CBC Search in Practice & Applications to Astrophysics

Kyungmin Kim
(Ewha Womans Univ.)

2022 NRGW Summer School

Contents

- Part I: CBC Search in Practice w/ Python
  - Introduction to matched filtering
  - Matched filtering in action
  - Signal consistency and significance

- Part II: Applications to Astrophysics
  - Brief recap of GW astrophysics
  - Applications to machine learning (ML)-based astrophysics w/ selected examples

CBC Search in Practice w/ Python

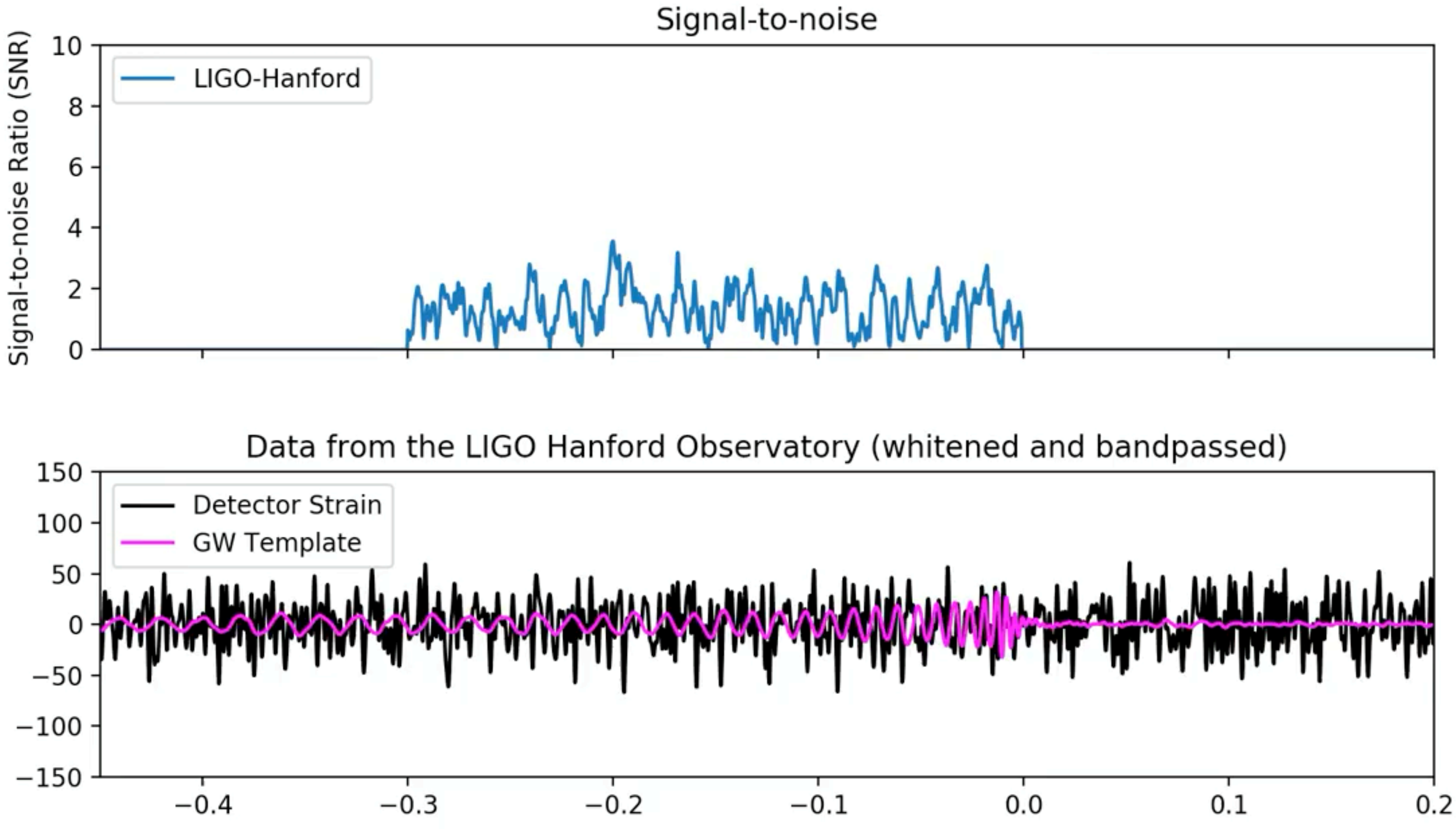
CBC Search in Practice w/ Python

- References
  - Main: GW Open Data Workshop (ODW) 2022 - Day 2 Tutorial
    - Homepage: https://www.gw-openscience.org/odw/odw2022
    - Github: https://github.com/gw-odw/odw-2022/tree/main/Tutorials/Day_2
  - Additional: GW Open Science Center
    - "Signal processing with GW150914 Open Data"
    - https://www.gw-openscience.org/s/events/GW150914/GW150914_tutorial.html

- ODW Day 2 Tutorial materials:
  - Tuto_2.1_Matched_filtering_introduction
  - Tuto_2.2_Matched_filtering_in_action
  - Tuto_2.3_Signal_consistency_and_significance

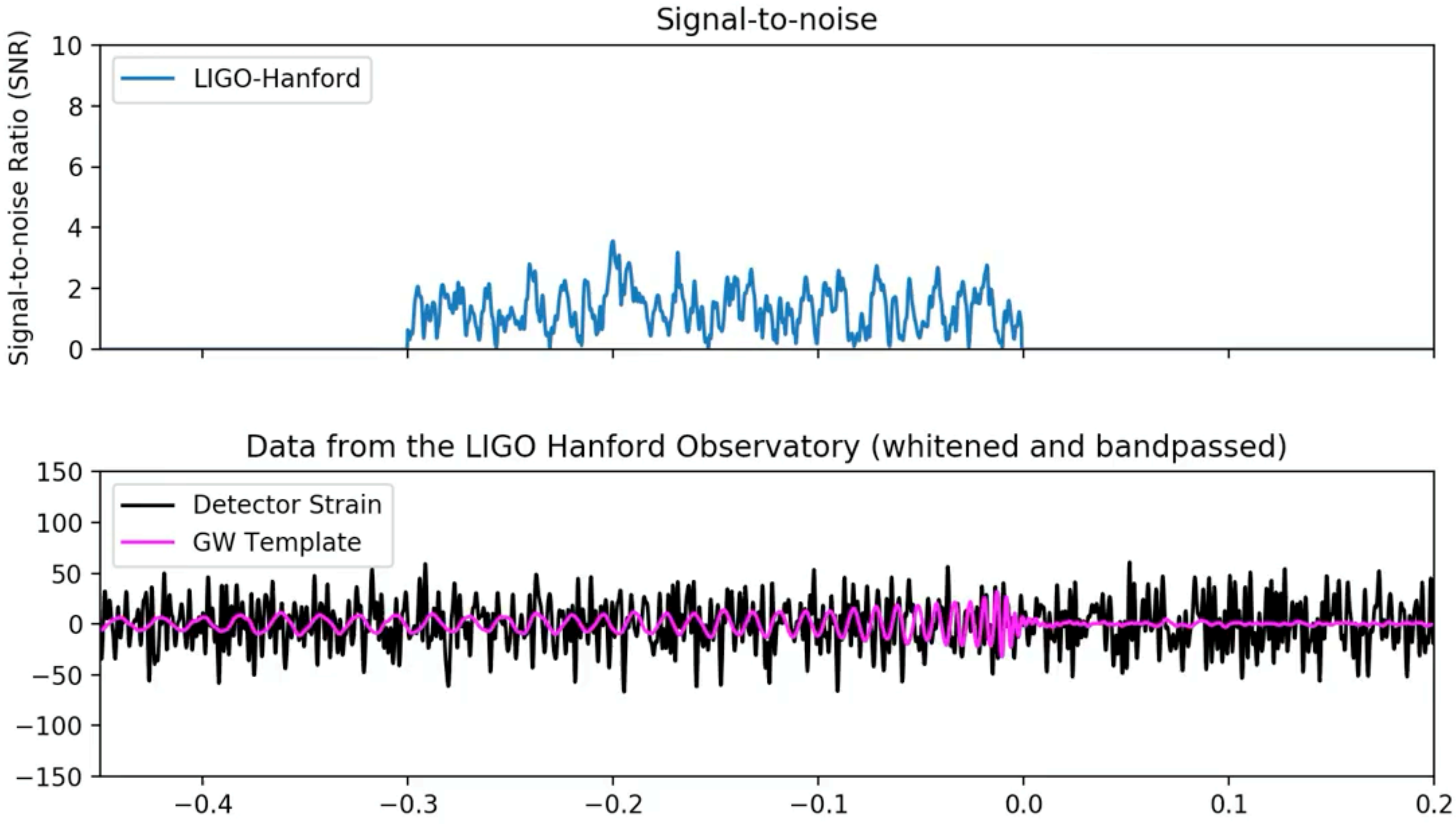
Introduction to matched filtering

- Matched filtering
  - optimal method for “detecting” known signals in Gaussian noise via computing cross-correlation



Introduction to matched filtering

- Matched filtering
  - optimal method for "detecting" known signals in Gaussian noise via computing cross-correlation

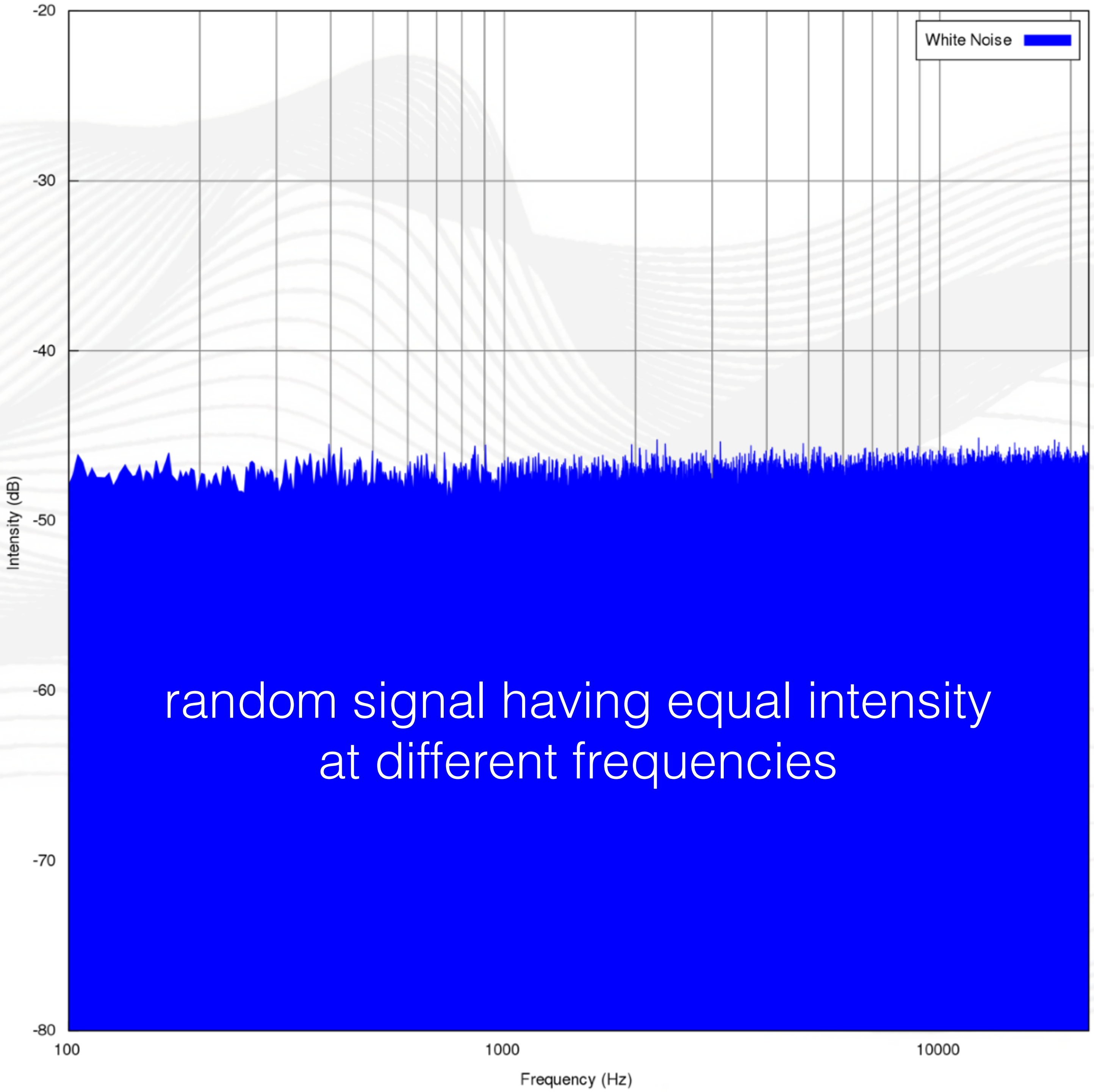


Introduction to matched filtering

- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.
  - What’s white noise?

Introduction to matched filtering

- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.
  - What’s white noise?



random signal having equal intensity at different frequencies

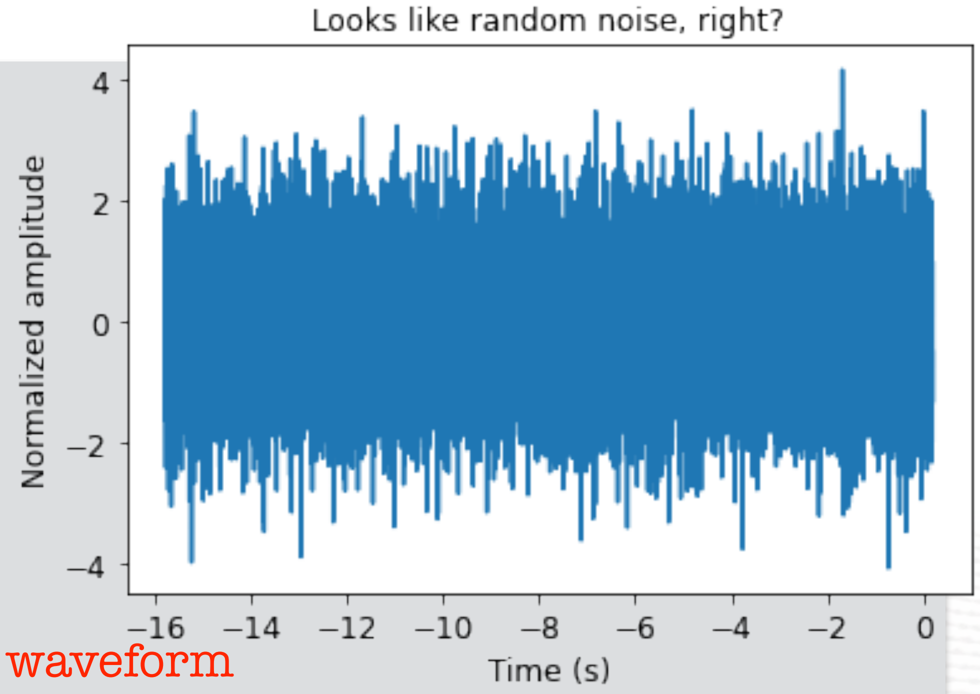
Introduction to matched filtering

- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.

```
import numpy

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

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


from pycbc.waveform import get_td_waveform  # to generate time series waveform

apx = ‘IMRPhenomD’  # Specify a waveform model; IMRPhenomD is a phenomenological
                     Inspiral-Merger-Ringdown waveform model
                     (dosen’t include effects such as non-aligned spins or high order modes)

hp, hx = get_td_waveform(approximant=apx, mass1=10, mass2=10, delta_t=1.0/sample_rate,
                         f_lower=25)    # it returns ‘+’ and ‘×’ polarization modes of a GW signal

# use $h_+$ only for now. if you want to use a whole waveform, just sum hp and hx such as h = hp + hx.
hp = hp / max(numpy.correlate(hp, hp, mode=‘full’))**0.5    # to demonstrate the method on white noise
                                                             with amplitude $\mathcal{O}(1)$, we normalize our signal
                                                             so the cross-correlation of the signal with
                                                             itself will give a value of 1.
```

Introduction to matched filtering

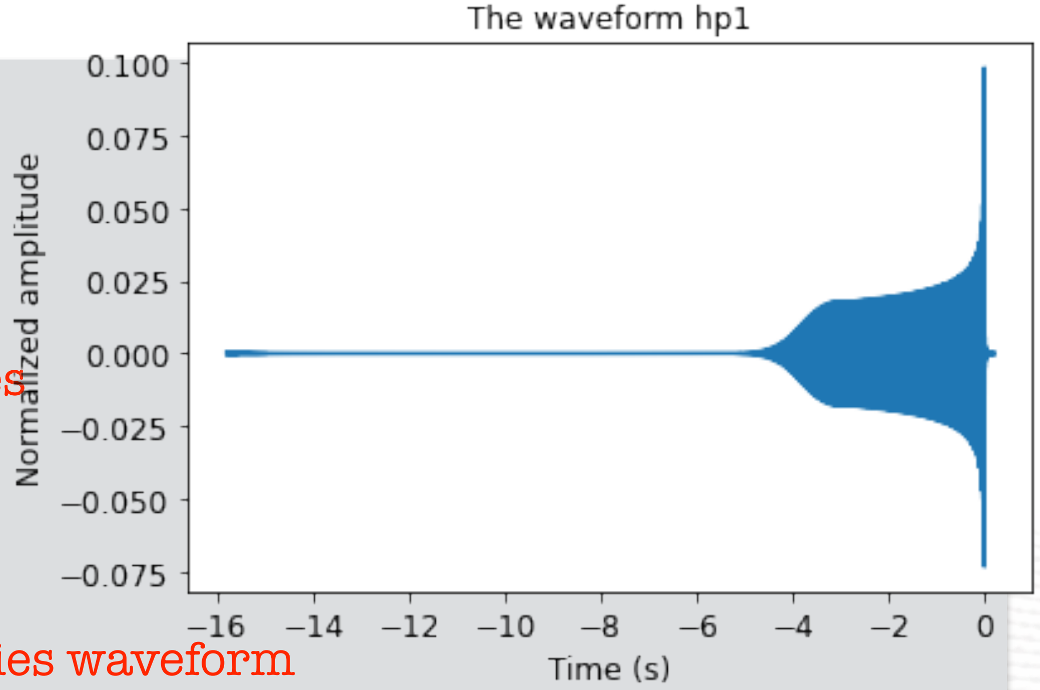
- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.


Looks like random noise, right?

```
import numpy

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

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


from pycbc.waveform import get_td_waveform  # to generate time series waveform

apx = ‘IMRPhenomD’  # Specify a waveform model; IMRPhenomD is a phenomenological
                      Inspiral-Merger-Ringdown waveform model
                      (dosen’t include effects such as non-aligned spins or high order modes)

hp, hx = get_td_waveform(approximant=apx, mass1=10, mass2=10, delta_t=1.0/sample_rate,
                                        f_lower=25)     # it returns ‘+’ and ‘×’ polarization modes of a GW signal

# use $h_+$ only for now. if you want to use a whole waveform, just sum hp and hx such as h = hp + hx.
hp = hp / max(numpy.correlate(hp, hp, mode=‘full’)) * *0.5     # to demonstrate the method on white noise
                                                                  with amplitude $\mathcal{O}(1)$, we normalize our signal
                                                                  so the cross-correlation of the signal with
                                                                  itself will give a value of 1.
```

Introduction to matched filtering

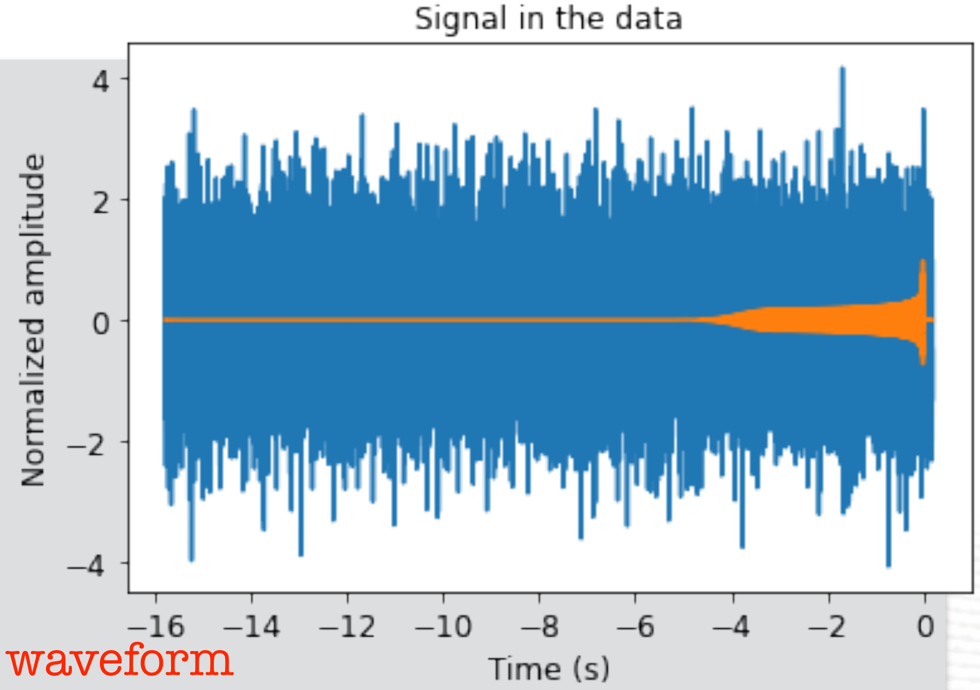
- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.



```
import numpy

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

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


from pycbc.waveform import get_td_waveform  # to generate time series waveform

apx = ‘IMRPhenomD’  # Specify a waveform model; IMRPhenomD is a phenomenological
                      Inspiral-Merger-Ringdown waveform model
                      (dosen’t include effects such as non-aligned spins or high order modes)

hp, hx = get_td_waveform(approximant=apx, mass1=10, mass2=10, delta_t=1.0/sample_rate,
                                        f_lower=25)     # it returns ‘+’ and ‘×’ polarization modes of a GW signal

# use $h_+$ only for now. if you want to use a whole waveform, just sum hp and hx such as h = hp + hx.
hp = hp / max(numpy.correlate(hp, hp, mode=‘full’))**0.5     # to demonstrate the method on white noise
                                                               with amplitude $\mathcal{O}(1)$, we normalize our signal
                                                               so the cross-correlation of the signal with
                                                               itself will give a value of 1.
```

Introduction to matched filtering

- Let’s learn how matched filtering works.
- Start with an example waveform in white noise.



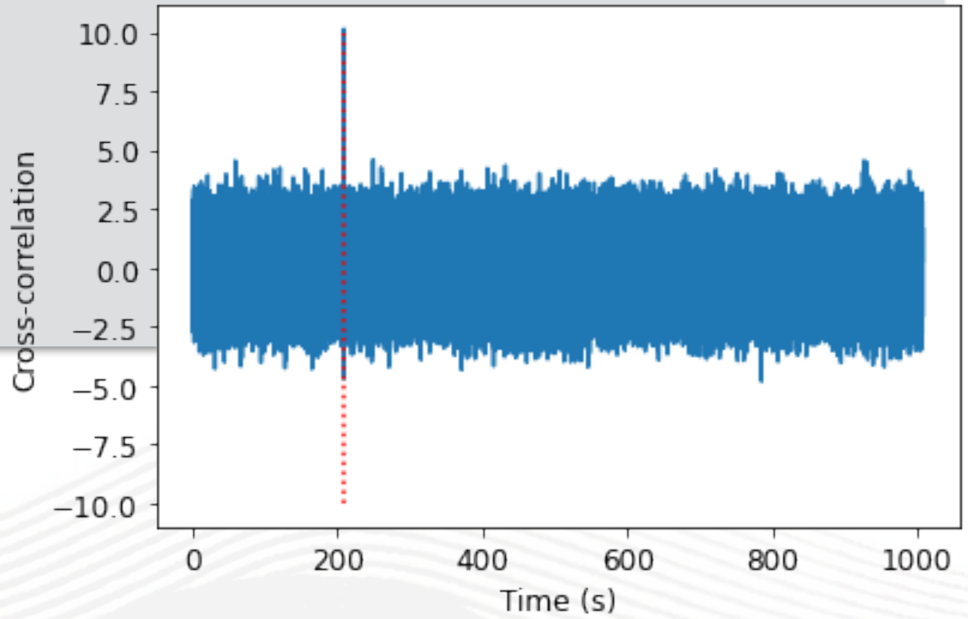
```
import numpy

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

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


from pycbc.waveform import get_td_waveform  # to generate time series waveform

apx = ‘IMRPhenomD’  # Specify a waveform model; IMRPhenomD is a phenomenological
                      Inspiral-Merger-Ringdown waveform model
                      (dosen’t include effects such as non-aligned spins or high order modes)

hp, hx = get_td_waveform(approximant=apx, mass1=10, mass2=10, delta_t=1.0/sample_rate,
                                    f_lower=25)      # it returns ‘+’ and ‘×’ polarization modes of a GW signal

# use $h_+$ only for now. if you want to use a whole waveform, just sum hp and hx such as h = hp + hx.
hp = hp / max(numpy.correlate(hp, hp, mode=‘full’))**0.5     # to demonstrate the method on white noise
                                                                  with amplitude $\mathcal{O}(1)$, we normalize our signal
                                                                  so the cross-correlation of the signal with
                                                                  itself will give a value of 1.
```

Introduction to matched filtering

```
# To search for this signal, we can cross-correlate the signal with the entire dataset.
# We do the cross-correlation in the time domain.

cross_correlation = numpy.zeros([len(data)-len(hp)])
hp_numpy = hp.numpy()
for i in range(len(data) - len(hp_numpy)):
   cross_correlation[i] = (hp_numpy * data[i:i+len(hp_numpy)]).sum()
```

Introduction to matched filtering

```
# To search for this signal, we can cross-correlate the signal with the entire dataset.
# We do the cross-correlation in the time domain.

cross_correlation = numpy.zeros([len(data)-len(hp)])
hp_numpy = hp.numpy()
for i in range(len(data) - len(hp_numpy)):
   cross_correlation[i] = (hp_numpy * data[i:i+len(hp_numpy)]).sum()
```



Introduction to matched filtering

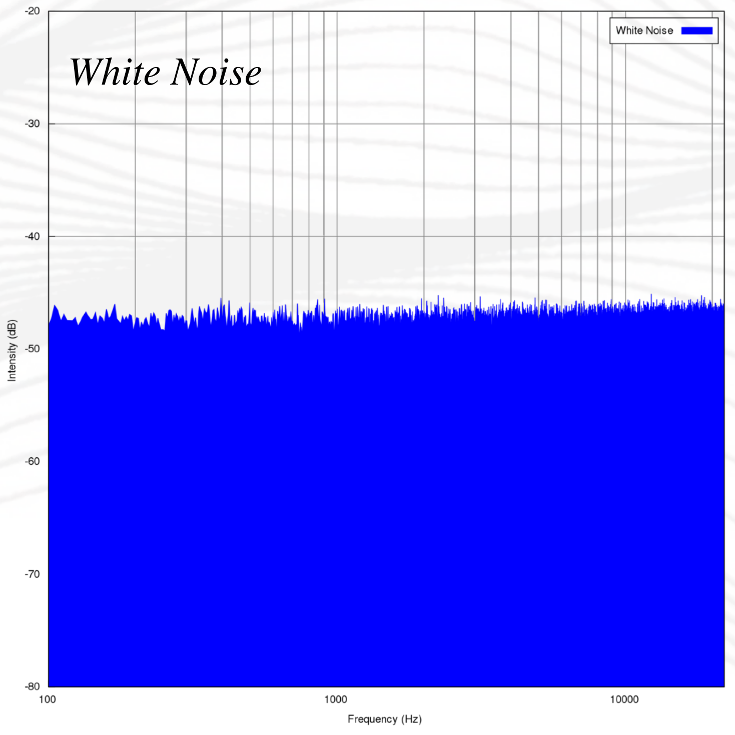
```
# To search for this signal, we can cross-correlate the signal with the entire dataset.
# We do the cross-correlation in the time domain.

cross_correlation = numpy.zeros([len(data)-len(hp)])
hp_numpy = hp.numpy()
for i in range(len(data) - len(hp_numpy)):
   cross_correlation[i] = (hp_numpy * data[i:i+len(hp_numpy)]).sum()
```

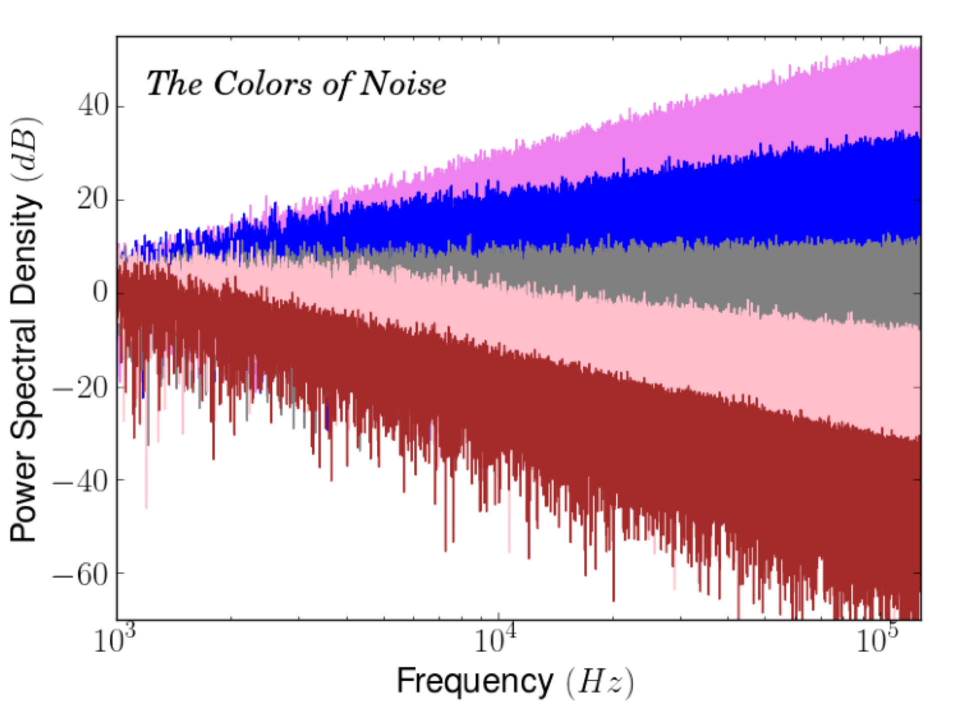
- Detection in Colored Noise
  - Let’s repeat the process, but generate a stretch of data colored with LIGO’s zero-detuned-high-power noise curve.

Introduction to matched filtering

```
# To search for this signal, we can cross-correlate the signal with the entire dataset.
# We do the cross-correlation in the time domain.

cross_correlation = numpy.zeros([len(data)-len(hp)])
hp_numpy = hp.numpy()
for i in range(len(data) - len(hp_numpy)):
   cross_correlation[i] = (hp_numpy * data[i:i+len(hp_numpy)]).sum()
```

- Detection in Colored Noise
  - Let's repeat the process, but generate a stretch of data colored with LIGO's zero-detuned-high-power noise curve.





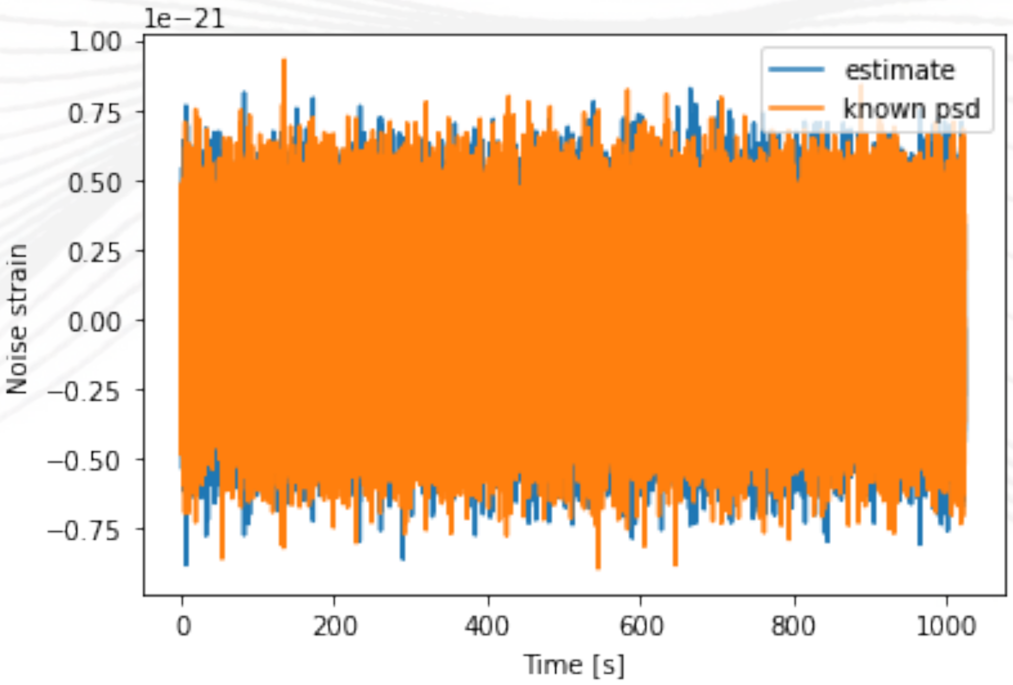
[Images:
from Wikipedia,
"Colors of noise"]

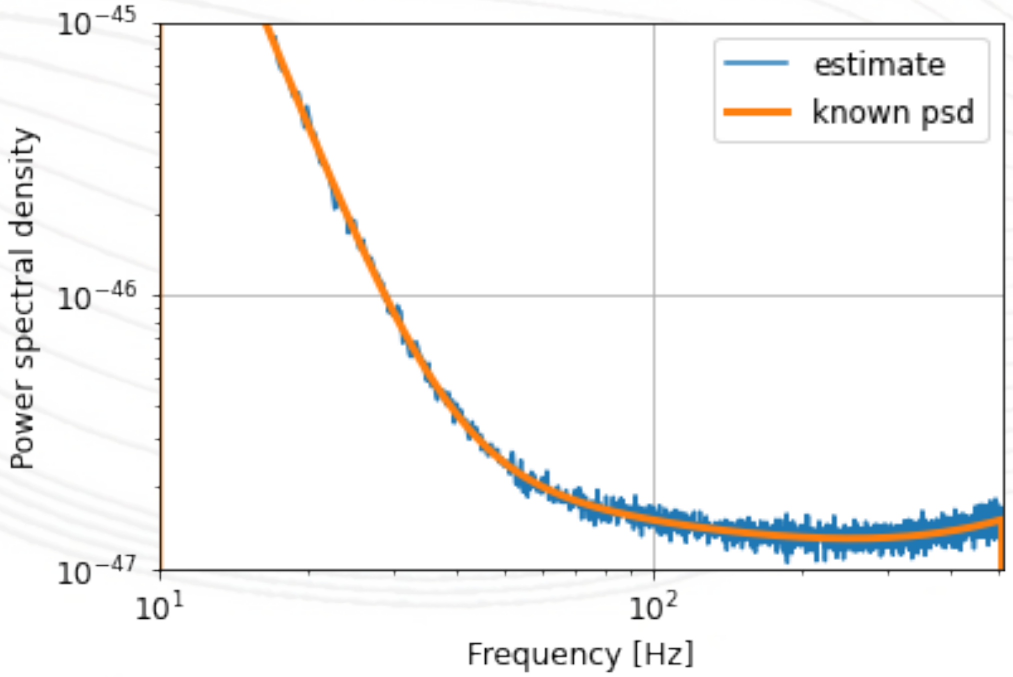
Introduction to matched filtering

```
import pycbc.noise, pycbc.psd

# The color of the noise matches a PSD which you provide, Advanced LIGO's zero-detuned-high-power noise curve
flow = 10.0
delta_f = 1.0 / 128
flen = int(sample_rate / (2*delta_f)) + 1  # sample_rate = 1024 samples per second
psd = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, flow)

# Generate colored noise
delta_t = 1.0 / sample_rate
ts = pycbc.noise.noise_from_psd(data_length*sample_rate, delta_t, psd, seed=127)

# Estimate the power spectral density for the noisy data using the "Welch" method.
# We'll choose 4 seconds PSD samples that are overlapped 50%
# For more details about the "Welch" method, see arXiv:gr-qc/0509116 (Section VI)
seg_len = int(4 / delta_t)
seg_stride = int(seg_len / 2)
estimated_psd = pycbc.psd.welch(ts, seg_len=seg_len, seg_stride=seg_stride)
```

Introduction to matched filtering

```
import pycbc.noise, pycbc.psd

# The color of the noise matches a PSD which you provide, Advanced LIGO’s zero-detuned-high-power noise curve
flow = 10.0
delta_f = 1.0 / 128
flen = int(sample_rate / (2*delta_f)) + 1  # sample_rate = 1024 samples per second
psd = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, flow)

# Generate colored noise
delta_t = 1.0 / sample_rate
ts = pycbc.noise.noise_from_psd(data_length*sample_rate, delta_t, psd, seed=127)

# Estimate the power spectral density for the noisy data using the “Welch” method.
# We’ll choose 4 seconds PSD samples that are overlapped 50%
# For more details about the “Welch” method, see arXiv:gr-qc/0509116 (Section VI)
seg_len = int(4 / delta_t)
seg_stride = int(seg_len / 2)
estimated_psd = pycbc.psd.welch(ts, seg_len=seg_len, seg_stride=seg_stride)
```





Introduction to matched filtering

- Then, all we need to do is to “whiten” both the data and the template waveform.

- Why do we need whitening?
    - From the PSD, we can see that the data are very strongly “colored”.
    - We can “whiten” the data suppressing the extra noise at low frequencies to better see the weak signals in the most sensitive band.
    - Whitening is always one of the first steps in astrophysical data analysis.

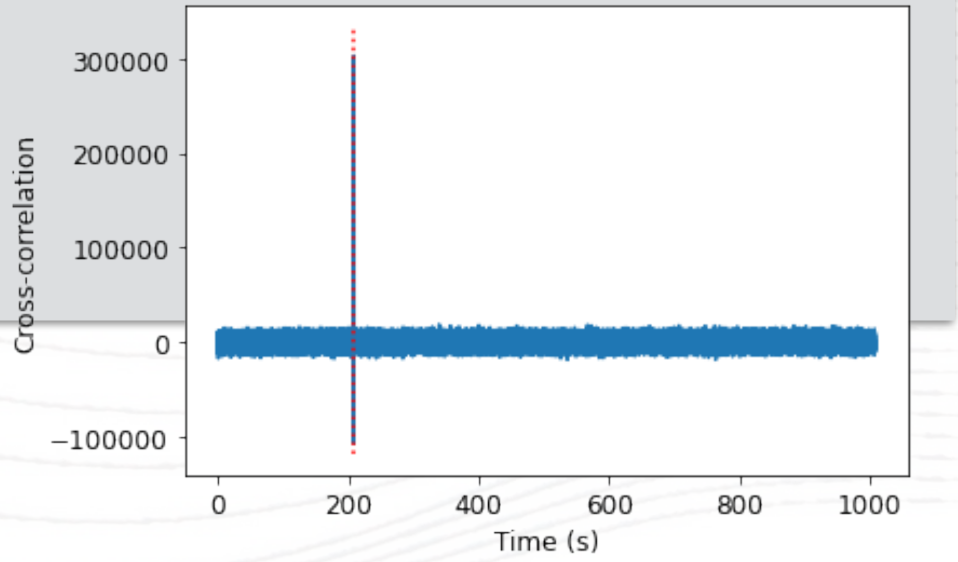
- This can be done, in the frequency domain, by dividing by the PSD.
  (This can be done in the time domain as well, but it’s more intuitive in the frequency domain.)

Introduction to matched filtering

```
# The PSD, sampled properly for the noisy data
delta_f = 1.0 / data_length  # data_length = 1024 seconds
flen = int(sample_rate / (2*delta_f)) + 1  # sample_rate = 1024 samples per second
psd_td = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# The PSD, sampled properly for the signal
delta_f = sample_rate / float(len(hp))
flen = int(sample_rate / (2*delta_f)) + 1
psd_hp = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# Convert both noisy data and the signal to frequency domain, and divide each by ASD,
# then covert back to time domain. This "whitens" the data and the signal template.
# Multiplying the signal template by 1E-21 puts it into realistic units of strain.
data_whitened = (ts.to_frequencyseries() / psd_td**0.5).to_timeseries()
hp_whitened = (hp.to_frequencyseries() / psd_hp**0.5).to_timeseries() * 1E-21

# Now let's re-do the correlation, in the time domain, but with
# whitened data and template.
cross_correlation = numpy.zeros([len(data)-len(hp1)])
hpn = hp_whitened.numpy()
datan = data_whitened.numpy()
for i in range(len(datan) - len(hpn)):
    cross_correlation[i] = (hpn * datan[i:i+len(hpn)]).sum()
```

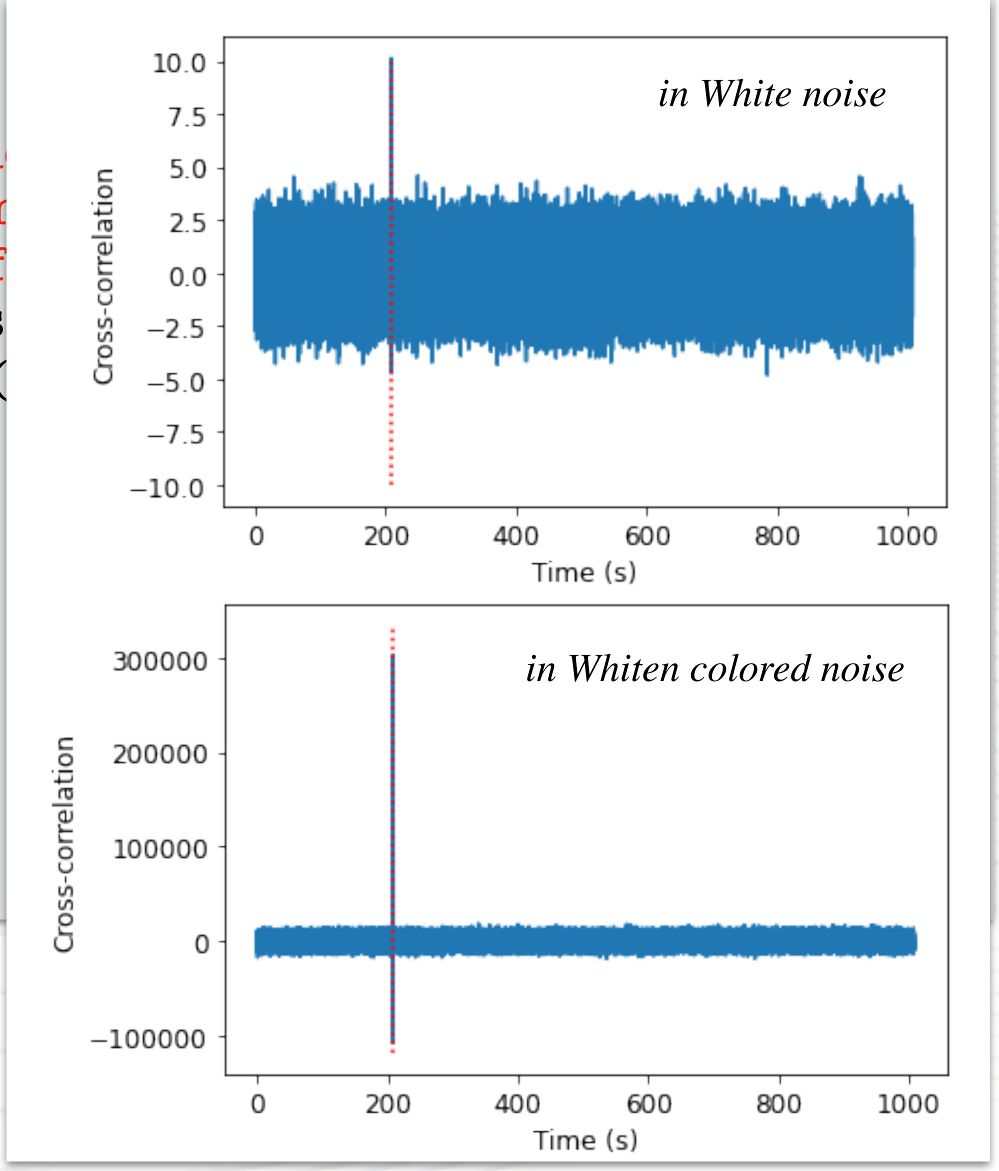
Introduction to matched filtering

```
# The PSD, sampled properly for the noisy data
delta_f = 1.0 / data_length  # data_length = 1024 seconds
flen = int(sample_rate / (2*delta_f)) + 1  # sample_rate = 1024 samples per second
psd_td = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# The PSD, sampled properly for the signal
delta_f = sample_rate / float(len(hp))
flen = int(sample_rate / (2*delta_f)) + 1
psd_hp = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# Convert both noisy data and the signal to frequency domain, and divide each by ASD,
# then covert back to time domain. This “whitens” the data and the signal template.
# Multiplying the signal template by 1E-21 puts it into realistic units of strain.
data_whitened = (ts.to_frequencyseries() / psd_td**0.5).to_timeseries()
hp_whitened = (hp.to_frequencyseries() / psd_hp**0.5).to_timeseries() * 1E-21

# Now let’s re-do the correlation, in the time domain, but with
# whitened data and template.
cross_correlation = numpy.zeros([len(data)-len(hp1)])
hpn = hp_whitened.numpy()
datan = data_whitened.numpy()
for i in range(len(datan) - len(hpn)):
    cross_correlation[i] = (hpn * datan[i:i+len(hpn)]).sum()
```



Introduction to matched filtering

```
# The PSD, sampled properly for the noisy data
delta_f = 1.0 / data_length  # data_length = 1024 seconds
flen = int(sample_rate / (2*delta_f)) + 1  # sample_rate = 1024 samples per second
psd_td = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# The PSD, sampled properly for the signal
delta_f = sample_rate / float(len(hp))
flen = int(sample_rate / (2*delta_f)) + 1
psd_hp = pycbc.psd.aLIGOZeroDetHighPower(flen, delta_f, 0)

# Convert both noisy data and the signal to frequency domain, and divi
# then covert back to time domain. This “whitens” the data and the sign
# Multiplying the signal template by 1E-21 puts it into realistic units of
data_whitened = (ts.to_frequencyseries() / psd_td**0.5).to_timeseries
hp_whitened = (hp.to_frequencyseries() / psd_hp**0.5).to_timeseries(

# Now let’s re-do the correlation, in the time domain, but with
# whitened data and template.
cross_correlation = numpy.zeros([len(data)-len(hp1)])
hpn = hp_whitened.numpy()
datan = data_whitened.numpy()
for i in range(len(datan) - len(hpn)):
   cross_correlation[i] = (hpn * datan[i:i+len(hpn)]).sum()
```



Matched filtering in action

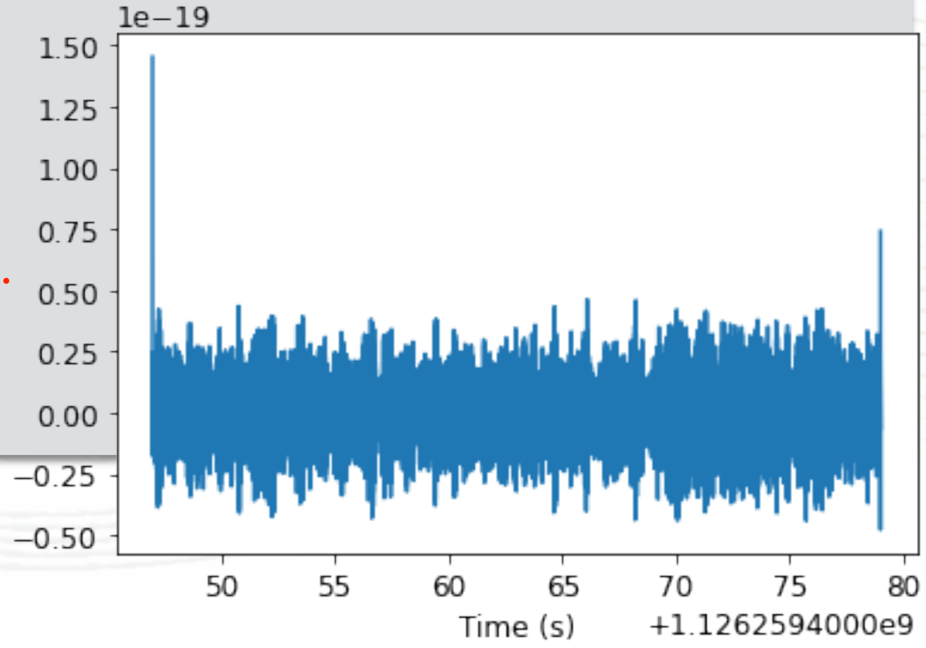
- Looking for a specific signal in the data
  - If you know what signal you are looking for in the data, then matched filtering is known to be the optimal method in Gaussian noise to extract the signal.
  - Even when the parameters of the signal are unknown, one can test any set of parameters interested in finding.

```
# Preconditioning the data.

# The purpose of preconditioning the data is to reduce the dynamic range of the data and to suppress low
frequency behavior that can introduce numerical artifacts. We may also wish to reduce the sample rate of the
data if high frequency content is not important.

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 2048 Hz.
strain = highpass(strain, 15.0)
strain = resample_to_delta_t(strain, 1.0/2048)
```

Matched filtering in action

- Looking for a specific signal in the data
  - If you know what signal you are looking for in the data, then matched filtering is known to be the optimal method in Gaussian noise to extract the signal.
  - Even when the parameters of the signal are unknown, one can test any set of parameters interested in finding.

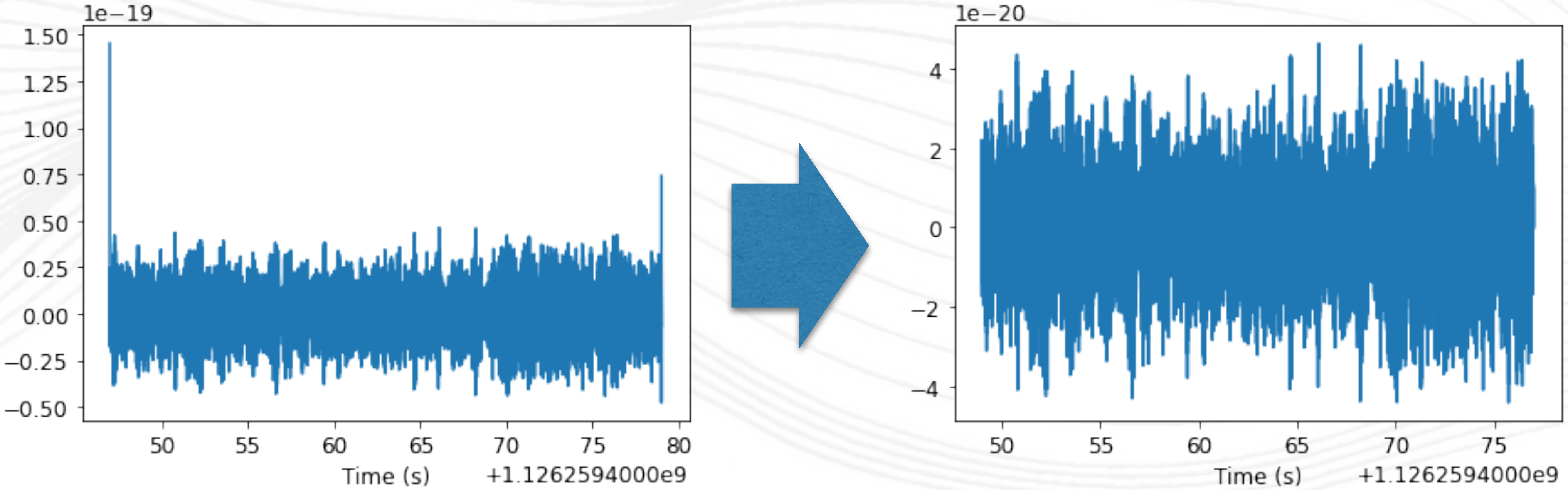
```
# Preconditioning the data.

# The purpose of preconditioning the data is to reduce the dynamic range of the data and to suppress low
frequency behavior that can introduce numerical artifacts. We may also wish to reduce the sample rate of the
data if high frequency content is not important.

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 2048 Hz.
strain = highpass(strain, 15.0)
strain = resample_to_delta_t(strain, 1.0/2048)
```



Matched filtering in action

- Filter wraparound
  - Note the spike in the data at the boundaries. This is caused by the highpass and resampling stages filtering the data. When the filter is applied to the boundaries, it wraps around to the beginning of the data. Since the data itself has a discontinuity (i.e. it is not cyclic) the filter itself will ring off for a time up to the length of the filter.
  - Even if a visible transient is not seen, we want to avoid filters that act on times which are not causally connected. To avoid this, we trim the ends of the data sufficiently to ensure that they do not wrap around the input. We will enforce this requirement in all steps of our filtering.

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



Matched filtering in action

- Calculate the power spectral density
  - Optimal matched filtering requires weighting the frequency components of the potential signal and data by the noise amplitude. We can view this as filtering the data with the time series equivalent of 1 / PSD. To ensure that we can control the effective length of the filter, we window the time domain equivalent of the PSD to a specific length.

```
from pycbc.psd import interpolate, inverse_spectrum_truncation

# 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.
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 the frequency
psd = inverse_spectrum_truncation(psd, int(4*conditioned.sample_rate), low_frequency_cutoff=15)
```

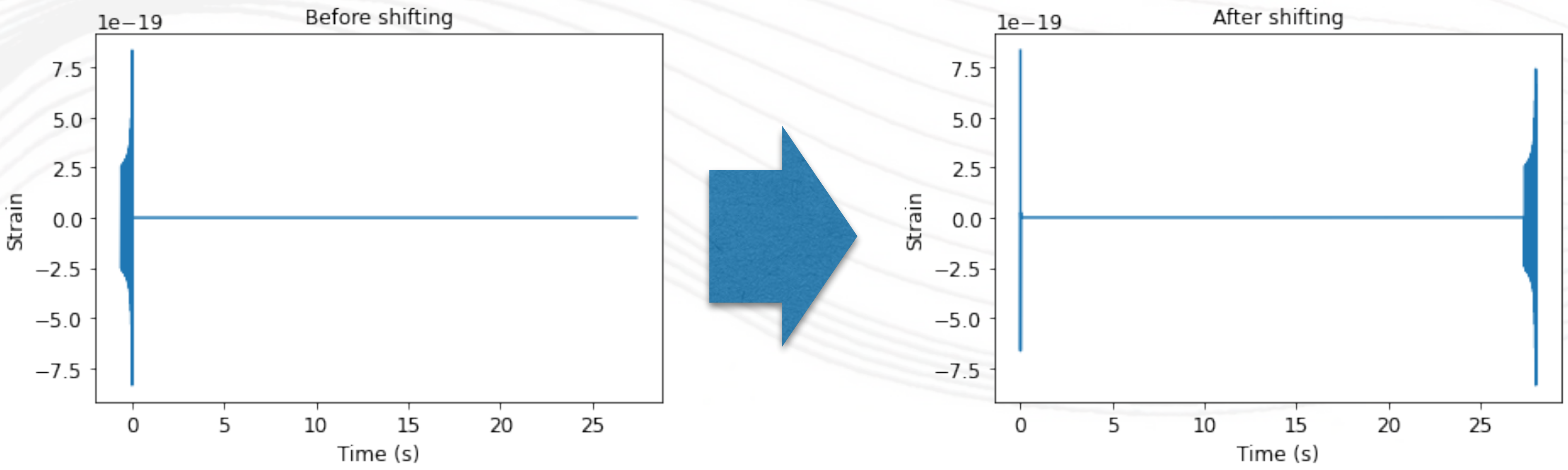
Matched filtering in action

- Make our signal model
  - In this case, we “know” what the signal parameters are. In a real search, we would grid over the parameters and calculate the SNR time series for each one.
  - We assume equal masses and non-rotating black holes.

```
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))
```

- The waveform begins at the start of the vector, so if we want the SNR time series to correspond to the approximate merger location (time), we need to shift the data so that the

```
template = hp.cyclic_time_shift(hp.start_time)
```



Matched filtering in action

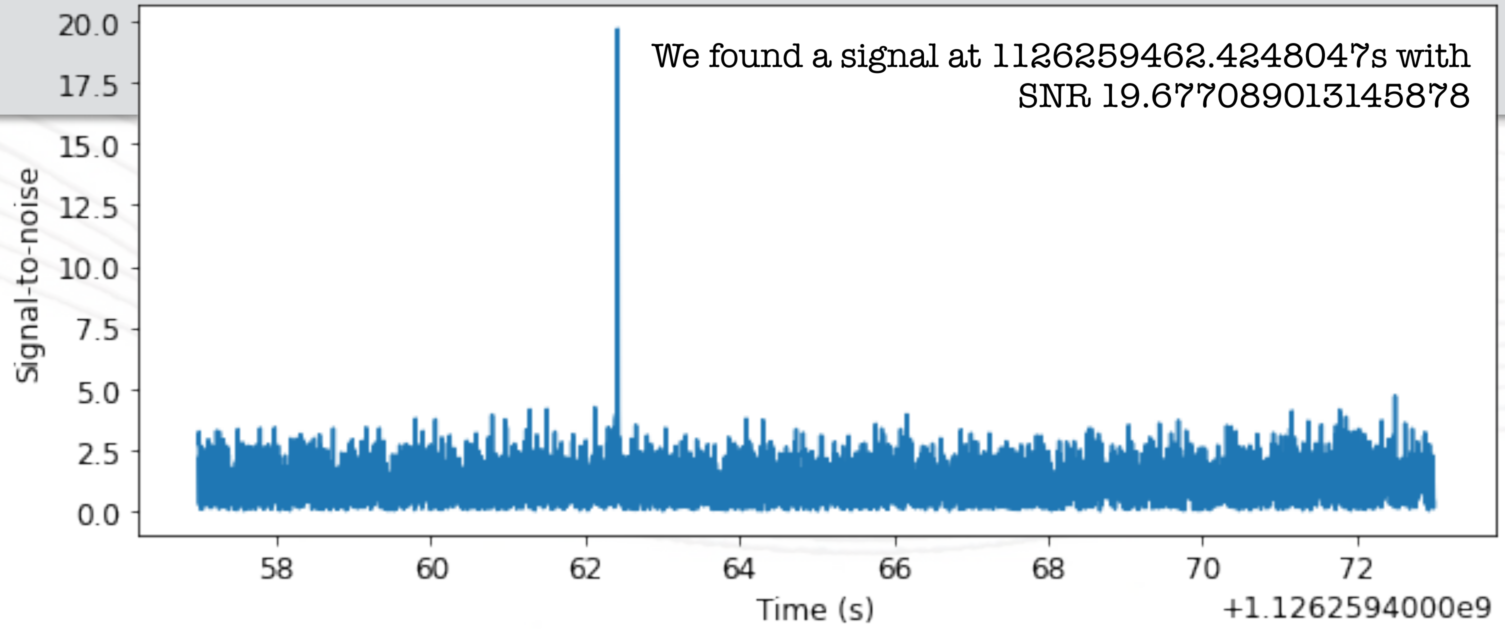
- Calculating the signal-to-noise time series
  - We'll take care to handle issues of filter corruption / wraparound by truncating the output time series. We need to account for both the length of the template and 1/PSD.

```
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 seconds 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)

peak = abs(snr).numpy().argmax(). # returns the index of peak SNR
snrp = snr[peak]
time = snr.sample_times[peak]
```

Matched filtering in action

- Calculating the signal-to-noise time series
    - We’ll take care to handle issues of filter corruption / wraparound by truncating the output time series. We need to account for both the length of the template and 1/PSD.

```
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 seconds 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)

peak = abs(snr).numpy().argmax(). # returns the index of peak SNR
snrp = snr[peak]
time = snr.sample_times[peak]
```



We found a signal at 1126259462.4248047s with SNR 19.677089013145878

Matched filtering in action

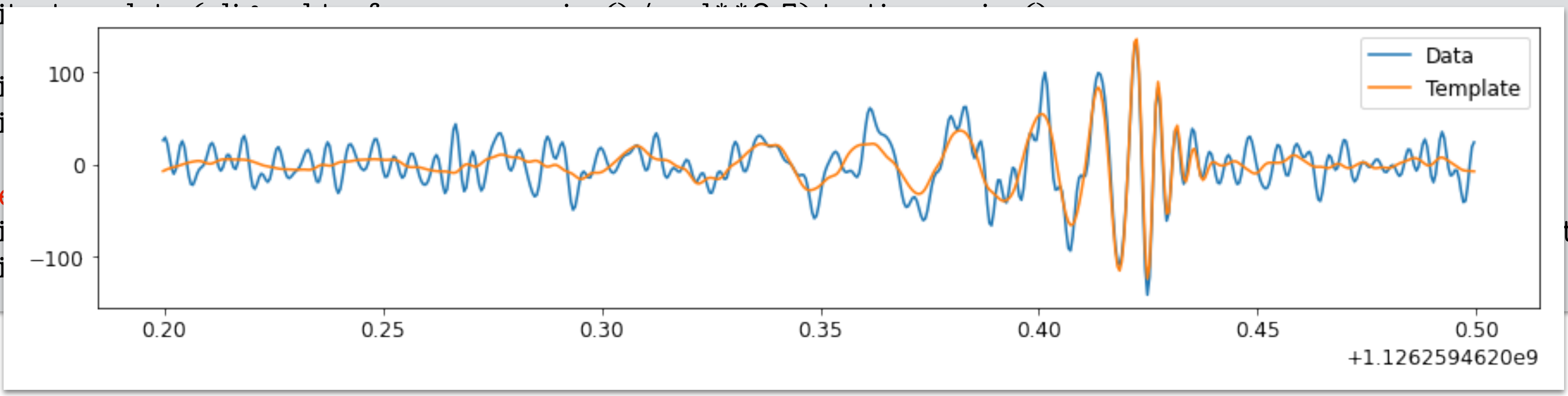
- Visualize the overlap between the signal and the data

```
from pycbc.filter import sigma

# 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

# To compare the data and signal on equal footing, and to concentrate on the frequency range that is important, we whiten both the template and the data.
# Then, bandpass both the data and template between 30-300 Hz. In this way, any signal that is in the data is transformed in the same way that the template is.
white_data = (conditioned.to_frequencyseries() / psd**0.5).to_timeseries()
white_template (aligned.to_frequencyseries() / psd**0.5).to_timeseries()

white_data = white_data.highpass_fir(30, 512).lowpass_fir(300, 512)
white_template = white_template.highpass_fir(30, 512).lowpass_fir(300, 512)

# Select the time around the merger
white_data = white_data.time_slice(merger.time-.2, merger.time+.1). # take [-0.2s, +0.1s] around the merger time
white_template = white_template.time_slice(merger.time-.2, merger.time+.1)
```

Matched filtering in action

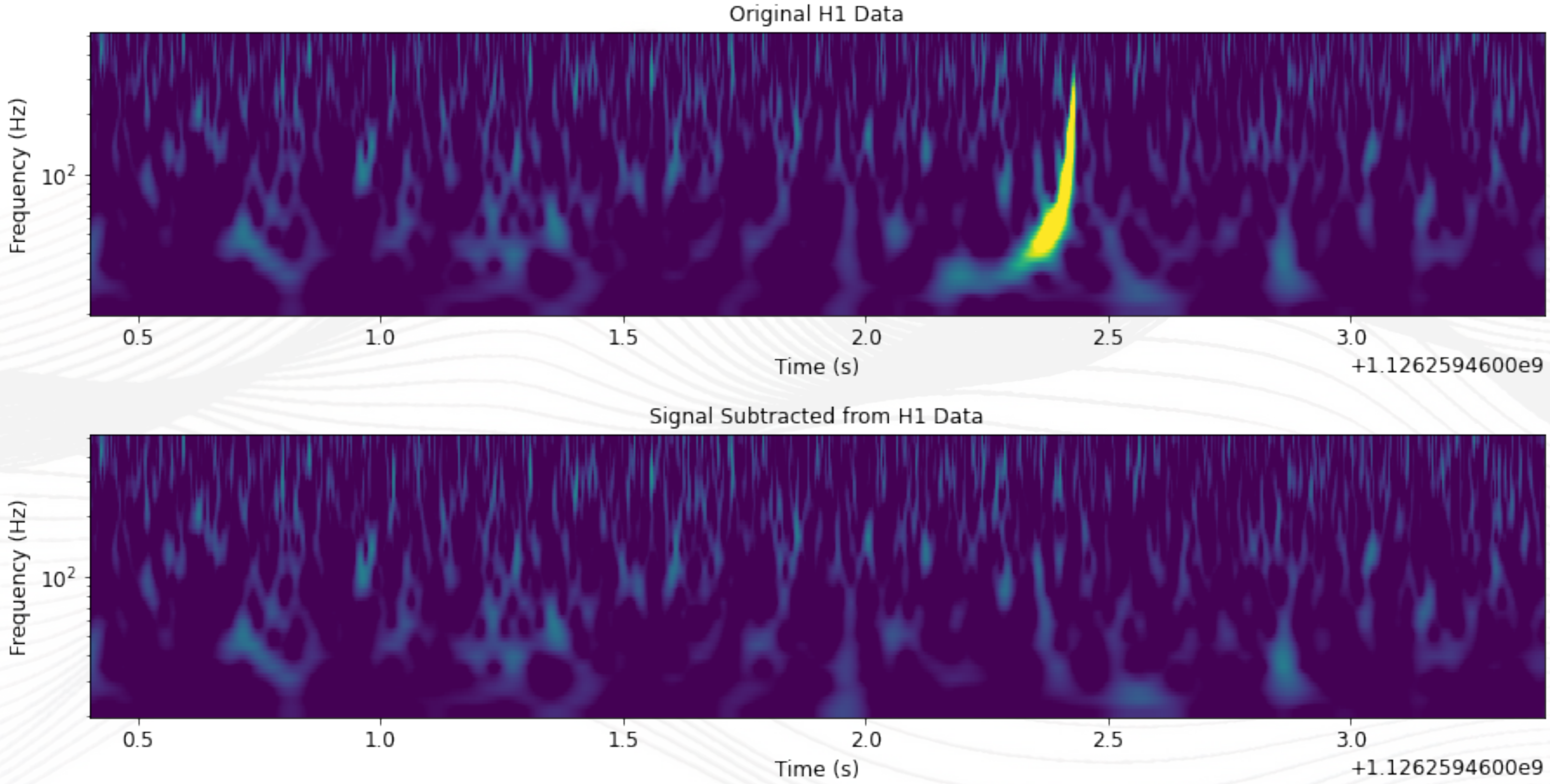
- Visualize the overlap between the signal and the data

```
from pycbc.filter import sigma

# 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

# To compare the data and signal on equal footing, and to concentrate on the frequency range that is important,
we whiten both the template and the data.
# Then, bandpass both the data and template between 30-300 Hz. In this way, any signal that is in the data is
transformed in the same way that the template is.
white_data = (conditioned.to_frequencyseries() / psd**0.5).to_timeseries()
white_template = (aligned.to_frequencyseries() / psd**0.5).to_timeseries()

whi
whi

# Se
whi                                                                                                   time
whi
```



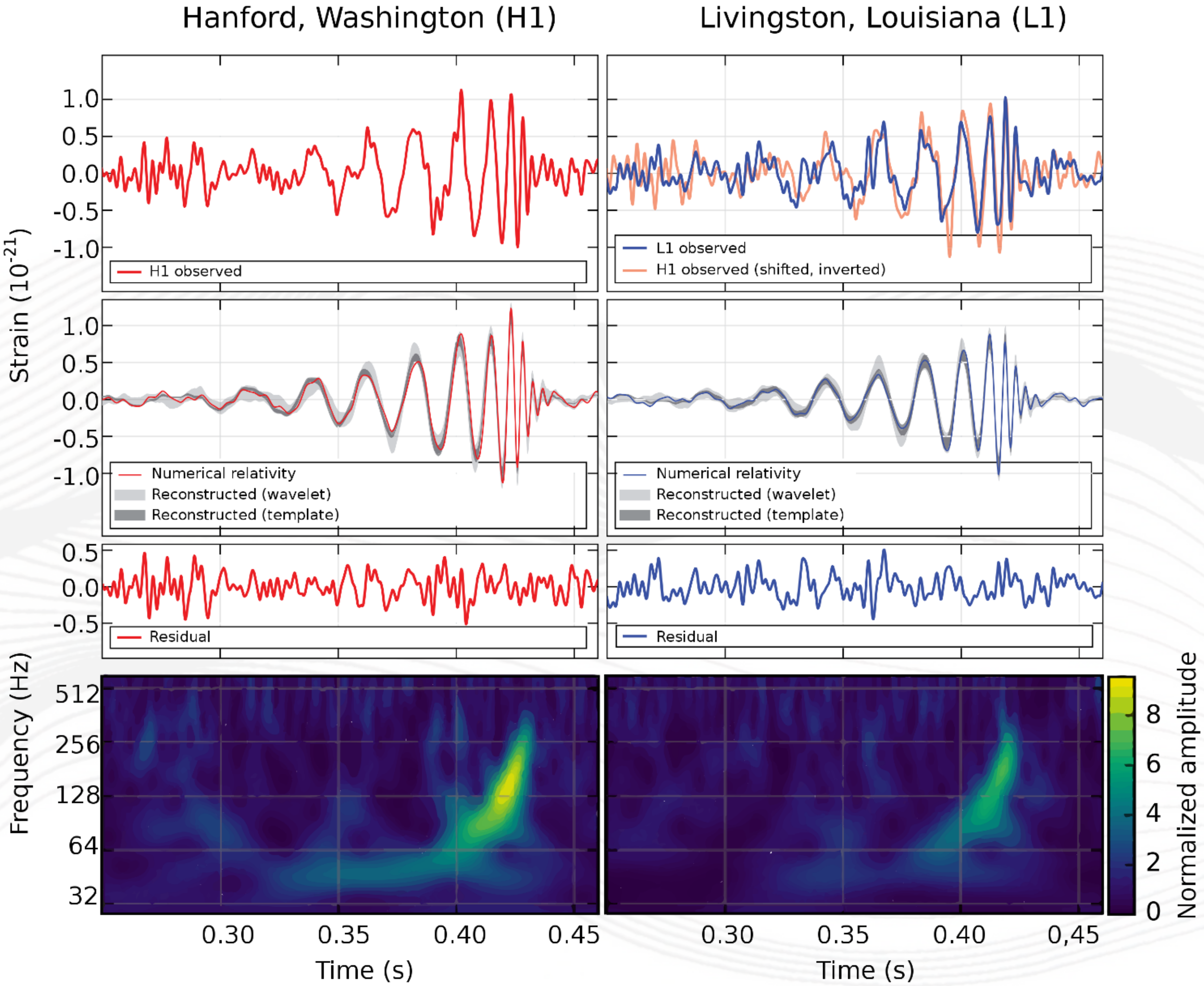
Matched filtering in action

- Subtracting the signal from the data
  - Now that we've aligned the template we can simply subtract it. Let's see it how that looks in the time-frequency plots.



Matched filtering in action

- Subtracting the signal from the data in reality.



[Figure from Abbott+ (2016, PRL)]

Signal consistency and significance

- How well is the data actually fitting our model?
  - χ^2-based signal consistency test is a standard one for the purpose.

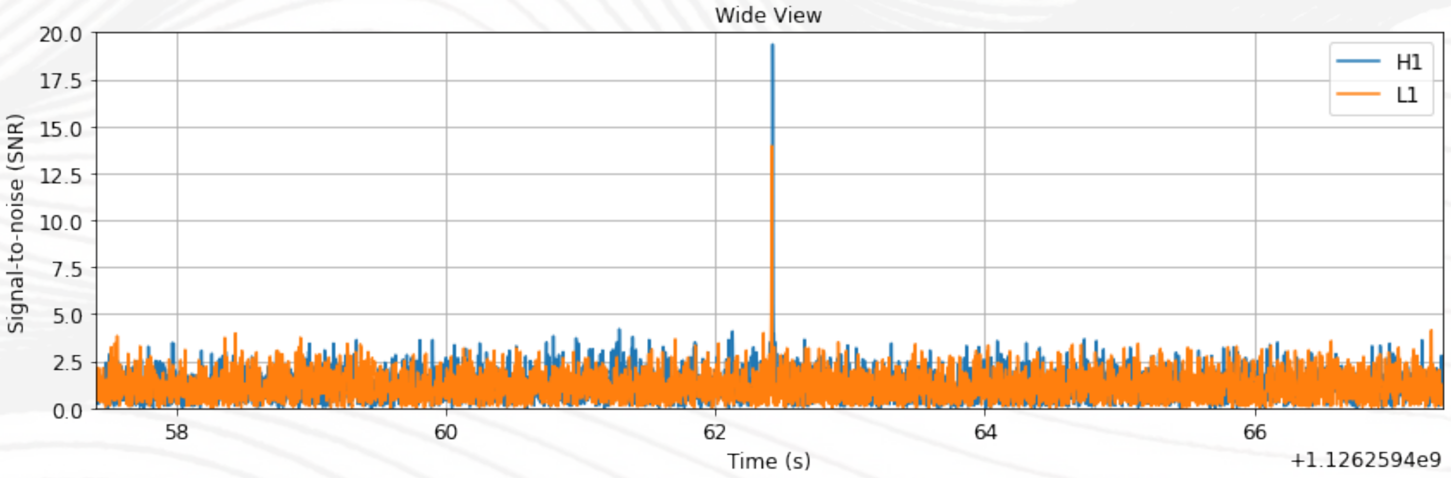
$$\chi^2 = \sum_{i=0}^{p} (\rho_i - \rho/p)^2$$

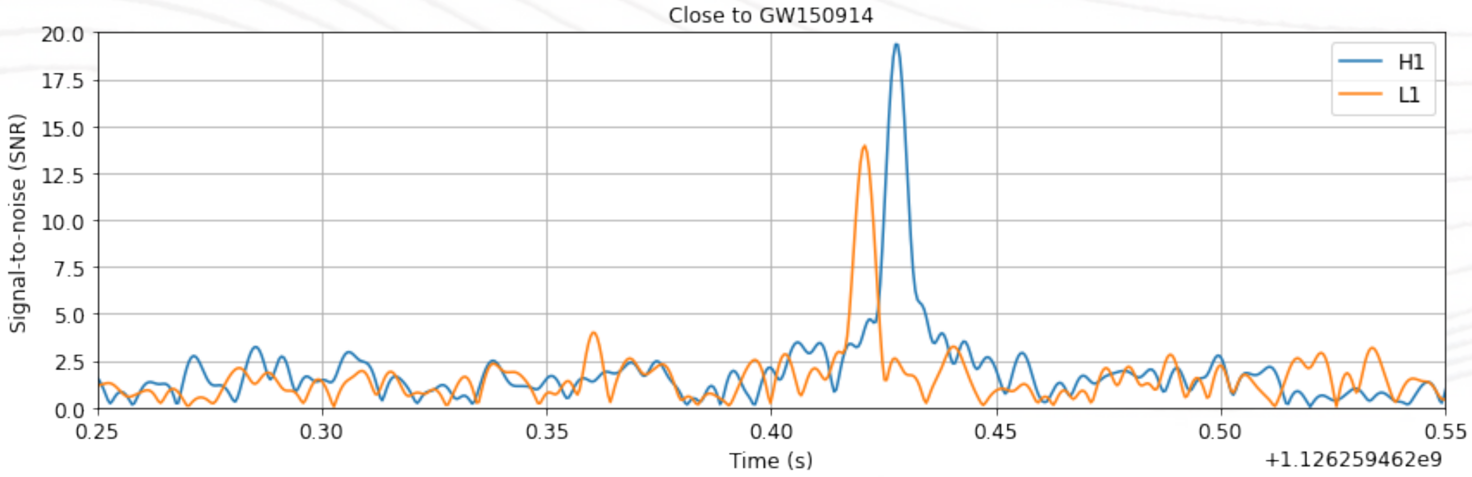
  - Schematically, we chop up our template into p number of bins and see how much each contributes to the SNR (ρ_i).
- Now, we use both LIGO-Hanford (H1) and LIGO-Livingston (L1) data of GW150914.

```
merger = Merger(“GW150914”)

ifos = [‘H1’, ‘L1’]
from pycbc.vetos import power_chisq
data = { }
psd = { }

for ifo in ifos:
   ts = merger.strain(ifo).highpass_fir(20, 512)
   data[ifo] = resample_to_delta_t(ts, 1.0/2048).crop(2, 2)

   # Estimate the power spectral density of the data
   p = data[ifo].psd(4)
   p = interpolate(p, data[ifo].delta_f)
   p = inverse_spectrum_truncation(p, int(2 * data[ifo].sample_rate), low_frequency_cutoff=20.0)
   psd[ifo] = p
```

Signal consistency and significance

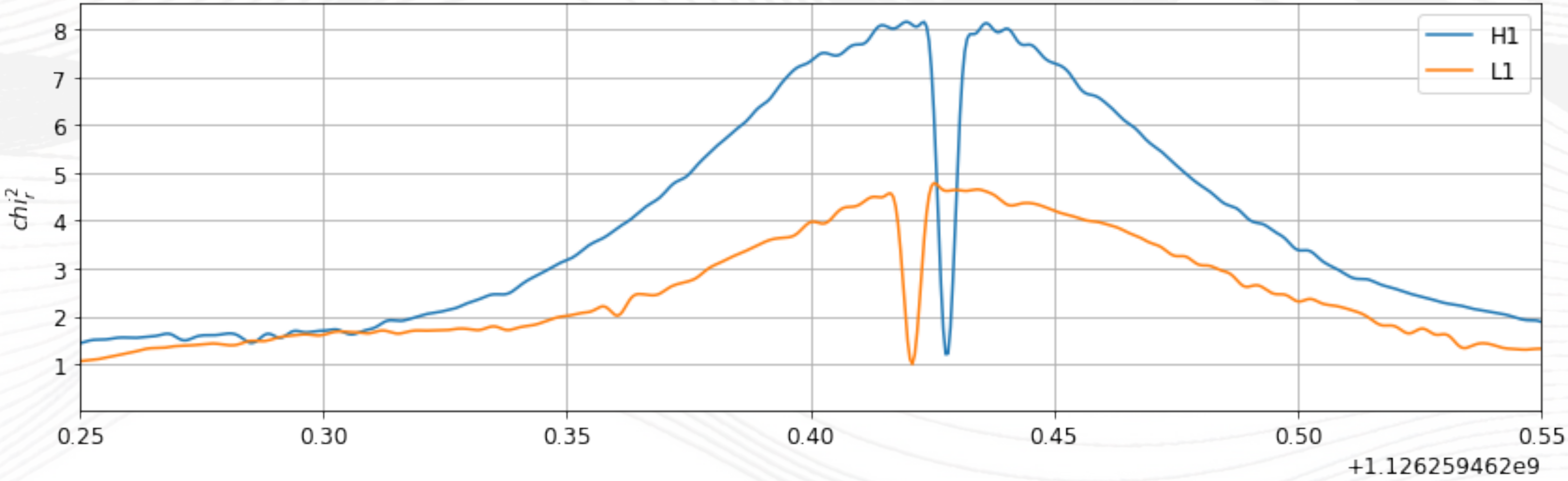
```
# Calculate the component mass of each black hole in the detector frame
cmass = (merger.median1d(“mass1”)+merger.meedian1d(“mass2”)) / 2  # This is in the source frame
cmass *= (1 + merger.median1d(“redshift”)). # apply redshift to get to the detector frame

hp, _ = get_fd_waveform(approximant=“IMRPhenomD”, mass1=cmass, mass2=cmass, f_lower=20.0,
                        delta_f=data[ifo].delta_f)
hp.resize(len(psd[ifo]))

# For each observatory, use this template to calculate the SNR time series
snr = { }
for ifo in ifos:
   snr[ifo] = matched_filtering(hp, data[ifo], psd=psd[ifo], low_frequency_cutoff=20)
   snr[ifo] = snr[ifo].crop(4+4, 4)
```





Signal consistency and significance

```
from pycbc.vetoes import power_chisq

chisq = { }
for ifo in ifos:
    # The number of bins to use. In principle, this choice is arbitrary. In practice, this is empirically tuned.
    nbins = 26
    chisq[ifo] = power_chisq(hp, data[ifo], nbins, psd[ifo], low_frequency_cutoff=20.0)
    chisq[ifo] = chisq[ifo].crop(4+4, 4)

    dof = nbins * 2 - 2
    chisq[ifo] /= dof
```



Signal consistency and significance

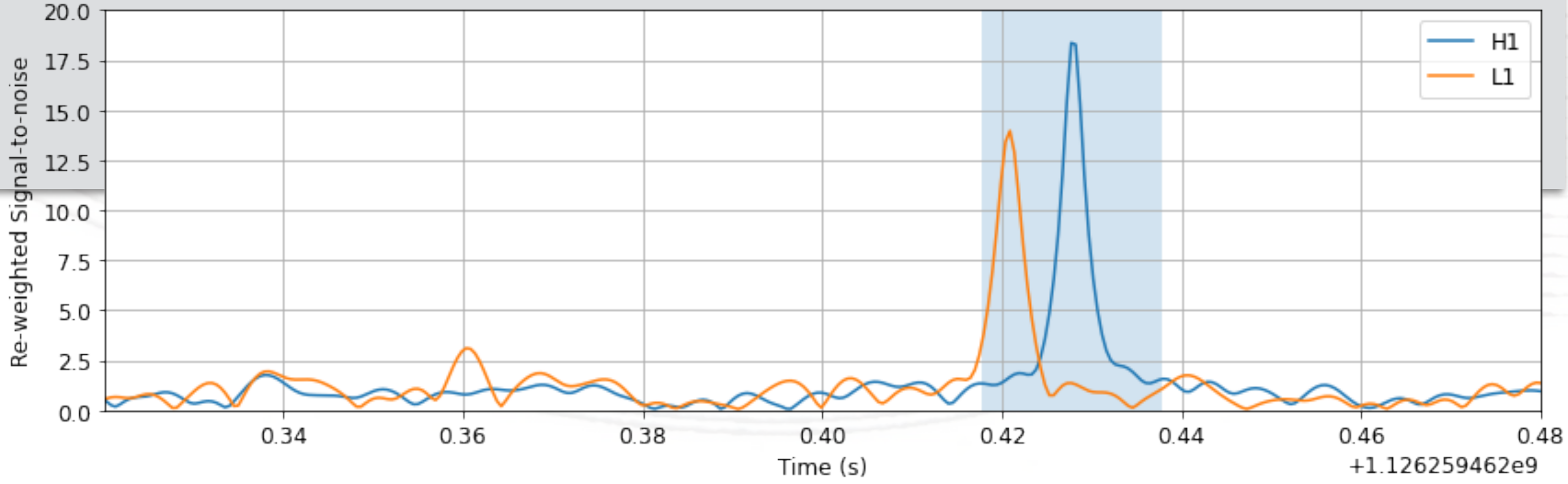
- We see the SNR of L1 is lower than that of H1. Let’s see the significance of L1 event.

```
from pycbc.detector import Detector

# Calculate the time of flight between the LIGO-Livingston and LIGO-Hanford
d = Detector(“L1”)
tof = { }
tof[‘H1’] = d.light_travel_time_to_detector(Detector(“H1”))

# Record the time of the peak in the LIGO-Hanford
ptime = { }
ptime[‘H1’] = snr[‘H1’].sample_times[snr[‘H1’].argmax()]

# Calculate the span of time that LIGO-Livingston peak could in principle happen in from time of flight
considerations.
start = ptime[‘H1’] - tof[‘H1’]
end = ptime[‘H1’] + tof[‘H1’]

# convert the times to indices along with how large the region is in number of samples
window_size = int((end - start) * snr[‘L1’].sample_rate)
sidx = int((start - snr[‘L1’].start_time) * snr[‘L1’].sample_rate)
eidx = sidx + window_size

# Calculate the “on-source” peak
onsource = snr[‘L1’][sidx:eidx].max()
```

Signal consistency and significance

- We see the SNR of L1 is lower than that of H1. Let’s see the significance of L1 event.

```
from pycbc.detector import Detector

# Calculate the time of flight between the LIGO-Livingston and LIGO-Hanford
d = Detector(“L1”)
tof = { }
tof[‘H1’] = d.light_travel_time_to_detector(Detector(“H1”))

# Record the time of the peak in the LIGO-Hanford
ptime = { }
ptime[‘H1’] = snr[‘H1’].sample_times[snr[‘H1’].argmax()]

# Calculate the span of time that LIGO-Livingston peak could in principle happen in from time of flight
considerations.
start = ptime[‘H1’] - tof[‘H1’]
end = ptime[‘H1’] + tof[‘H1’]

# convert the times to indices along with how large the region is in number of samples
window_size = int((end - start) * snr[‘L1’].sample_rate)
sidx = int((start - snr[‘L1’].start_time) * snr[‘L1’].sample_rate)
eidx = sidx + window_size

# Calculate the “on-source” peak
onsource = snr[‘L1’][sidx:eidx].max()
```



Signal consistency and significance

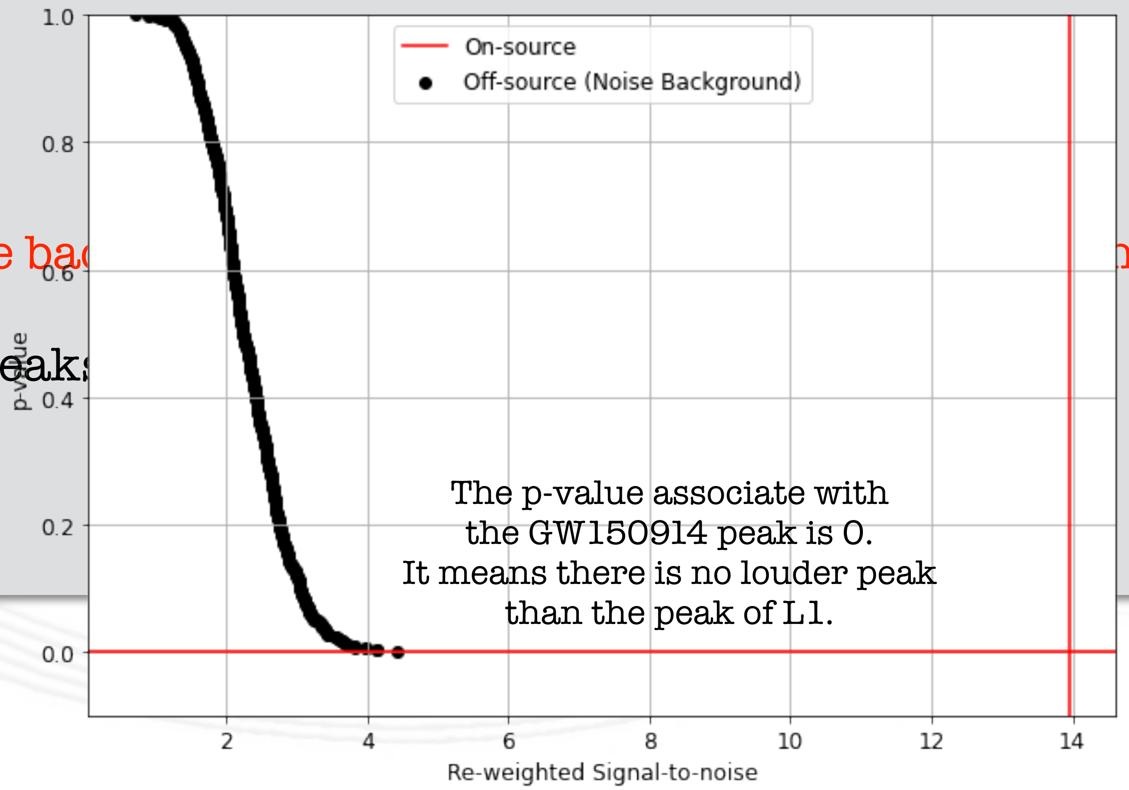
- Now that we’ve calculated the on-source peak, we should calculate the background peak values.
    - We do this by chopping up the time series into chunks that are the same size as our on-source and repeating the same peak finding (max) procedure.

```
import numpy

peaks = []
i = 0
while i + window_size < len(snr[‘L1’]):
   p = snr[‘L1’][i:i+window_size].max()
   peaks.append(p)
   i += window_size

   # skip past the onsource time
   if abs(i - sidx) < window_size:
      i += window_size * 2
peaks = numpy.array(peaks)

# The p-value is just the number of samples observed in the background with a value equal or higher than the on-source divided by the number of samples.
pcurve = numpy.arange(1, len(peaks)+1)[::-1] / float(len(peaks))
peaks.sort()

pvalue = (peaks > onsource).sum() / float(len(peaks))
```

Signal consistency and significance

- Now that we've calculated the on-source peak, we should calculate the background peak values.
    - We do this by chopping up the time series into chunks that are the same size as our on-source and repeating the same peak finding (max) procedure.

```
import numpy

peaks = []
i = 0
while i + window_size < len(snr['L1']):
   p = snr['L1'][i:i+window_size].max()
   peaks.append(p)
   i += window_size

   # skip past the onsource time
   if abs(i - sidx) < window_size:
      i += window_size * 2
peaks = numpy.array(peaks)

# The p-value is just the number of samples observed in the bac                                he on-
source divided by the number of samples.
pcurve = numpy.arange(1, len(peaks)+1)[::-1] / float(len(peak
peaks.sort()

pvalue = (peaks > onsource).sum() / float(len(peaks))
```

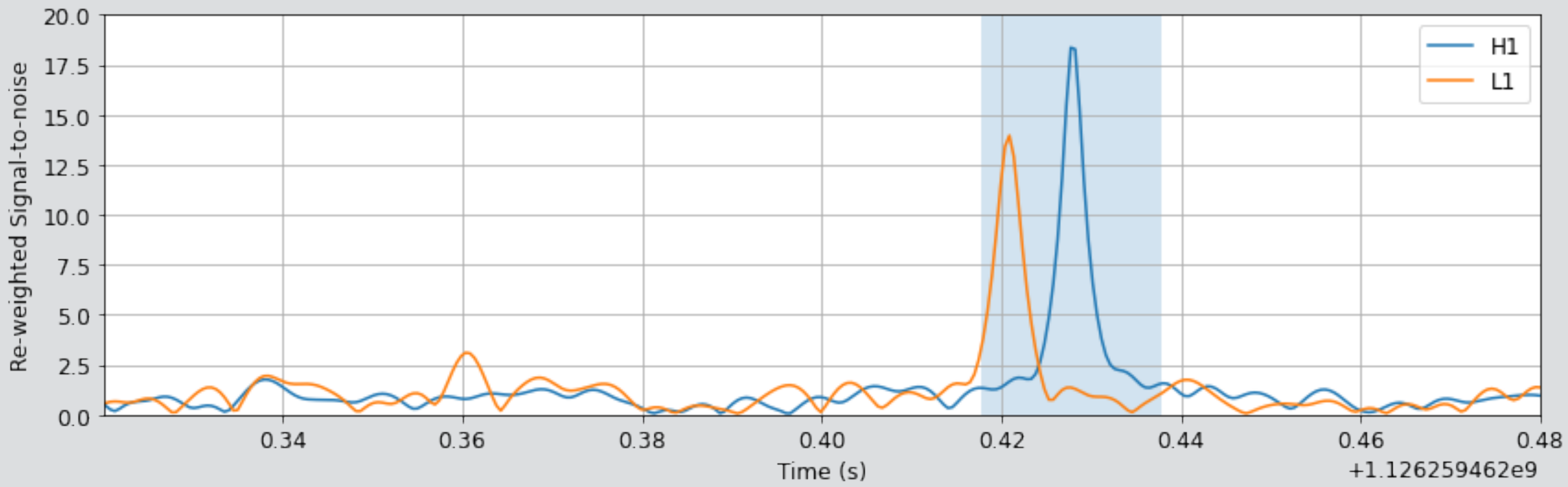


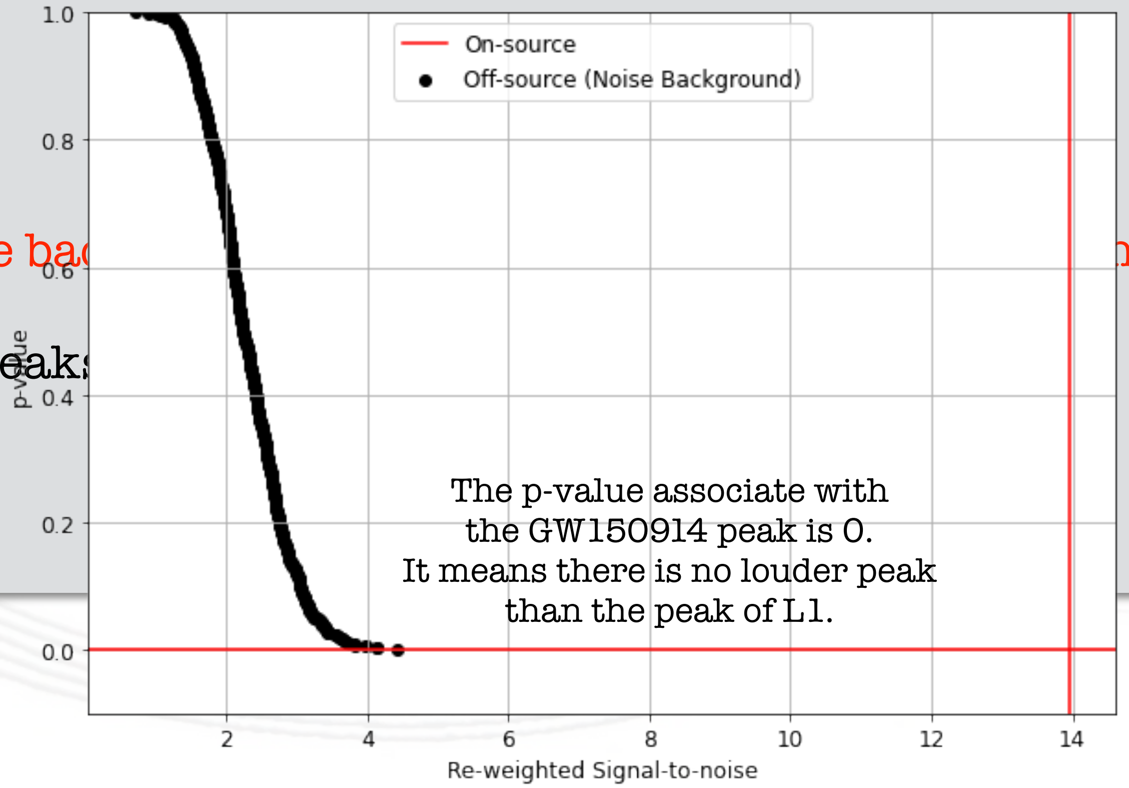
The p-value associate with the GW150914 peak is 0. It means there is no louder peak than the peak of L1.

Signal consistency and significance

- Now that we’ve calculated the on-source peak, we should calculate the background peak values.
  - We do this by chopping up the time series into chunks that are the same size as our on-source and repeating the same peak finding (max) procedure.

```
import numpy

peaks = []
i = 0
while i + window_size < len(snr[‘L1’]):
   p = snr[‘L1’][i:i+window_size].max()
   peaks.append(p)
   i += window_size

   # skip past the onsource time
   if abs(i - sidx) < window_size:
      i += window_size * 2
peaks = numpy.array(peaks)

# The p-value is just the number of samples observed in the ba                               he on-
source divided by the number of samples.
pcurve = numpy.arange(1, len(peaks)+1)[::-1] / float(len(peaks
peaks.sort()

pvalue = (peaks > onsource).sum() / float(len(peaks))
```

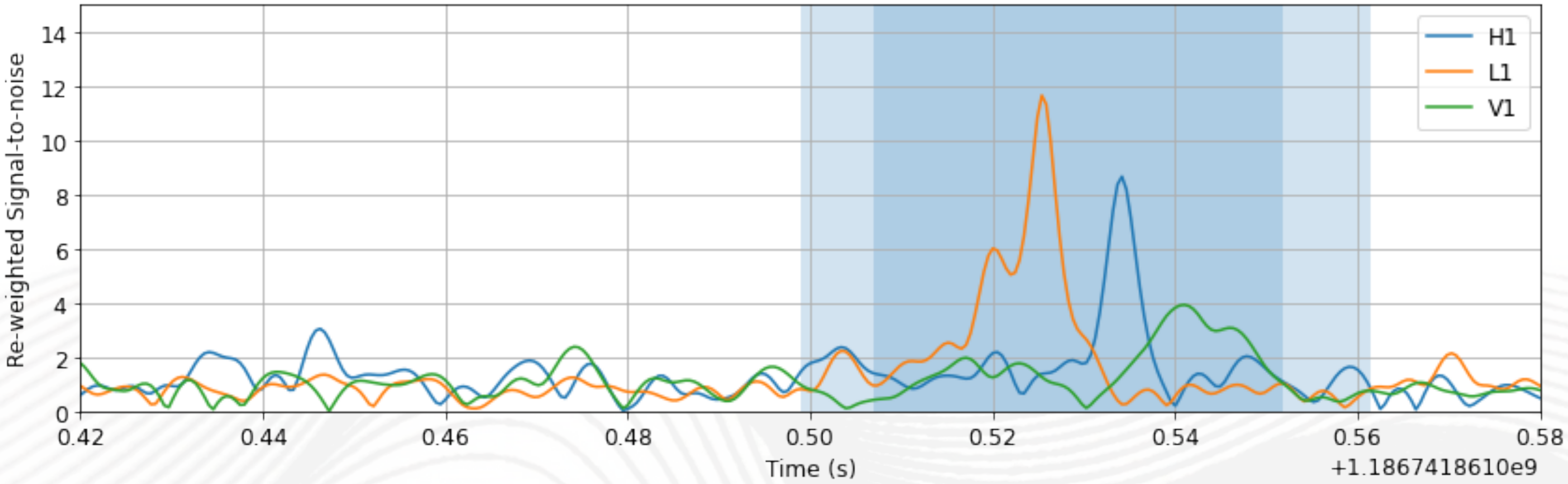


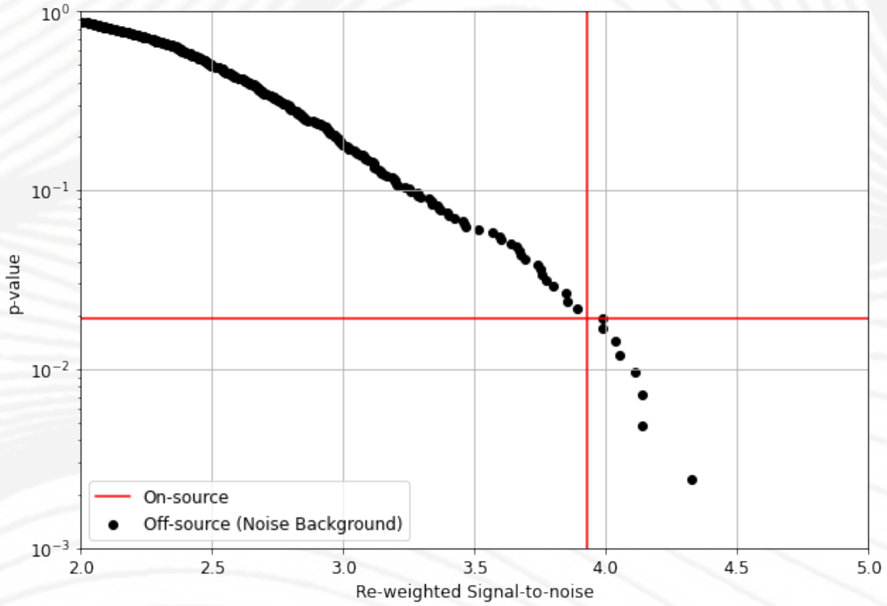


The p-value associate with the GW150914 peak is 0. It means there is no louder peak than the peak of L1.

Signal consistency and significance

- However, we may have $p > 0$ if a peak of any detector is not that much significant.
- Example: GW170814 observed by the LIGO observatories and Virgo





```
The p-value associate with the GW170814 peak of Virgo is
0.019277108433734494.
```

- We find a peak in Virgo as large as the observed one has an approximately 2% chance of occurring due to the noise alone.
- If $p < 0.05$, we may reject the null hypothesis that the observed peak is due to noise alone.

Summary

- We have demonstrated how to find a candidate GW signal from noisy data.

  (1) Estimating PSD from noisy data

  (2) Preparing template waveform

  (3) Whitening

  (4) Computing the cross-correlation (signal-to-noise ratio) between the template and the data

  (5) Testing consistency between the template and the data with χ^2 test

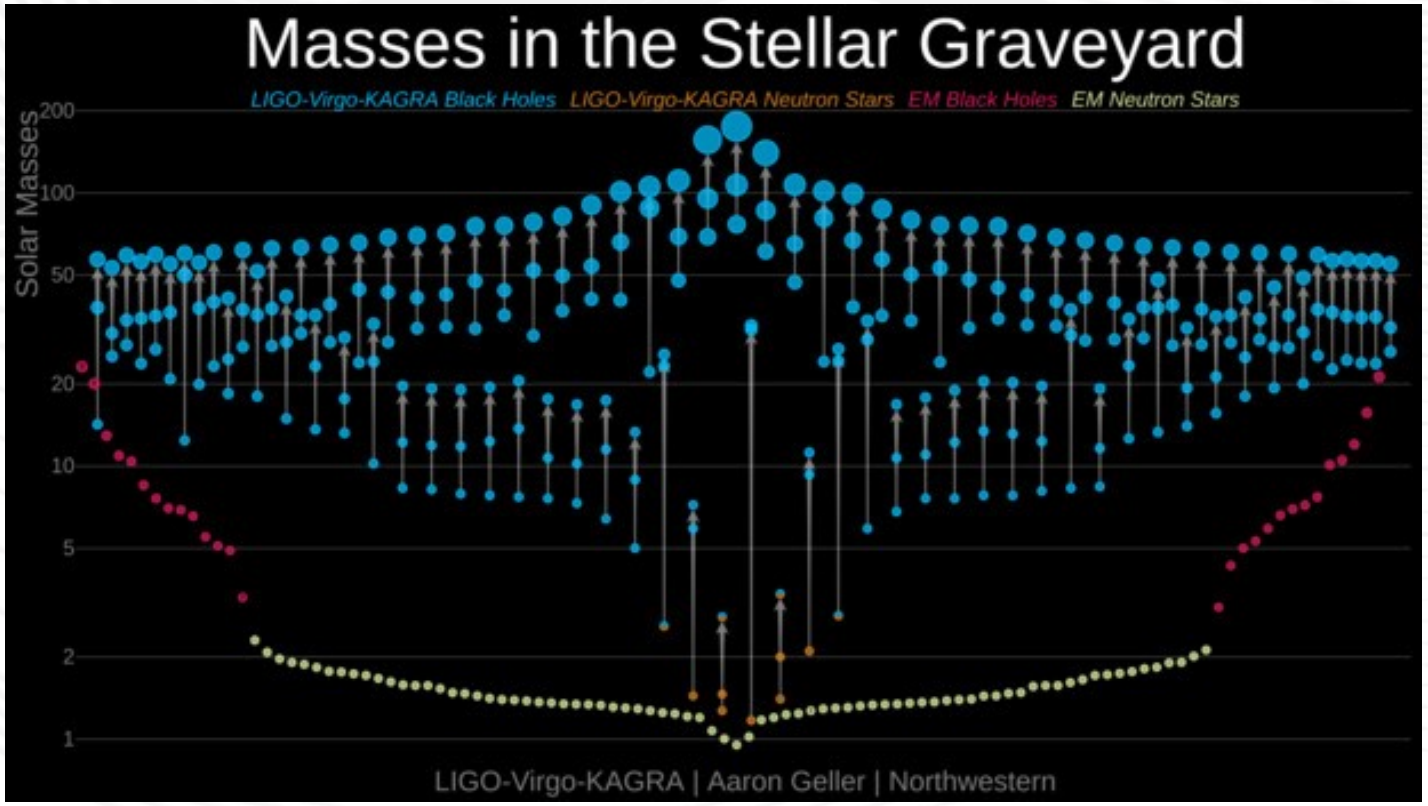
  (6) Evaluating significance with p-value estimation

Applications to Astrophysics

Applications to Astrophysics

- All observed events to date are compact binary coalescences as noted in Chunglee Kim's lecture.
  - Binary black holes
  - Binary neutron stars
  - Neutron star-black hole binaries

- Black hole
  - Non-zero black hole spins
    → existence of Kerr-type black holes
  - $M_{\rm BH} > 20 M_\odot$
- Neutron star
  - Equation of states (EoS)
    → constraint EoS models
  - Hubble constant measurement
    (cosmology)
- For more details, please refer
  Chunglee Kim's lecture on GW Astrophysics.



LIGO-Virgo-KAGRA | Aaron Geller | Northwestern

Applications to Astrophysics

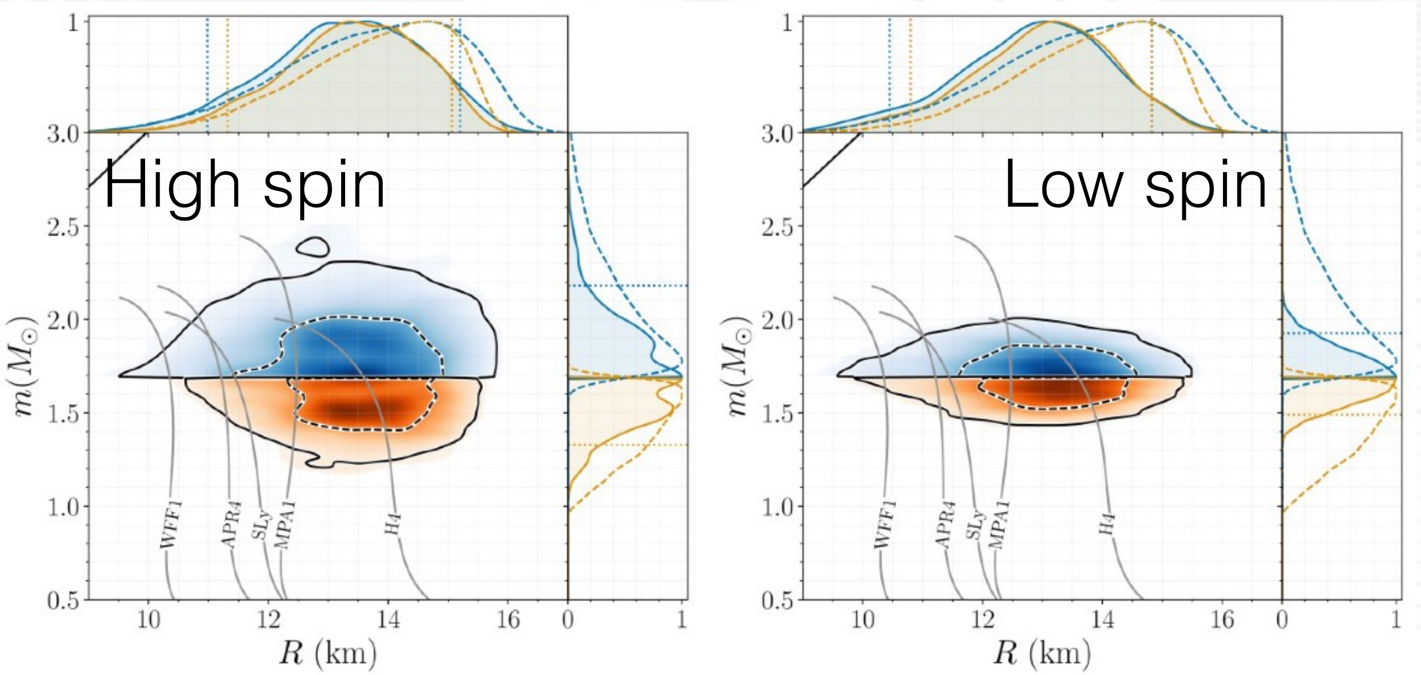
- All observed events to date are compact binary coalescences as noted in Chunglee Kim’s lecture.
    - Binary black holes
    - Binary neutron stars
    - Neutron star-black hole binaries

GW Detections

GW Astrophysics

- Black hole
    - Non-zero black hole spins
      → existence of Kerr-type black holes
    - $M_{\mathrm{BH}} > 20M_{\odot}$
- Neutron star
    - Equation of states (EoS)
      → constraint EoS models
    - Hubble constant measurement
      (cosmology)
- For more details, please refer
  Chunglee Kim’s lecture on GW Astrophysics.

GW190425



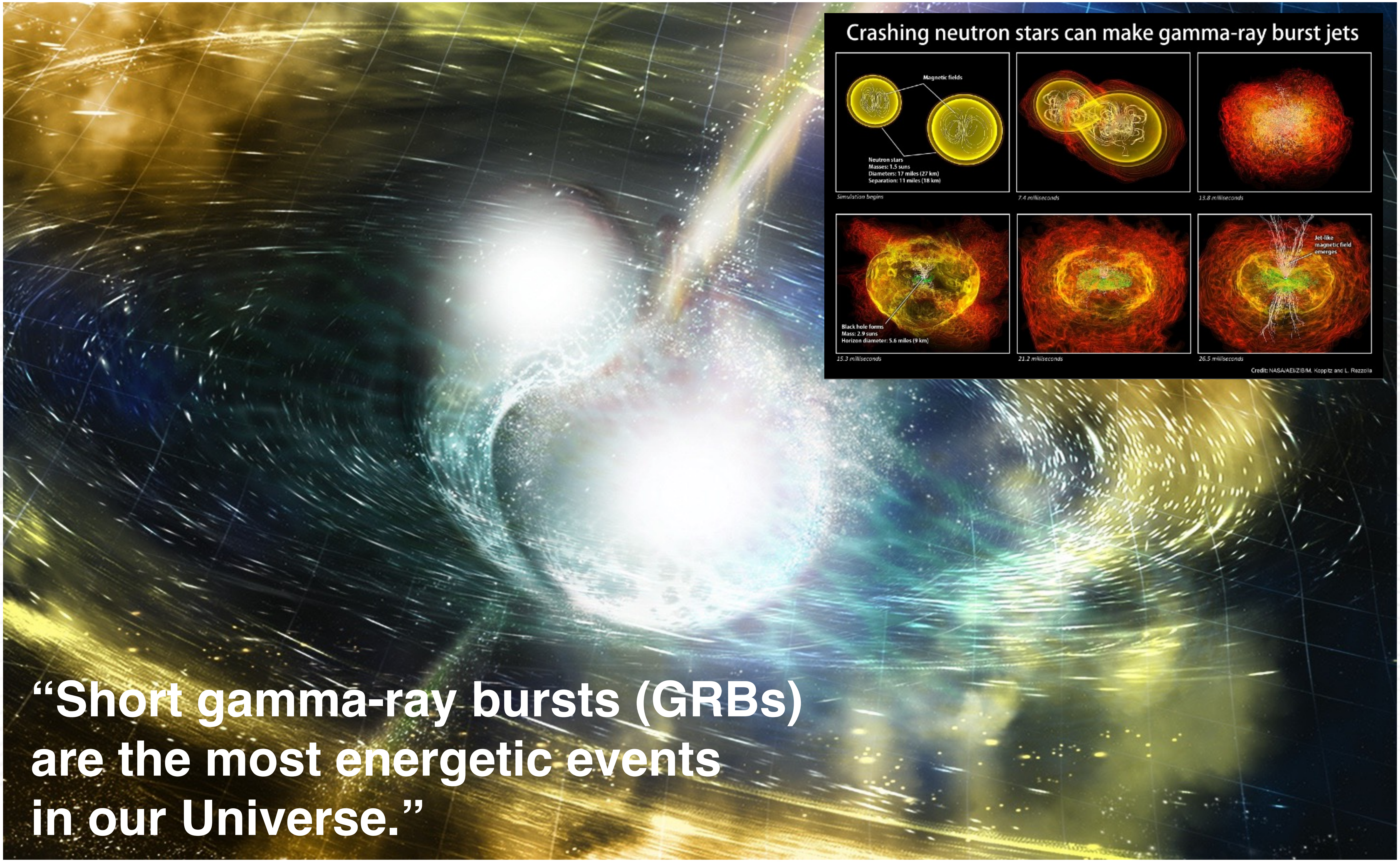
[Abbott+ (2020, ApJL)]

Applications to ML-based Astrophysics (Selected Examples)

ML for GW Search Related to Short GRBs

KK+, CQG **32** (2015) 24, 245002

- Motivation
  - Progenitors of short GRBs can radiate both GW and EM waves.



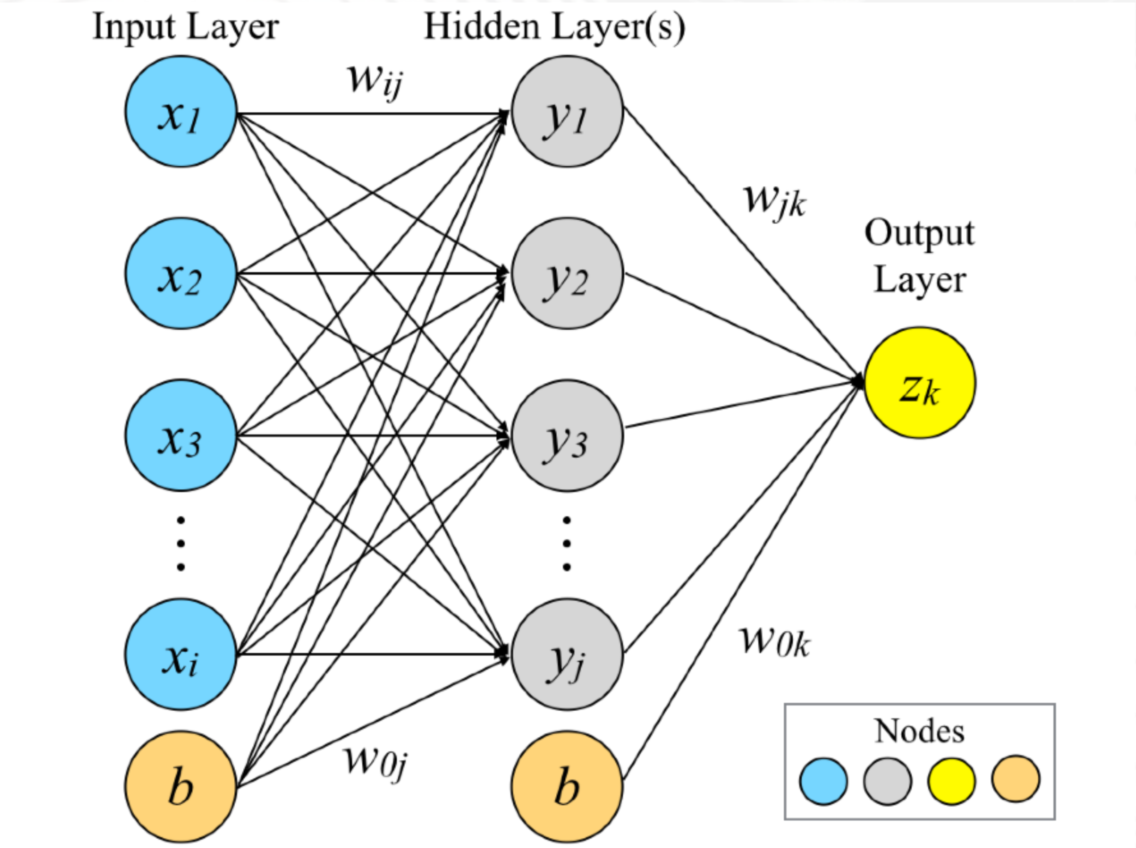
Credit: NASA/AEI/ZIB/M. Koppitz and L. Rezzolla

"Short gamma-ray bursts (GRBs) are the most energetic events in our Universe."

ML for GW Search Related to Short GRBs

KK+, CQG **32** (2015) 24, 245002

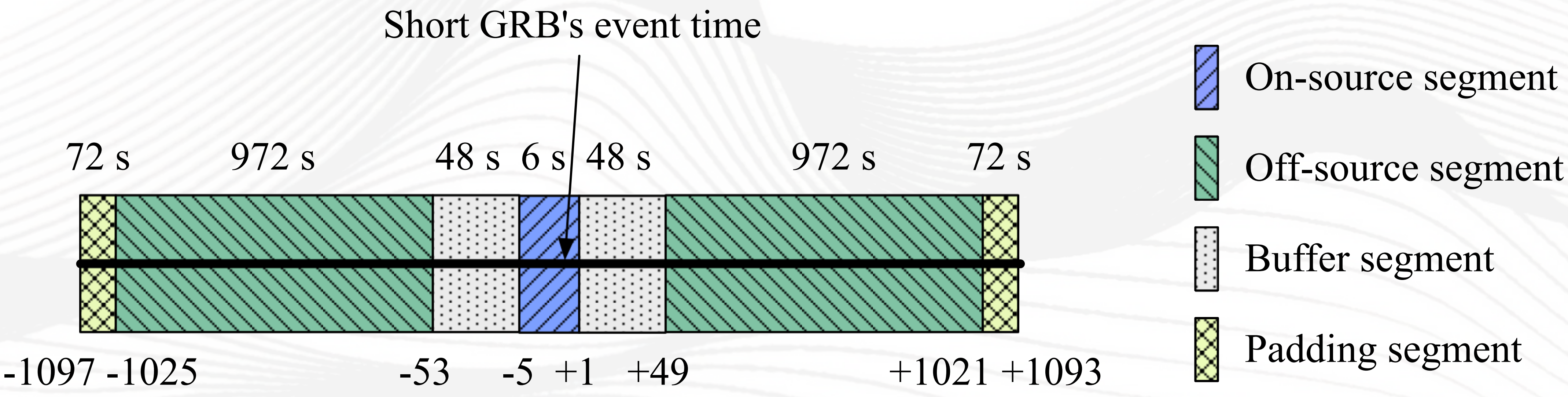
- Motivation
  - Progenitors of short GRBs can radiate both GW and EM waves.
    - proved by GW170817 and GRB170817 later on.
  - Previous searches for LIGO's S5 & S6 and Virgo's VSR1, VSR2, & VSR3 data couldn't find any evidence from the candidate triggers (events) evaluated by a ranking statistics of a matched-filtering-based search method (Abadie+ (2010, 2012); Aasi+ (2014)).
  - Neural networks can be a new ranking method for candidate events.



ML for GW Search Related to Short GRBs

KK+, CQG **32** (2015) 24, 245002

- Date preparation
  - We use some triggers generated by the existing analysis pipeline which produces
    - on-source triggers: regarded as containing a candidate GW signal
    - off-source triggers: estimating background distribution around the candidate
    - software injection triggers: evaluating the performance of the search pipeline



  - We use the software injection triggers as signal samples and the off-source triggers as background samples.
    - software injection: considering both BNS and NSBH systems

ML for GW Search Related to Short GRBs

KK+, CQG **32** (2015) 24, 245002

For both neutron star binary (BNS) and neutron star - black hole binary (NSBH)…

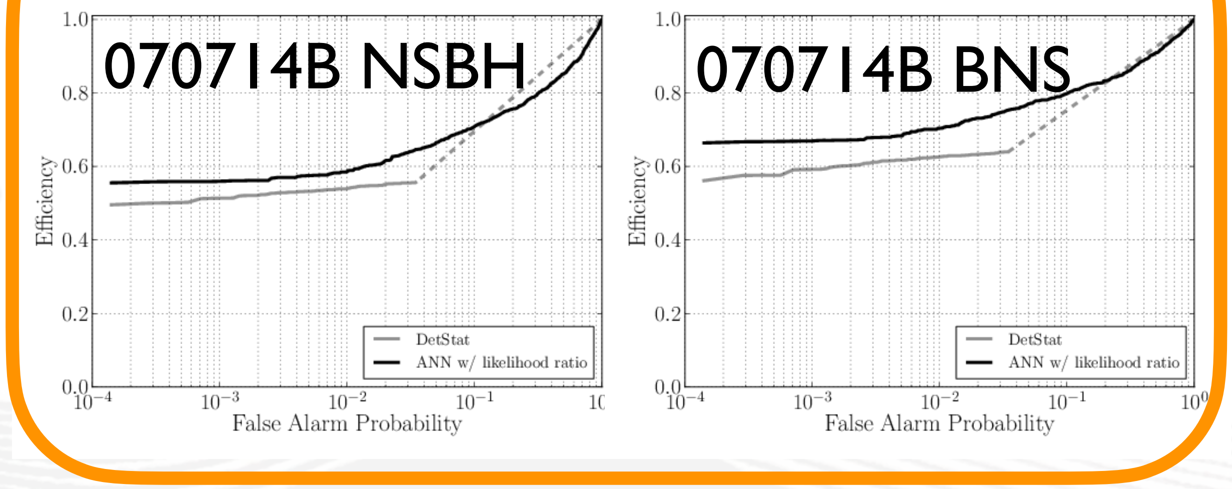
Signal samples (~2 000 samples) / Background samples (~7 000 samples)
+
10 Feature Parameters from CBC-GRB triggers

- Single IFO's SNRs
- Coherent SNR, New SNR
- Coherent χ^2-test, bank χ^2-test, auto-correlation χ^2-test value
- Mass 1 and Mass 2 of BNS or NSBH

with two S5 & VSR1 triple-coincidence data (070714B & 070923)

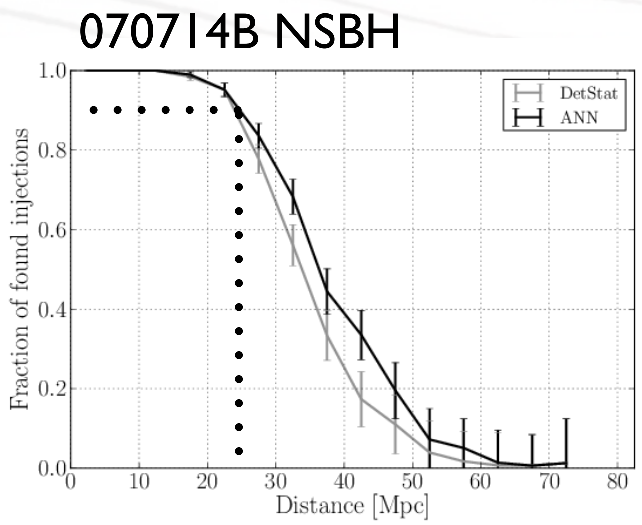
Classification with Neural Network as post-processing

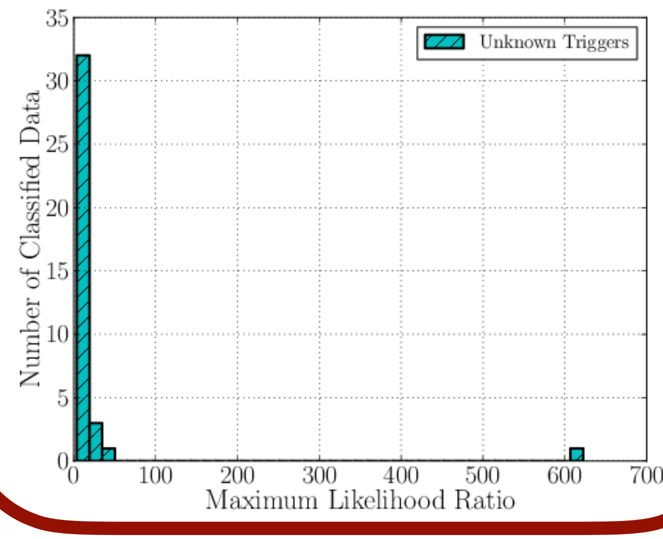
~5% — 10% improved efficiency



Sensitivity

Evaluating Unknown Triggers

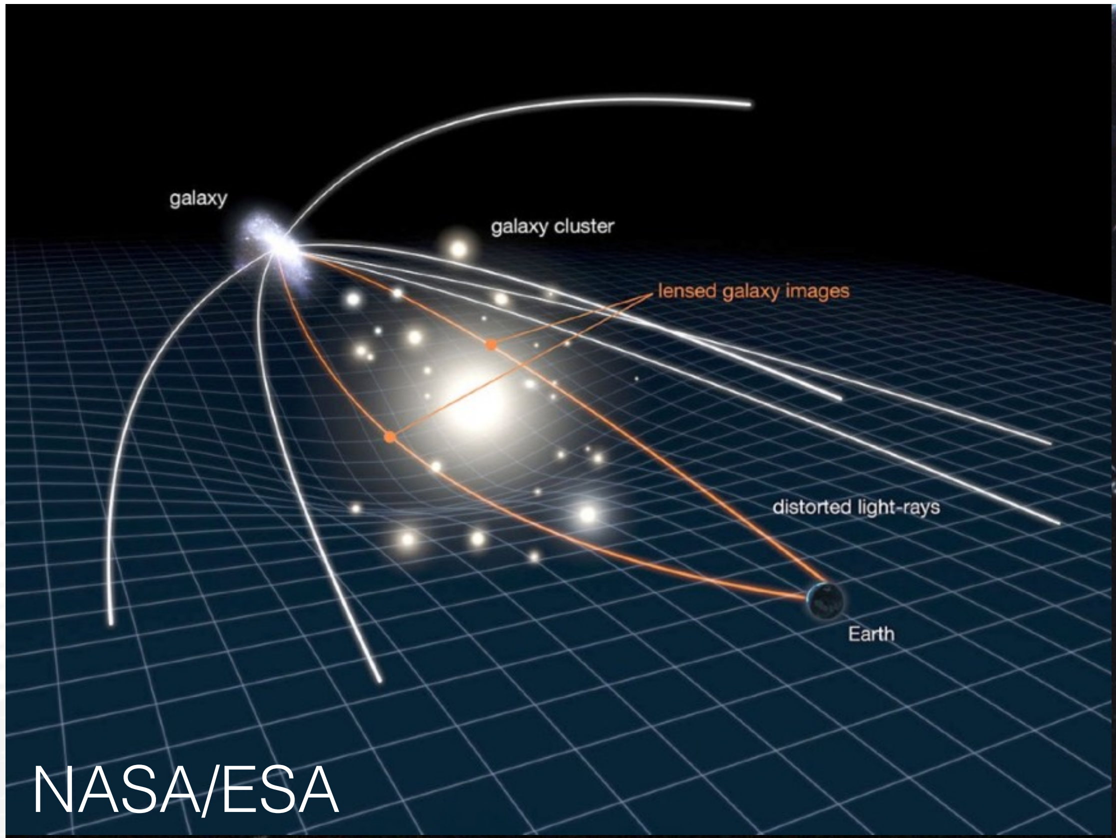




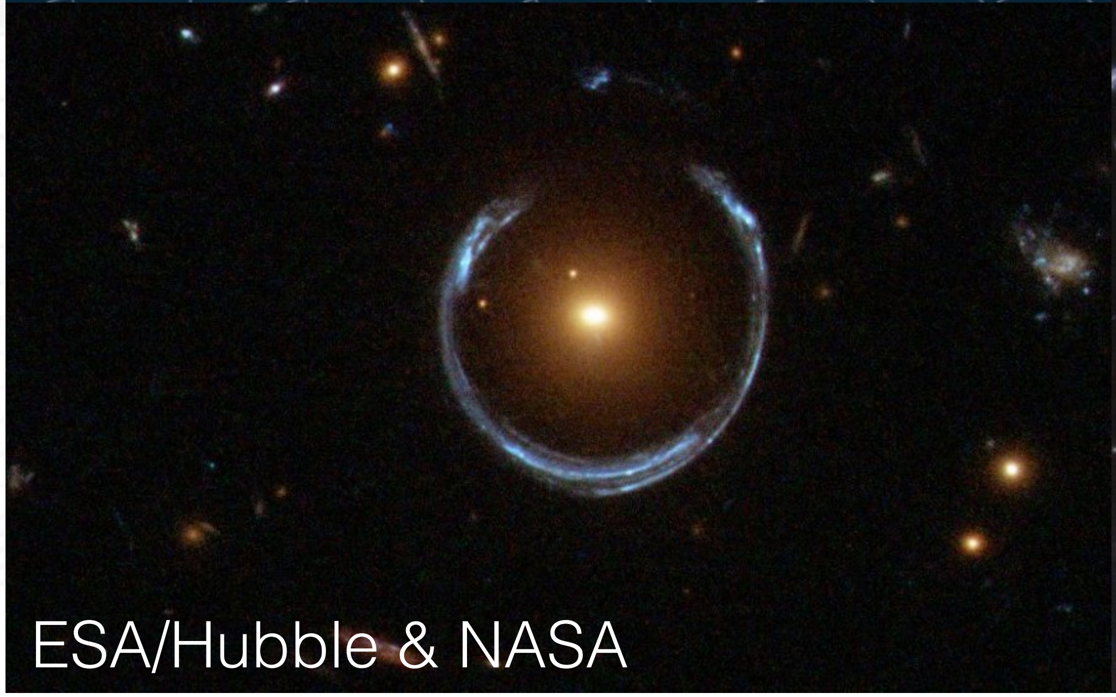
ML for Identification of Lensed GWs

KK+, ApJ **915** (2021) 2, 119

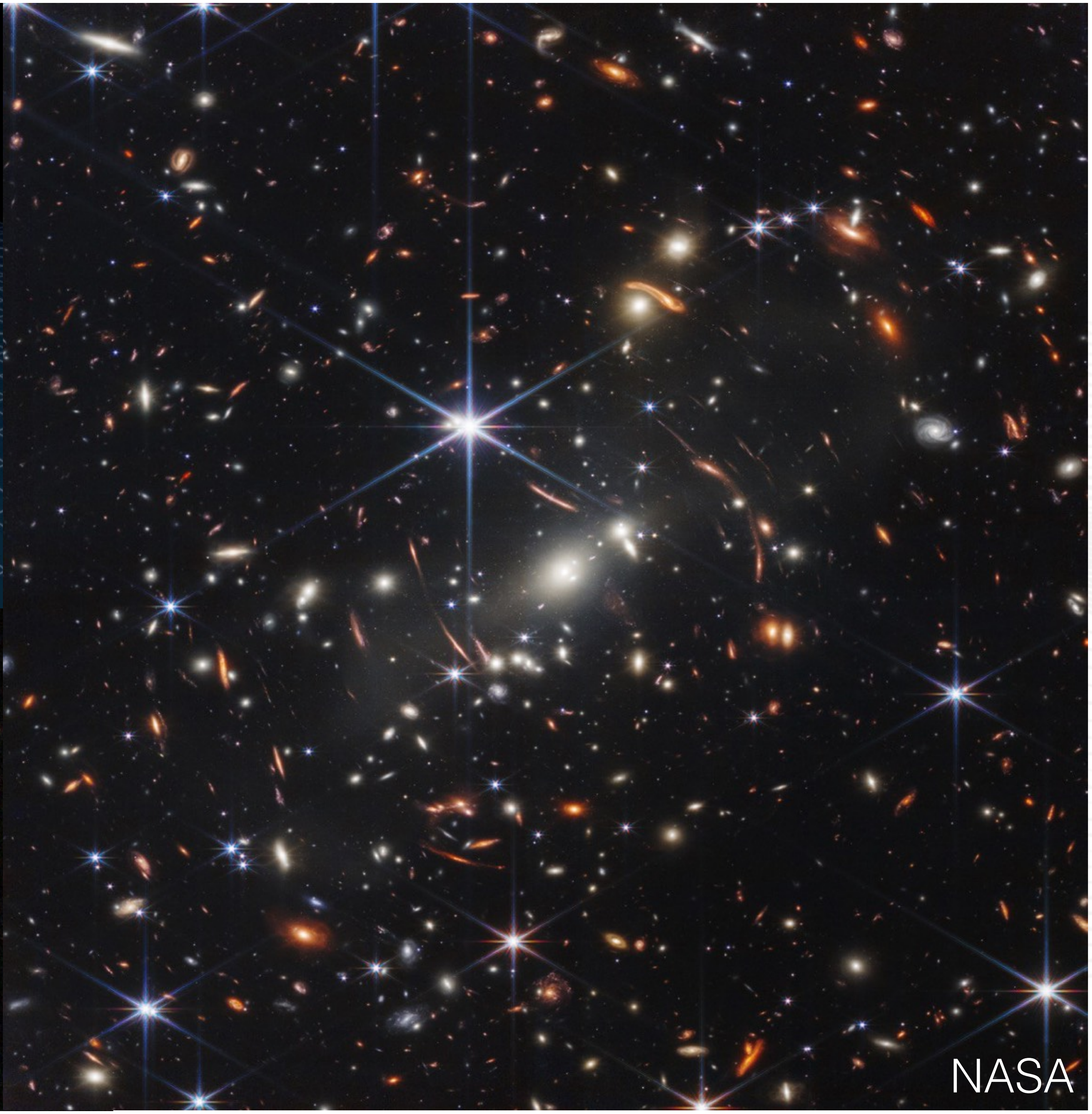
- Motivation
  - If GWs propagate around heavy mass systems, they can be lensed like EM waves.



NASA/ESA



ESA/Hubble & NASA

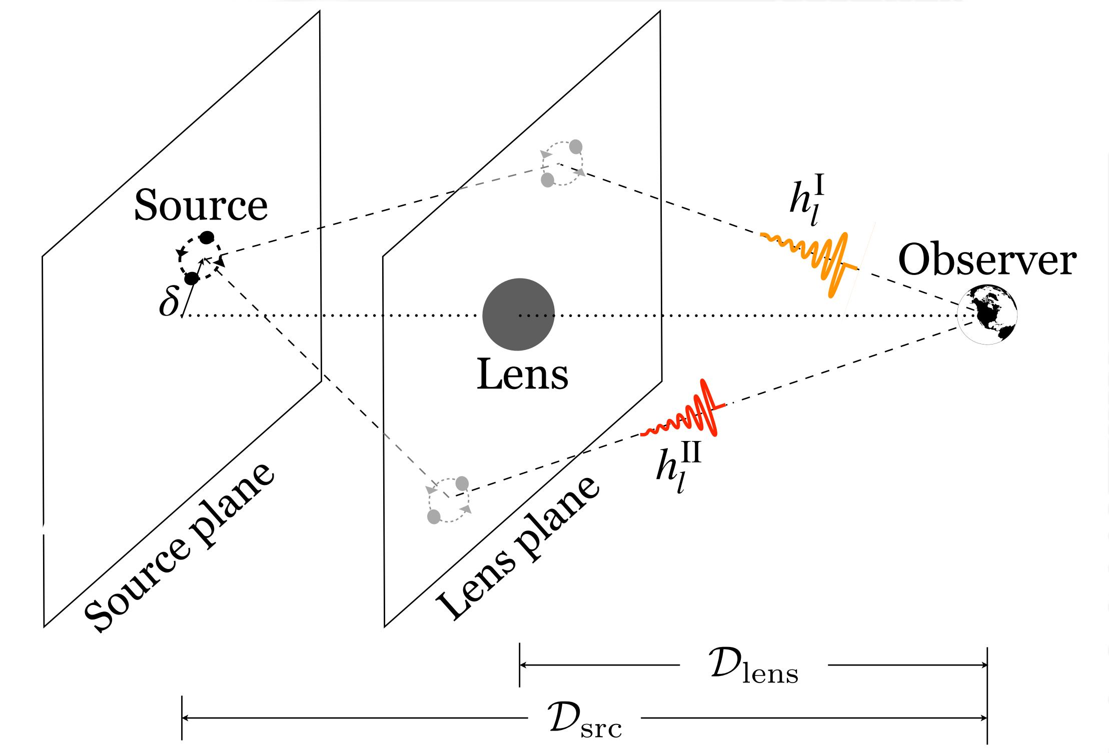


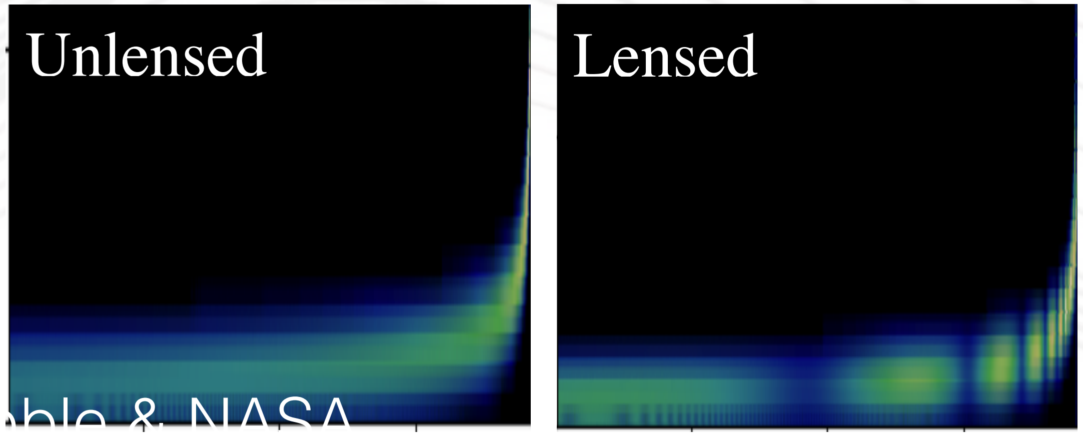
NASA

ML for Identification of Lensed GWs

KK+, ApJ **915** (2021) 2, 119

- Motivation
  - If GWs propagate around heavy mass systems, they can be lensed like EM waves.
  - If the time delay of two lensed images is short enough (~ms), the images would be superposed.

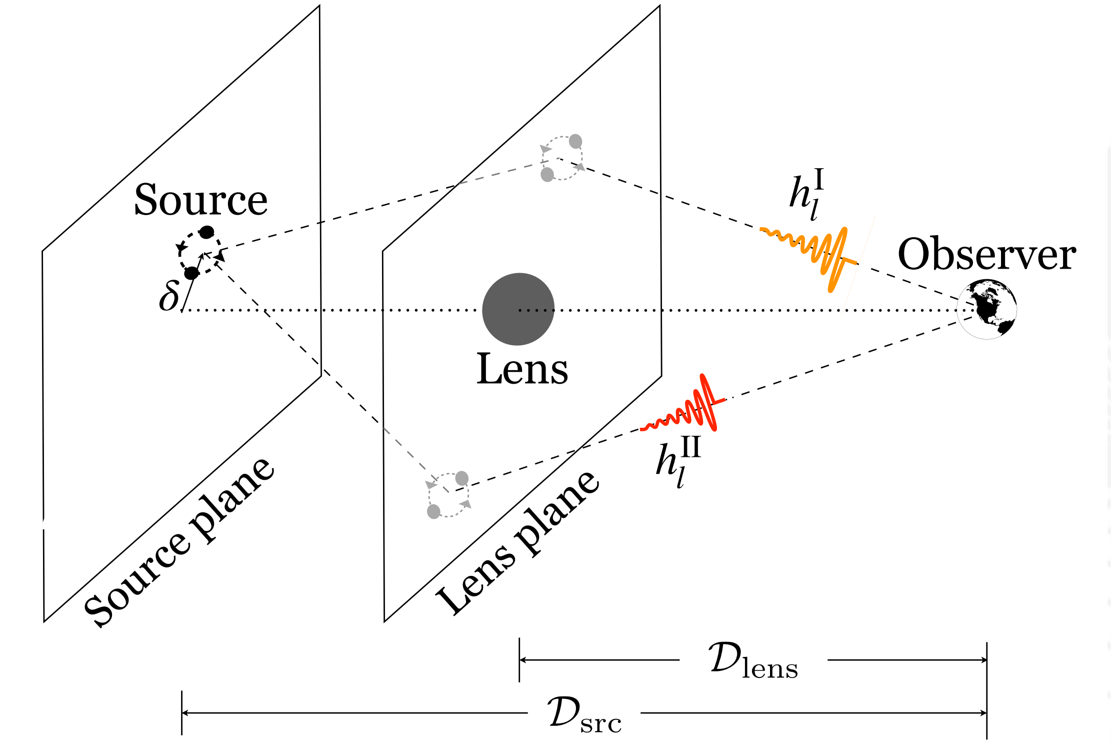


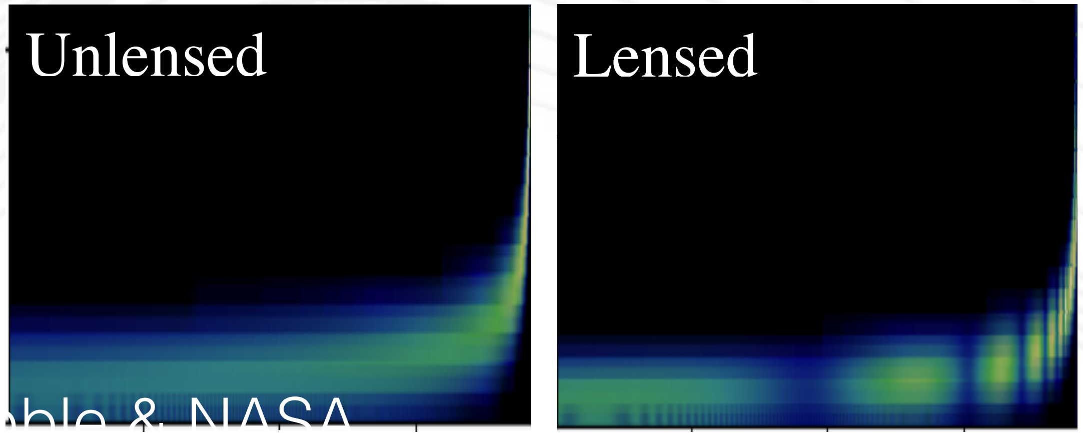


ML for Identification of Lensed GWs

KK+, ApJ **915** (2021) 2, 119

- Motivation
  - If GWs propagate around heavy mass systems, they can be lensed like EM waves.
  - If the time delay of two lensed images is short enough (~ms), the images would be superposed.





- Thin lens approximation
- Strain amplitude of lensed GW in frequency domain

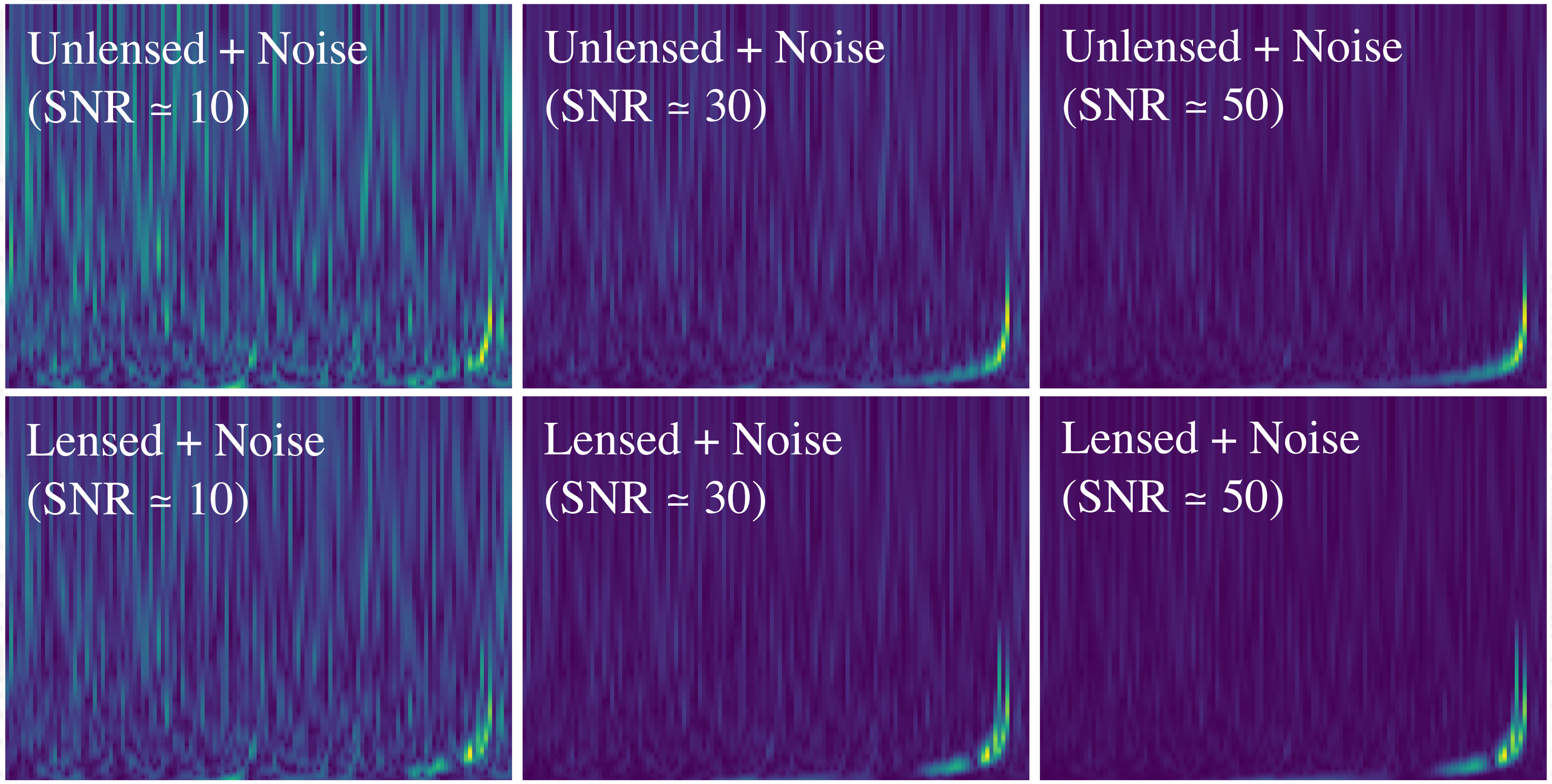
$$h_L(f) = F(f)h(f)$$

where $F(f)$ is the ***amplification factor*** which determines the lensing signatures, e.g., magnification of lensed signals and time delays between them.

ML for Identification of Lensed GWs

KK+, ApJ **915** (2021) 2, 119

- Input data: spectrogram using IMRPhenomPv2 and constant-Q transform
  - unlensed+non-precessing (U_N),
unlensed+precessing (U_P), and
lensed+non-precessing (L)
  - Poin Mass model and Singular Isothermal Sphere model
  - Parameters
    - m_1, m_2: $5 - 55 M_\odot$
    - D_L: $10 - 1000$Mpc
    - D_{LS}: $10 - 1000$Mpc
    - M_L: $10^3 - 10^5 M_\odot$
    - δ: $10^{-6} - 0.5$pc
  - Noise: aLIGO's DetHighPower model
    - $10 \leq$ SNR ≤ 50
(c.f. ≤ 23.6 for BBHs in GWTC-1)
  - # of samples: 45,000 for each type and each lens model
    - training (80%), validation (10%), and evaluation (10%)



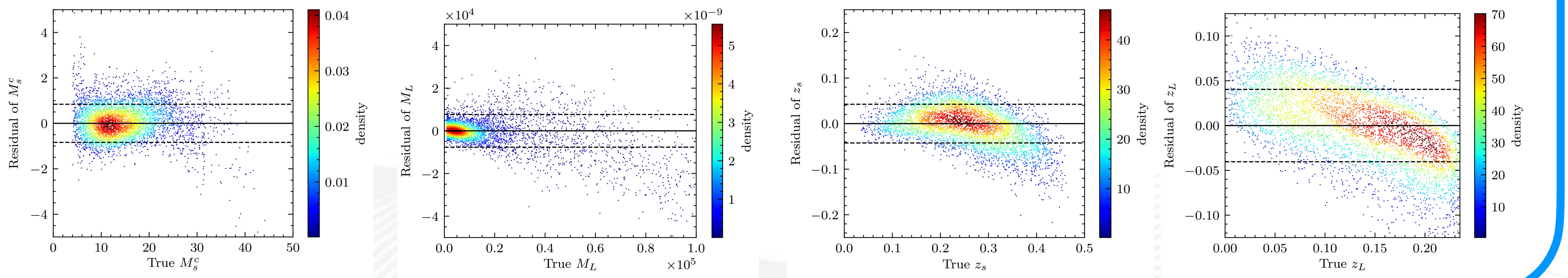
↖ $m_1 = m_2 = 20 M_\odot$; $M_L = 10^4 M_\odot$
$D_S = 1$Gpc; $D_L = 800$Mpc

ML for Identification of Lensed GWs

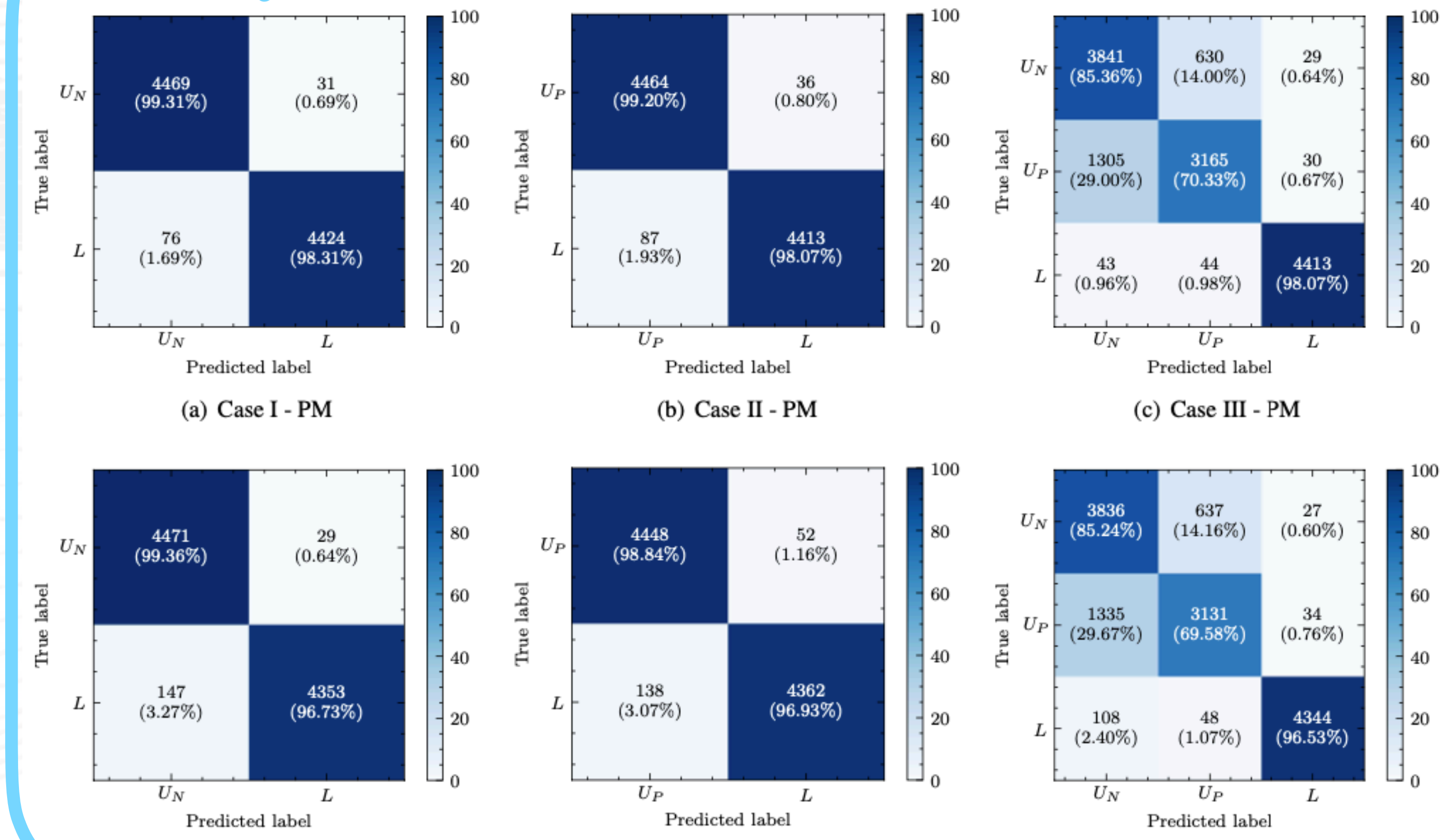
KK+, ApJ **915** (2021) 2, 119

Regression for Parameter Estimation

Chirp mass of source Lens mass Redshift of source Redshift of Lens



Classification



(a) Case I - PM

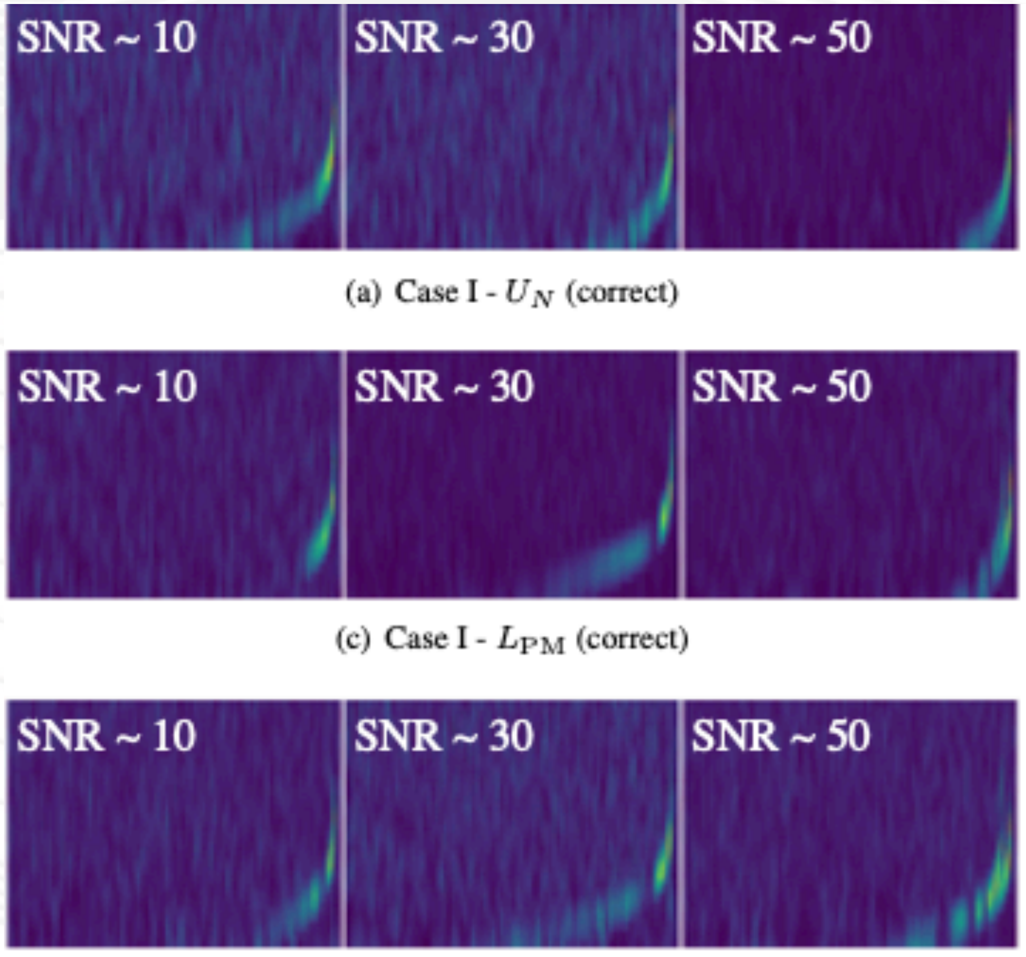
(b) Case II - PM

(c) Case III - PM

(d) Case I - SIS

(e) Case II - SIS

(f) Case III - SIS



(a) Case I - U_N (correct)

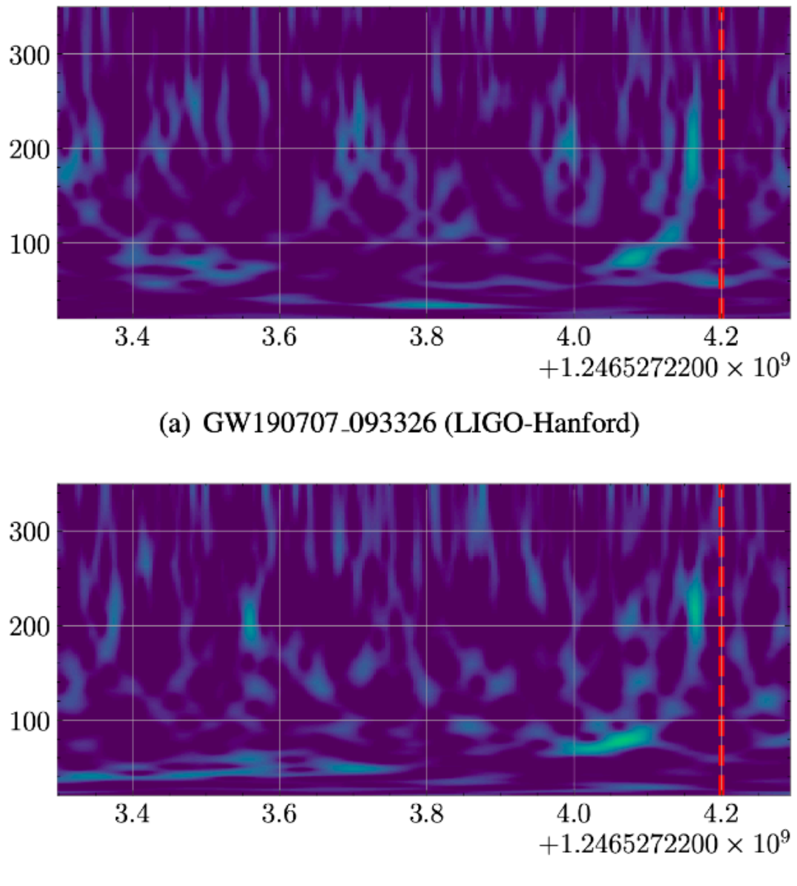
(c) Case I - $L_{\rm PM}$ (correct)

(e) Case I - $L_{\rm SIS}$ (correct)

Search for Lensed GWs in GWTC-1 and -2

KK+, arXiv:2206.08234

- Search for beating patterns might be occurred by lenses with masses of $10^3 - 10^5 M_\odot$ from spectrograms of BBH signals with deep learning (DL)
  - Revisit the BBH events already analyzed in previous lensing papers [Hannuksela+ (2019) & Abbott+ (2021)]
  - Use public open data from GWOSC
- DL-based method for searching such beating patterns [Kim+ (2021)]
- From the primary classification, only GW190707_093326 is classified as **lensed** out of 46 events.
  - No visually identifiable beating patterns
  - $r_\mathrm{L} = 0.984^{+0.012}_{-0.342}$ with 90% C.I. (from bootstrapping)
    - $0 \lesssim p \lesssim 0.1$
- The uncertainty of p contains where $p > 0.05$ that accepting the unlensed hypothesis being true.
  - c.f., $\mathscr{B}^{\mathrm{ML}}_{\mathrm{U}}$ = -0.4 disfavoring lensed hypothesis [Abbott+ (2021)]
- Conclude the GW signal of this event is likely an unlensed one.



(a) GW190707_093326 (LIGO-Hanford)

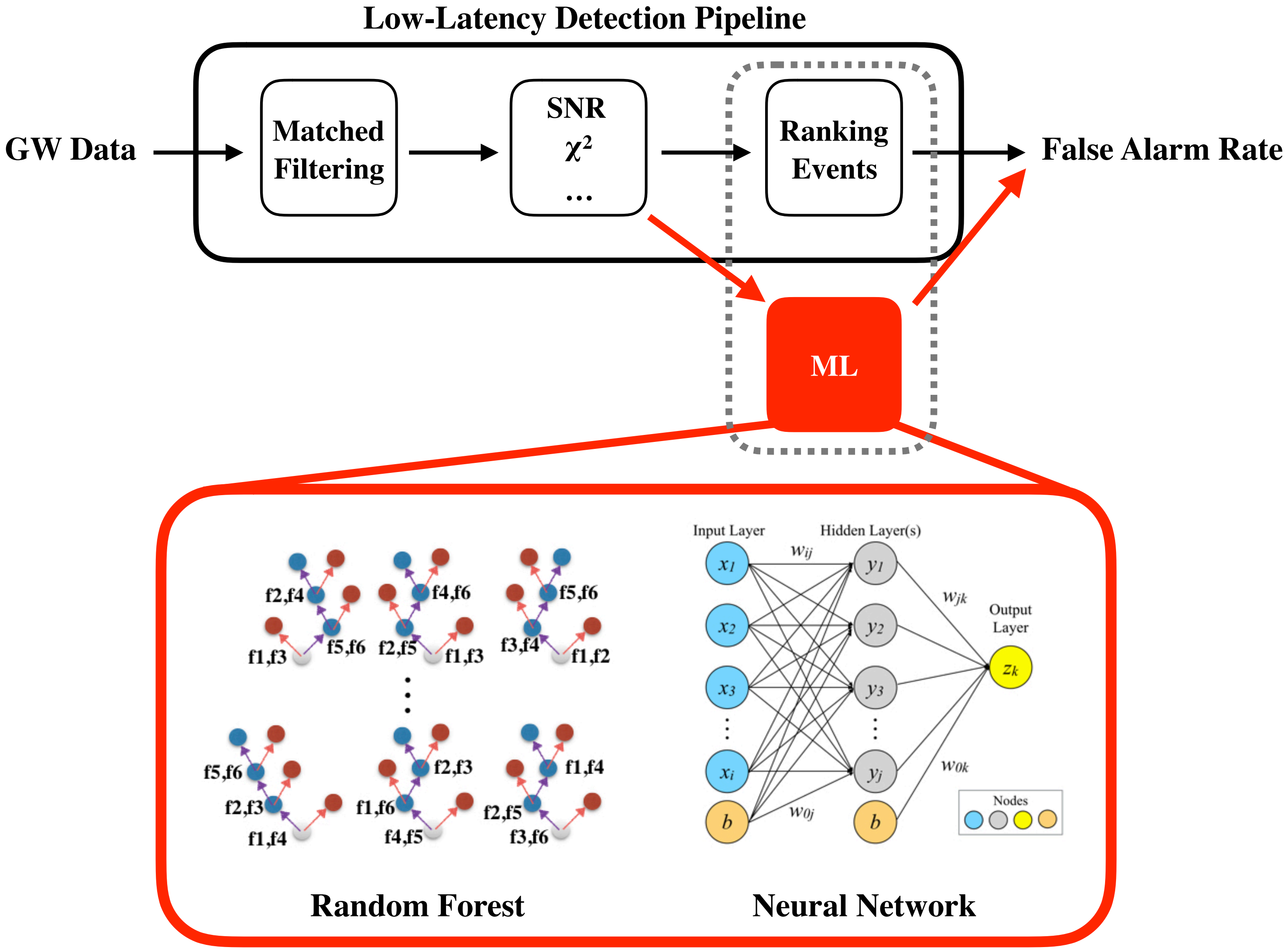
(b) GW190707_093326 (LIGO-Livingston)

ML for Low-Latency GW Search

KK+, Phys. Rev. D **101** (2020) 8, 083006

- Motivation
  - Low-latency search (detection) pipeline: real-time (online) search pipeline which produce candidate event triggers within $\mathcal{O}$(min).
    - c.f., offline search takes $\mathcal{O}$(hrs) – $\mathcal{O}$(days)
    - GstLAL inspiral pipeline (Messick+ '17)

  - Similar to the previous work, we assume the output of machine learning algorithms can be used to rank candidate events of low-latency pipeline.
    - In this work, we consider random forest and neural networks.



ML for Low-Latency GW Search

KK+, Phys. Rev. D **101** (2020) 8, 083006

Input Data

- Signal samples: mock data of GW150914 using GstLAL inspiral pipeline (~ 5 000 samples)
- Background samples: time-slide data around the GPS times of injections of the MDC (~ 172 000 samples)
- Features: mass1, mass2, spin1z, spin2z, snr, and chisq (6 features)
- Train/Test data: 75%/25% of shuffled samples (no validation data)

Training

- Time for training (w/ ~ 122 000 samples of 6 features) on MacBook Pro
  - Random Forest (scikit-learn): **~ 6—7 hrs** for running GridSearchCV with 288 combinations
  - Neural Network (TensorFlow): **~ 7—10 mins**

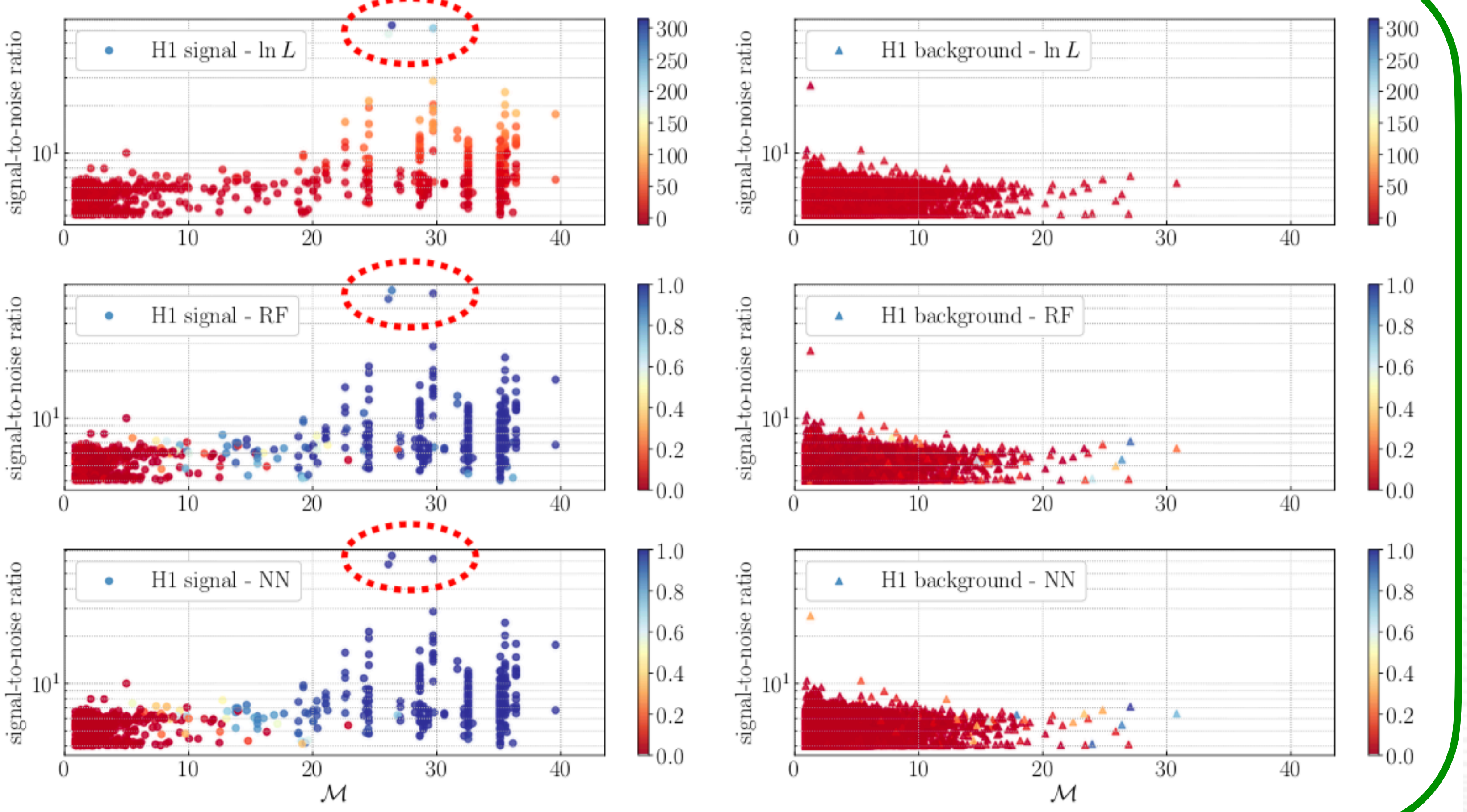
Evaluation

- Time for evaluation (w/ ~ 45 000 samples of 6 features): **~ O(100) ms**
- Output: probabilistic prediction between 0 and 1 → **rank**
- For the performance test of the evaluation result, 3 figure-of-merits were used:
  - Confusion matrix,
  - 2-D histogram: ln L vs. rank of ML,
  - Receiver Operation Characteristic (ROC) curve.

ML for Low-Latency GW Search

KK+, Phys. Rev. D **101** (2020) 8, 083006

Performance Test on Classification



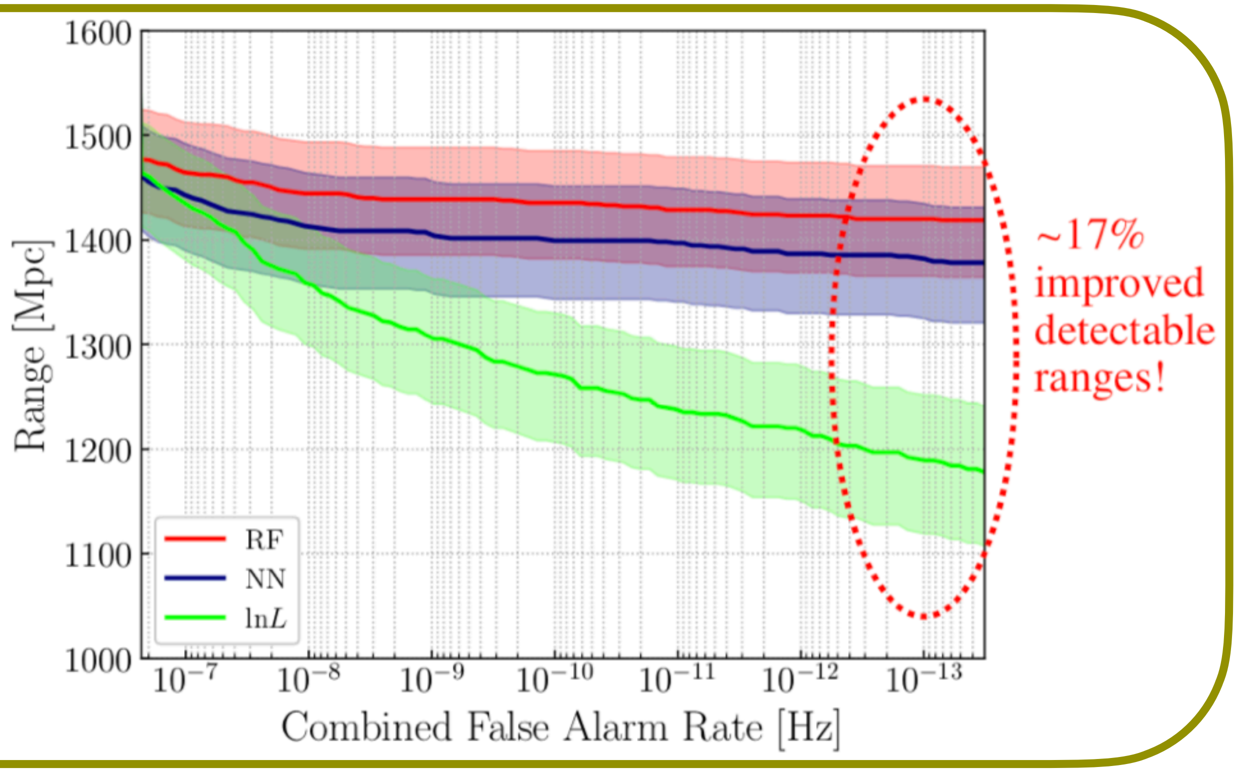
Remarks

- MLAs found high ranks candidate signals of GstLAL pipeline as well.
- MLAs found more candidates signals of lower signal-to-noise ratios than GstLAL pipeline.
- Similar performance on identifying noise samples.

Sensitivity in Detection Range

Remarks

- MLAs could capture more candidate signals generated from sources at farther distances at lower false alarm rate than GstLAL pipeline.

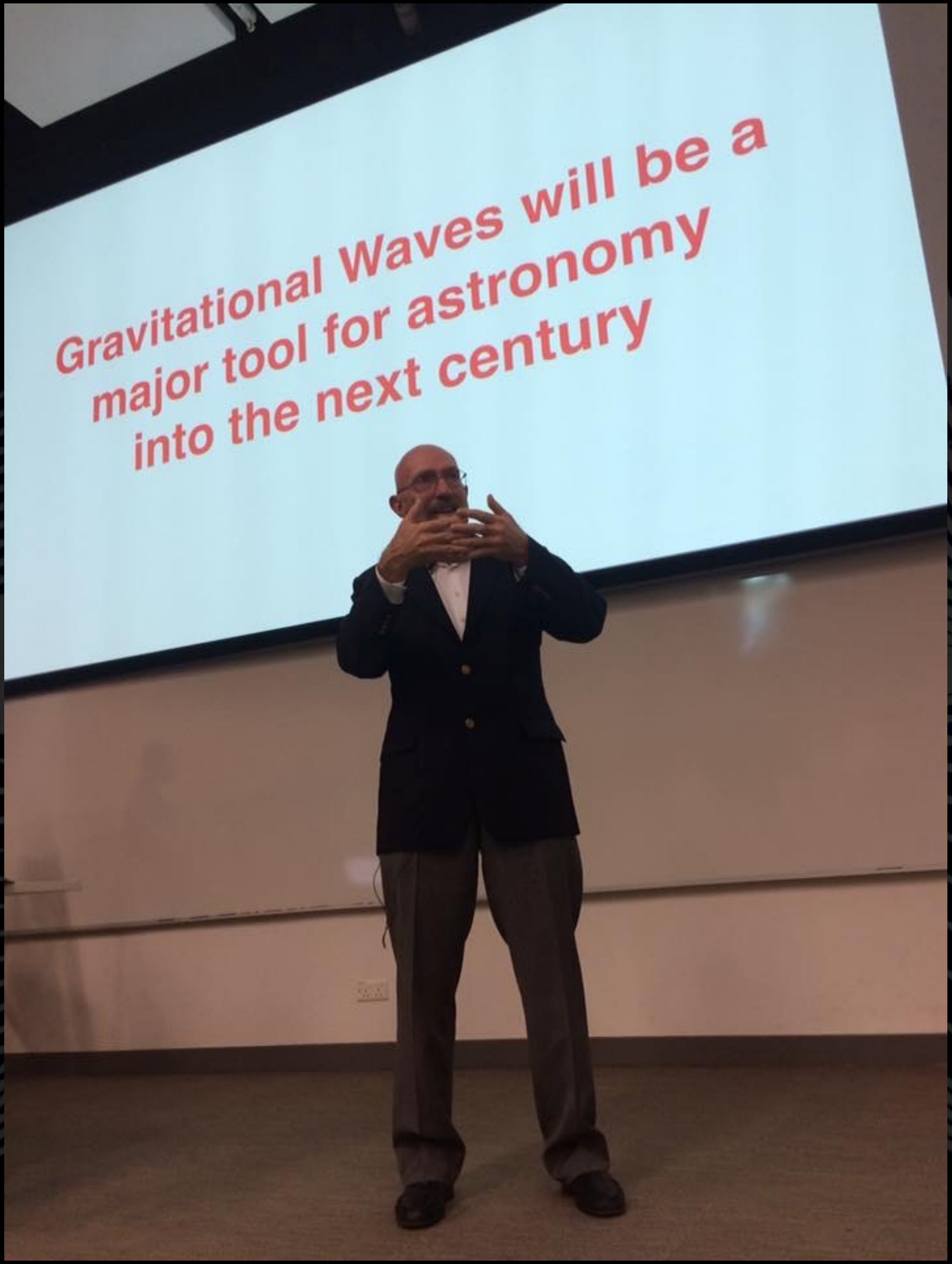


Summary

- We can do “astrophysics” with GW signals.
    - We can understand more about the nature of astrophysical compact objects such as black holes and neutron stars.
    - We may confirm the physics known from EM observations more robustly.
    - We may find new physics which haven’t seen from EM observations.

- Machine learning can be a useful tool for doing GW astrophysics.
    - It enables us to identify various astrophysical phenomena.
    - It can enhances the performance of data analysis.
        - ➡ increase the number of detections/observations.
        - ➡ help to understand those phenomena more deeper.

Kip Thorne said…



"Gravitational Waves will be a major tool for astronomy into the next century."

September 30, 2016
Public lecture @ CUHK, Hong Kong

Thank you!