chi^2 = p \sum_{i=1}^{p} \left[\left(\frac{\rho_{\cos}^2}{p} - \rho_{\cos,i}^2 \right)^2 + \left(\frac{\rho_{\sin}^2}{p} - \rho_{\sin,i}^2 \right)^2 \right],$$

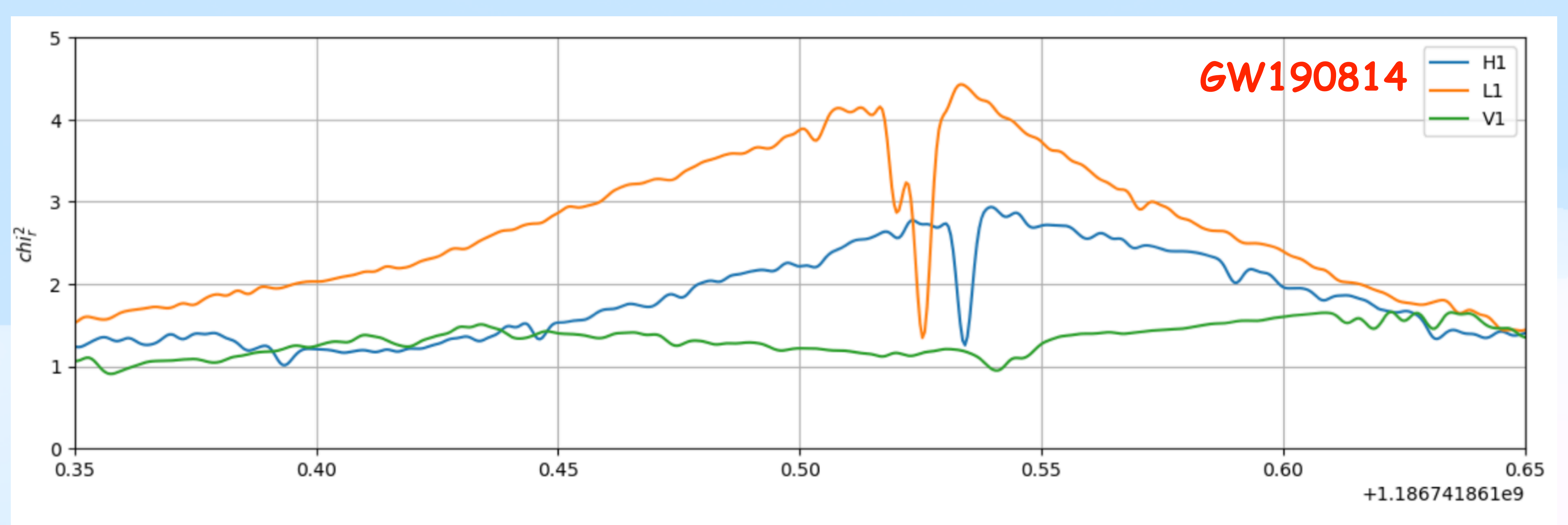
where ρ_{cos} and ρ_{sin} are the SNRs of the two orthogonal phases of the matched filter.

Re-Weighting ranking statistic for SNR:

(Abbott et al., 2020)

$$\tilde{\rho} = \begin{cases} \rho, & \text{if } \chi_r^2 \leq 1, \\ \rho \left[\left(1 + (\chi_r^2)^3 \right) / 2 \right]^{-\frac{1}{6}}, & \text{if } \chi_r^2 > 1. \end{cases}$$

Signal Consistency Tests



Coding in the Tutorial Resources

```
def interpolate(series, delta_f, length=None)
```

在分頁中開啟　查看原始碼

Return a new PSD that has been interpolated to the desired delta_f.

Parameters

series : FrequencySeries
  Frequency series to be interpolated.
delta_f : float
  The desired delta_f of the output
length : None or int
  The desired number of frequency samples. The default is None,

```
ts =
prin
data
prin
data
prin

# Es
# Th
# On                                                          hat
# re
p =
p = interpolate(p, data[ifo].delta_f)
p = inverse_spectrum_truncation(p, int(2 * data[ifo].sample_rate), low_frequency_cutoff=15.0)
psd[ifo] = p

plt.plot(psd[ifo].sample_frequencies, psd[ifo], label=ifo)
```

Coding in the Tutorial Resources

```python
    ts = m.strain(ifo).highpass_fir(15, 512)
    print(ifo,len(ts))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048)
    print(ifo,len(data[ifo]))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048).crop(2, 2)
    print(ifo,len(data[ifo]))

    # Estimate the power spectral density of the data
    # This chooses to use 2s samples in the PSD estimate.
    # One should note that the tradeoff in segment length is that
    # resolving narrow lines becomes more difficult.
    p = data[ifo].psd(2)
    p = interpolate(p, data[ifo].delta_f)
    p = inverse_spectrum_truncation(p, int(2 * data[ifo].sample_rate), low_frequency_cutoff=
    psd[ifo] = p

    plt.plot(psd[ifo].sample_frequencies, psd[ifo], label=ifo)

plt.yscale('log')
plt.xscale('log')
plt.ylim(1e-47, 1e-41)
plt.xlim(20, 1024)
plt.ylabel('$Strain^2 / Hz$')
plt.xlabel('Frequency (Hz)')
plt.grid()
plt.legend()
plt.show()
```

interpolate

```python
def interpolate(series, delta_f, length=None)
```

Return a new PSD that has been interpolated to the desired delta_f.

Parameters

series : FrequencySeries
  Frequency series to be interpolated.
delta_f : float
  The desired delta_f of the output
length : None or int
  The desired number of frequency samples. The default is None, so it will be calculated from the given `series` and `delta_f`. But this will cause an inconsistency issue of length sometimes, so if `length` is given, then just use it.

Returns

interpolated series : FrequencySeries
  A new FrequencySeries that has been interpolated.

Coding in the Tutorial Resources

```
    ts = m.strain(ifo).highpass_fir(15, 512)
    print(ifo,len(ts))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048)
    print(ifo,len(data[ifo]))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048).crop(2, 2)
    print(ifo,len(data[ifo]))

    # Estimate the power spectral density of the data
    # This chooses to use 2s samples in the PSD estimate.
    # One should note that the tradeoff in segment length is that
    # resolving narrow lines becomes more difficult.
    p = data[ifo].psd(2)
    p = interpolate(p, data[ifo].delta_f)
    p = inverse_spectrum_truncation(p, int(2 * data[ifo].sample_rate), low_
    psd[ifo] = p

    plt.plot(psd[ifo].sample_frequencies, psd[ifo], label=ifo)

plt.yscale('log')
plt.xscale('log')
plt.ylim(1e-47, 1e-41)
plt.xlim(20, 1024)
plt.ylabel('$Strain^2 / Hz$')
plt.xlabel('Frequency (Hz)')
plt.grid()
plt.legend()
plt.show()
```

```
ifos = ['H1', 'L1', 'V1']
data = {}
psd = {}
print(len(m.strain('H1')))

plt.figure(figsize=[10, 5])

for ifo in ifos:
    # Read in and precondition the data
    ts = m.strain(ifo).highpass_fir(15, 512)
    print(ifo,len(ts))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048)
    print(ifo,len(data[ifo]))
    data[ifo] = resample_to_delta_t(ts, 1.0/2048).crop(2, 2)
    print(ifo,len(data[ifo]))

    # Estimate the power spectral density of the data
    # This chooses to use 2s samples in the PSD estimate.
    # One should note that the tradeoff in segment length is that
    # resolving narrow lines becomes more difficult.
    p = data[ifo].psd(2)
    p = interpolate(p, data[ifo].delta_f)
    p = inverse_spectrum_truncation(p, int(2 * data[ifo].sample_rate), low_frequency_cutoff=15.0)
    psd[ifo] = p

    plt.plot(psd[ifo].sample_frequencies, psd[ifo], label=ifo)

plt.yscale('log')
plt.xscale('log')
plt.ylim(1e-47, 1e-41)
plt.xlim(20, 1024)
plt.ylabel('$Strain^2 / Hz$')
plt.xlabel('Frequency (Hz)')
plt.grid()
plt.legend()
plt.show()
```

```
131072
H1 130048
H1 65024
H1 56832
L1 130048
L1 65024
L1 56832
V1 130048
V1 65024
V1 56832
```

Plotting frequency evolution of TD waveform

```
import matplotlib.pyplot as pp
from pycbc import waveform


for phase_order in [2, 3, 4, 5, 6, 7]:
    hp, hc = waveform.get_td_waveform(approximant='SpinTaylorT4',
                                      mass1=10, mass2=10,
                                      phase_order=phase_order,
                                      delta_t=1.0/4096,
                                      f_lower=100)

    hp, hc = hp.trim_zeros(), hc.trim_zeros()
    amp = waveform.utils.amplitude_from_polarizations(hp, hc)
    f = waveform.utils.frequency_from_polarizations(hp, hc)

    pp.plot(f.sample_times, f, label="PN Order = %s" % phase_order)
```