PH 821 Course Project
Detecting a simulated gravitational-wave signal

Dhananjay Raman
210050044

November 25, 2024

Contents

1 Abstract

Gravitational wave detection has revolutionized our understanding of the cosmos, offering a unique probe into compact binary mergers such as binary black hole (BBH), neutron star-black hole (NSBH), and binary neutron star (BNS) mergers. This study details the implementation of a pipeline to identify and analyze simulated gravitational wave signals using the PyCBC toolkit. The signal was sourced from a simulated strain data file, processed through various steps including high-pass filtering, whitening, and power spectral density estimation to enhance signal clarity.

A coarse-to-fine grid search was conducted in the mass parameter space (10-100 $M_\odot$), leveraging matched filtering techniques to identify the optimal masses of the binary black hole system. The resulting signal exhibited a strong signal-to-noise ratio (SNR), with a peak observed at $M_1 = 10.1 M_\odot$ and $M_2 = 9.4 M_\odot$ (values based on results). A template waveform aligned with the detected signal was used to compute the residual data by subtracting the template from the conditioned signal. The time-frequency characteristics of the original and subtracted data were compared to confirm the successful detection and subtraction of the gravitational wave signal.

This pipeline demonstrates the power of matched filtering in detecting weak astrophysical signals buried in noise and provides a robust framework for parameter estimation of compact binary mergers. Future extensions of this analysis could include exploring other higher-order waveform models and expanding the parameter space to include spins and eccentricity.

All the code used in this analysis is available in the appendix and on GitHub.

2 Methods

This analysis was performed using the PyCBC library, a Python package tailored for gravitational wave data analysis. The methods employed in this analysis were adapted from publicly available tutorials on gravitational wave data analysis [1, 2].

2.1 Data Preprocessing

The strain data from the Virgo detector was read from a .gwf file. To improve the data quality:

- Low-frequency content was removed using a high-pass filter with a cutoff of 50 Hz.
- The data was downsampled to a uniform sampling rate of 2048 Hz.
- The first and last 2 seconds of the data were cropped to mitigate edge effects.

2.2 Power Spectral Density (PSD) Estimation

The PSD of the data was estimated using the Welch method over 4-second segments. The PSD was then interpolated to match the data's frequency resolution and truncated using the inverse spectrum truncation method to prevent noise amplification.

2.3 Whitening and Bandpassing

The strain data was whitened by dividing its Fourier transform by the square root of the PSD. Subsequently, the whitened data was bandpassed between 50 Hz and 512 Hz to isolate the relevant frequency range of the signal.

2.4 Time-Frequency Analysis

A Q-transform was applied to the whitened data to generate a time-frequency representation, which highlights transient features. Specific time and frequency windows were zoomed to focus on the merger signal.

2.5 Matched Filtering

Matched filtering was employed to search for compact binary coalescence signals (assuming non-spinning binary black holes) in the data. The following steps were performed:

- A coarse grid of masses was explored using the SEOBNRv4 waveform approximant.
- The signal-to-noise ratio (SNR) was computed for each template, and the highest SNR was identified to determine the most likely binary parameters.
- A finer grid of masses based on the best pair of masses from the coarse search refined the parameter estimation.

2.6 Template Generation and Signal Alignment

The optimal waveform template was generated using the SEOBNRv4 approximant. The template was aligned with the data using the peak SNR time and scaled to match the detected amplitude and phase.

2.7 Signal Subtraction

The aligned template was subtracted from the conditioned data to evaluate the residual signal. The time-frequency representations of the original and subtracted data were compared to assess the effectiveness of the subtraction.

3 Results

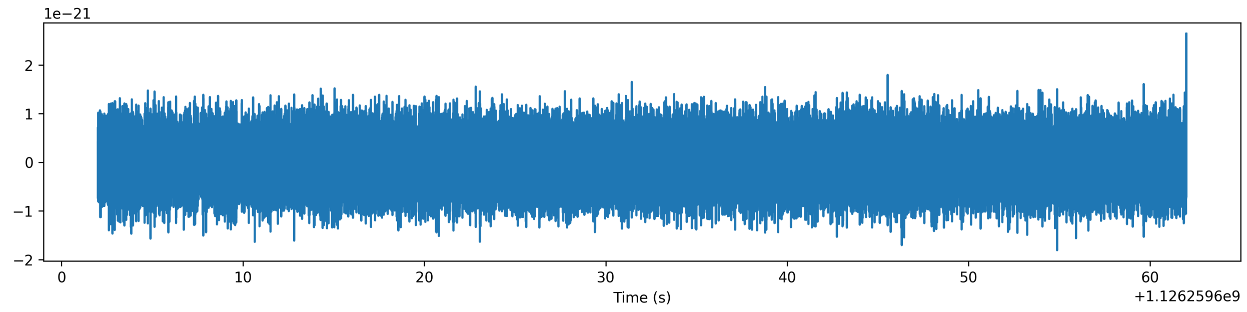


Figure 1: High-pass filtered strain data. The low-frequency noise has been removed.

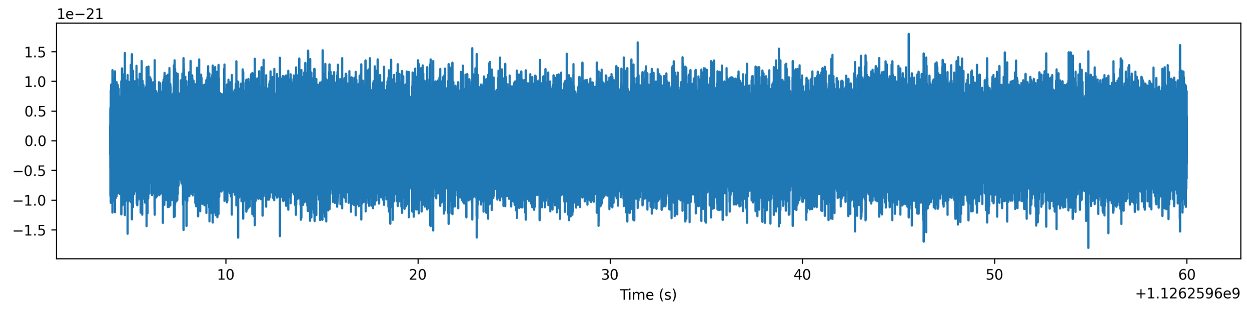


Figure 2: High-pass filtered and cropped strain data. The low-frequency noise has been removed, and the data is cropped to remove edge effects.

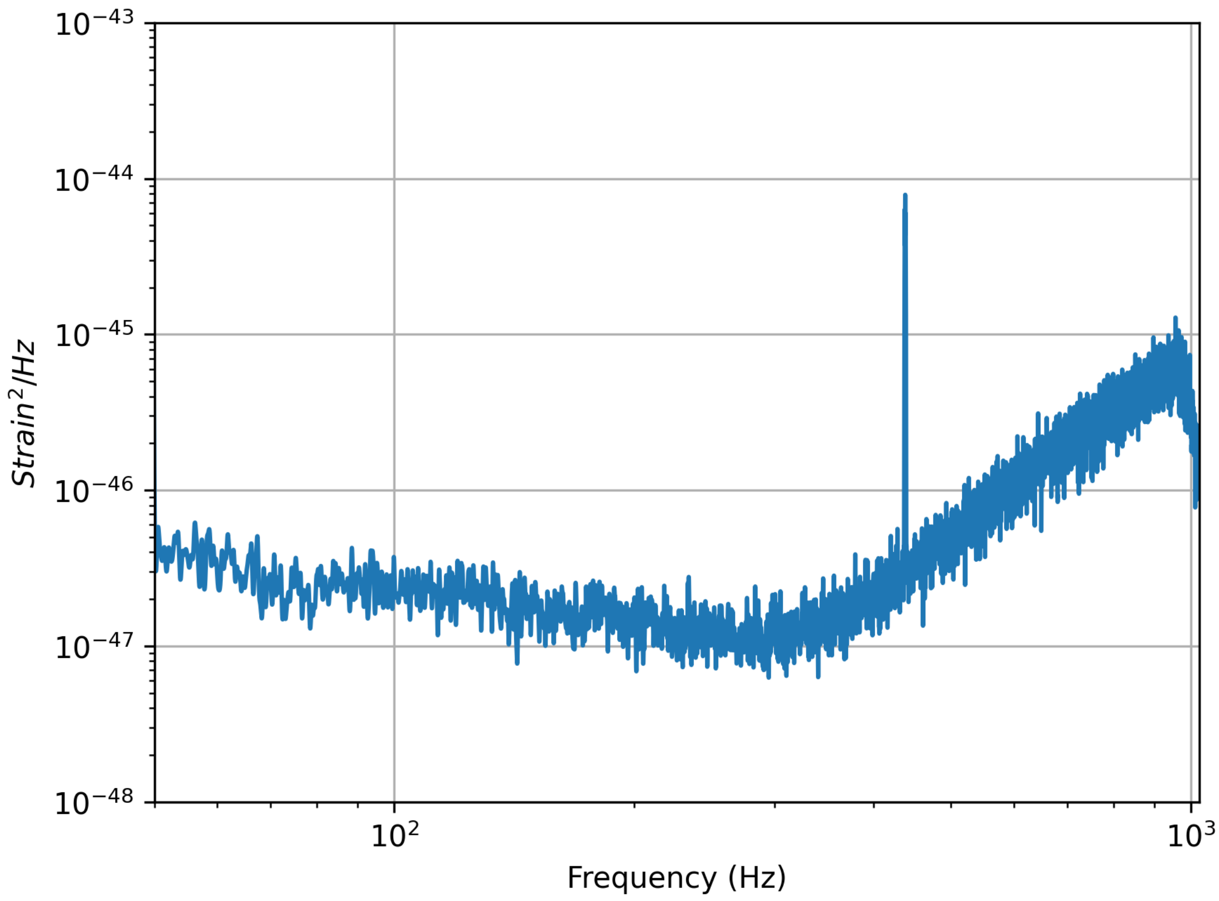


Figure 3: Power spectral density of the High-pass filtered and cropped strain. The lower cutoff frequency was chosen as 50 Hz since the GW signal is weak below this frequency.

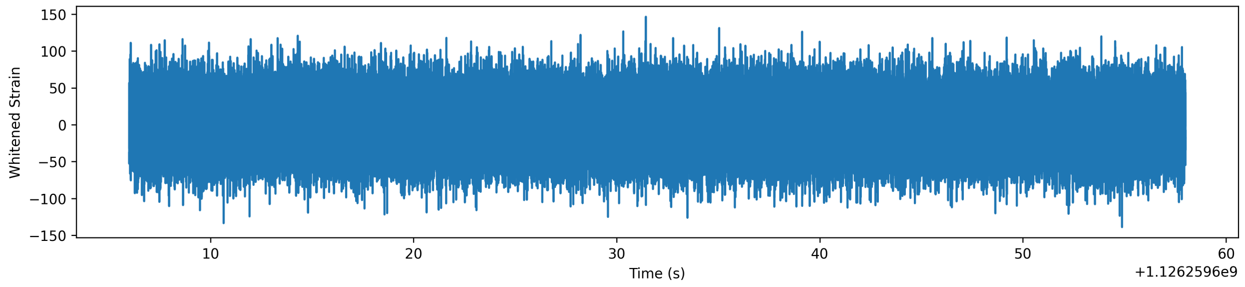


Figure 4: Whitened strain data. Whitening enhances the signal by removing the frequency-dependent noise.

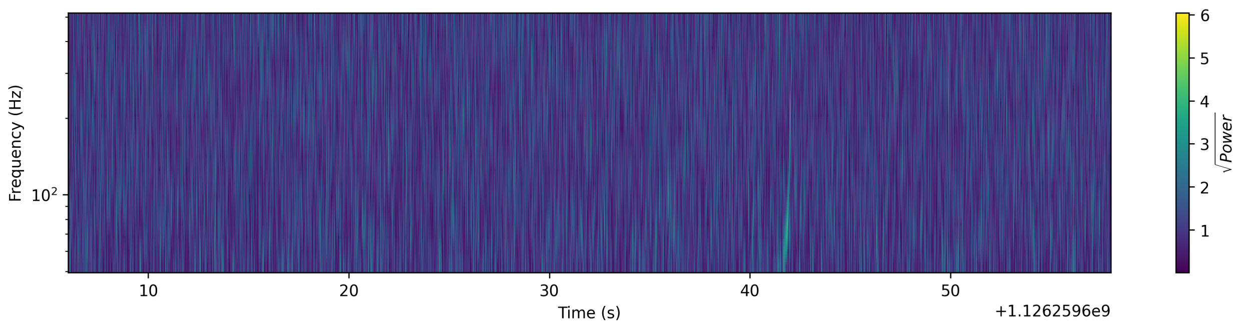


Figure 5: Q-transform of the whitened data, showing the signal in time-frequency space. A transient feature is visible in the 50–300 Hz range.

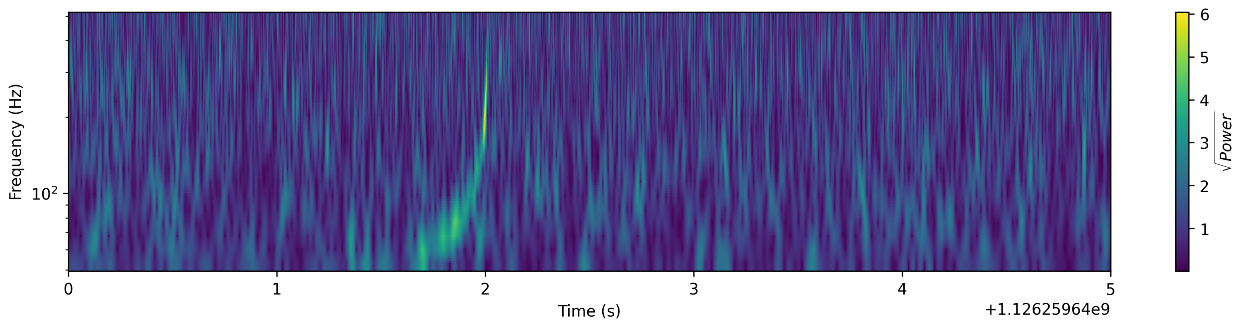


Figure 6: Zoomed Q-transform plot focusing on the signal around the merger time. The feature becomes more prominent in this view.

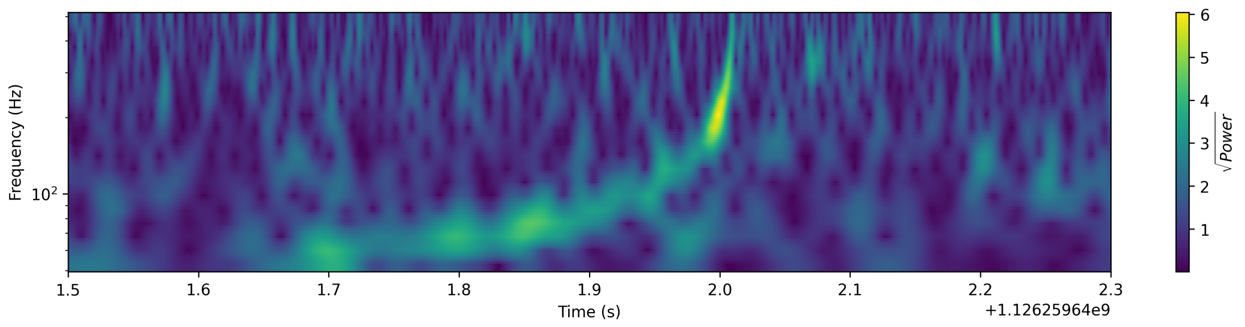


Figure 7: Further zoomed Q-transform plot around the merger time. The signal is clearly visible in the time-frequency representation, with a coalescence time around 1126259642.0 seconds.

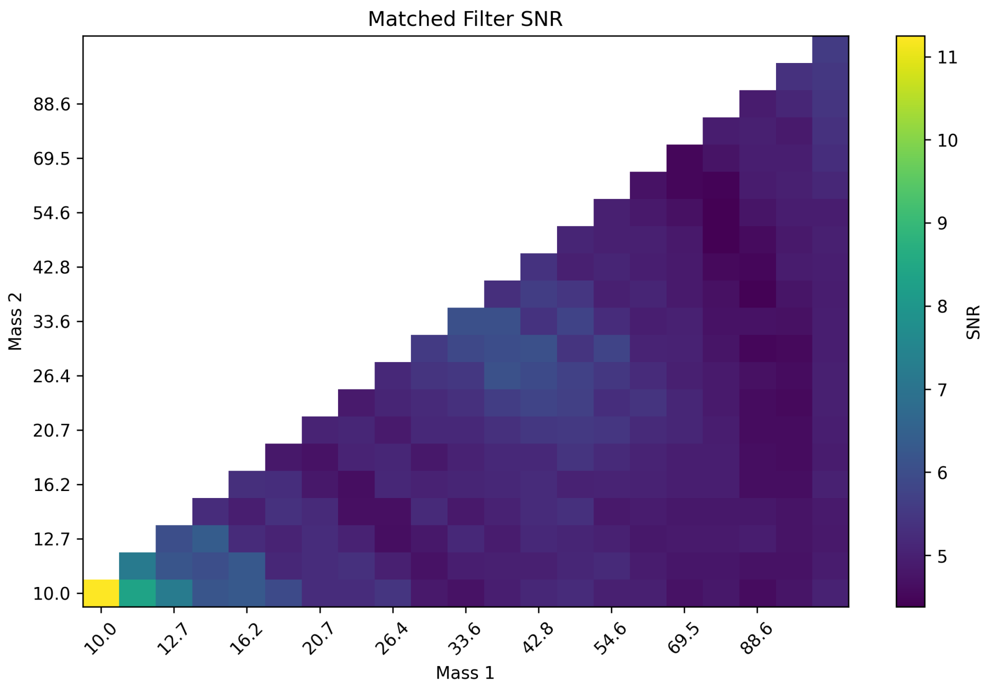


Figure 8: SNR heatmap from the coarse mass parameter search (10–100 $M_\odot$). The highest SNR corresponds to $M_1 = 10M_\odot$ and $M_2 = 10M_\odot$.

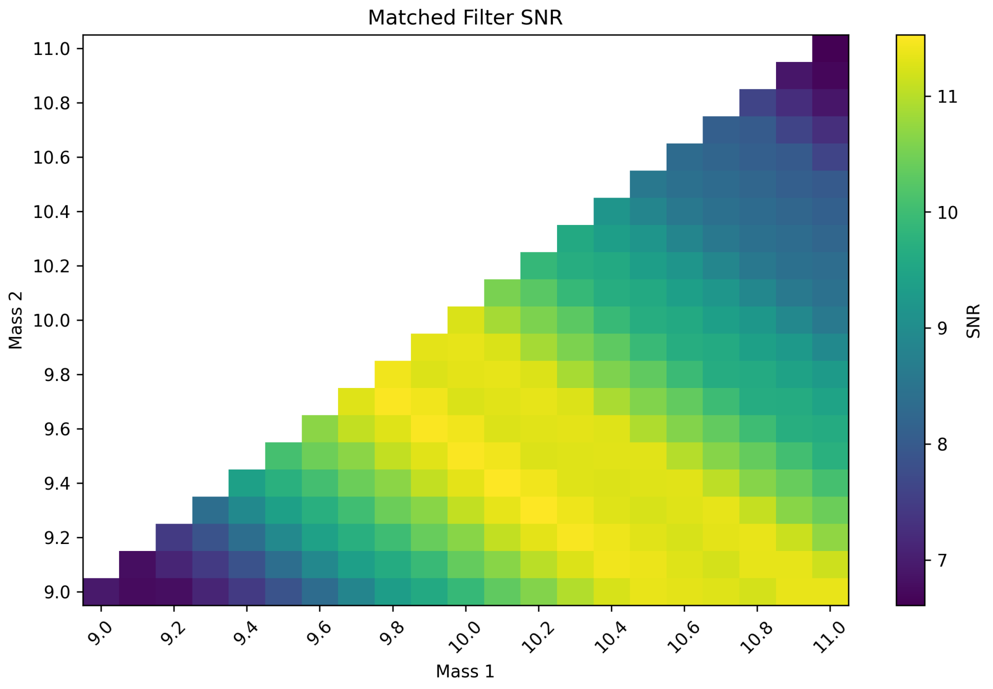


Figure 9: SNR heatmap from the fine mass parameter search (9–11 $M_\odot$). The parameters were refined to $M_1 = 10.1M_\odot$ and $M_2 = 9.4M_\odot$.

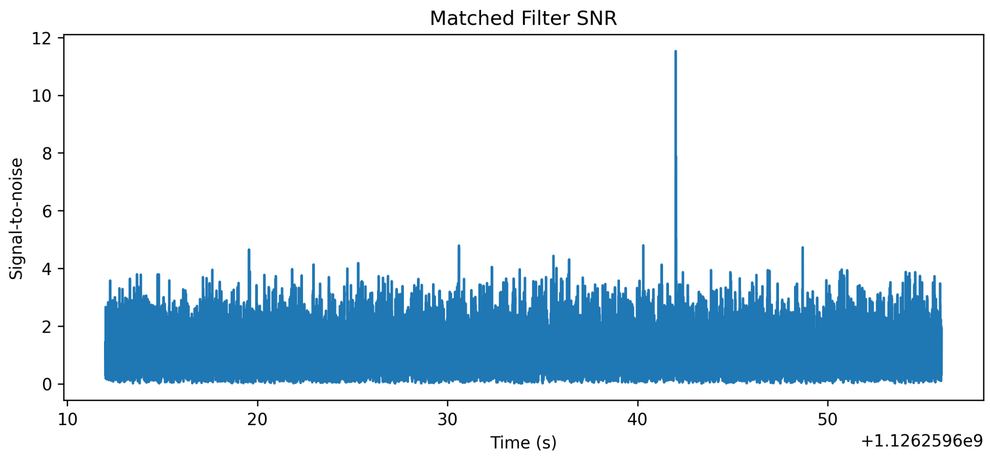


Figure 10: Matched filter SNR as a function of time shift. The peak SNR corresponds to the gravitational wave signal at 1126259642.0 seconds.

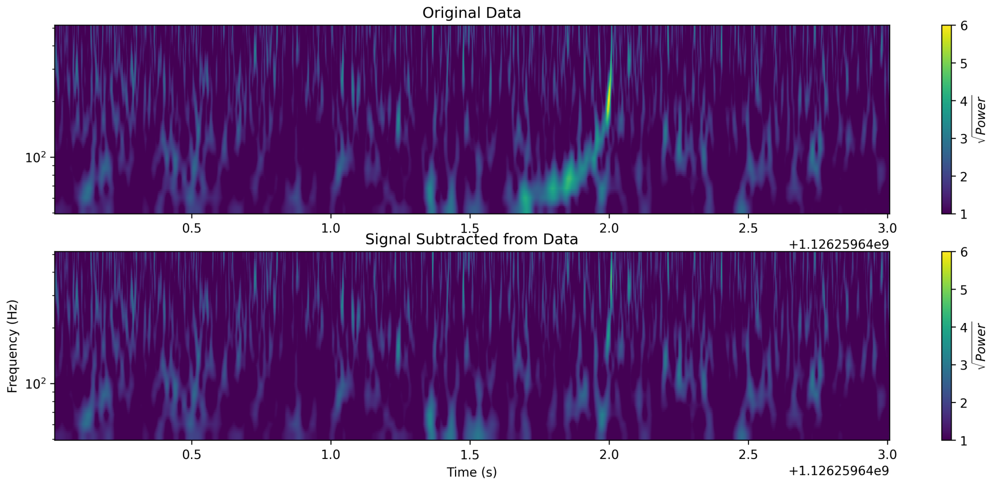


Figure 11: Q-transform of the residual after subtracting the signal from the data. The absence of significant features confirms the extracted signal closely matches the detected gravitational wave.

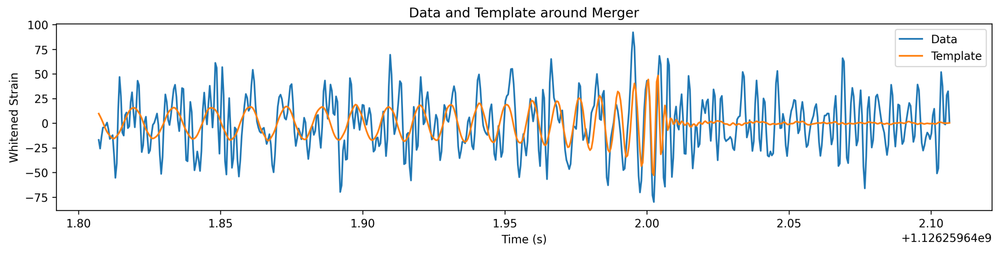


Figure 12: Overlay of the whitened data and best-fit template around the merger event. The excellent agreement highlights the accuracy of the waveform model and parameter estimation.

4 Discussion

The results from this analysis strongly indicate that the observed gravitational wave signal is consistent with a binary black hole (BBH) merger. Using matched filtering techniques, the masses of the component black holes were determined to be approximately $m_1 = 10.1M_\odot$ and $m_2 = 9.4M_\odot$, placing this system within the lower-mass regime for stellar-origin black holes. These values are refined from the coarse parameter space search, demonstrating the importance of iterative template matching for precise parameter estimation.

The matched filter signal-to-noise ratio (SNR) had a sharp peak of height 11.53, indicating a strong correlation between the data and the best-fit waveform template. This high SNR not only confirms the presence of a signal but also validates the accuracy of the mass parameters derived from the analysis. The time-frequency structure of the subtracted Q-transform plot further supports the successful detection and subtraction of the gravitational wave signal, with no significant features remaining in the residual data.

4.1 Limitations

While the results are robust, the analysis is limited by the assumptions inherent in the waveform models, such as aligned spins and quasi-circular orbits. Extending the parameter space to include spin-orbit misalignment or eccentric orbits could improve the accuracy of the method. Other models can also be explored in future work so that NSBH and BNS systems are also detected.

References

[1] PyCBC Team. *PyCBC Tutorials*. URL: https://github.com/gwastro/PyCBC-Tutorials/tree/master/tutorial.

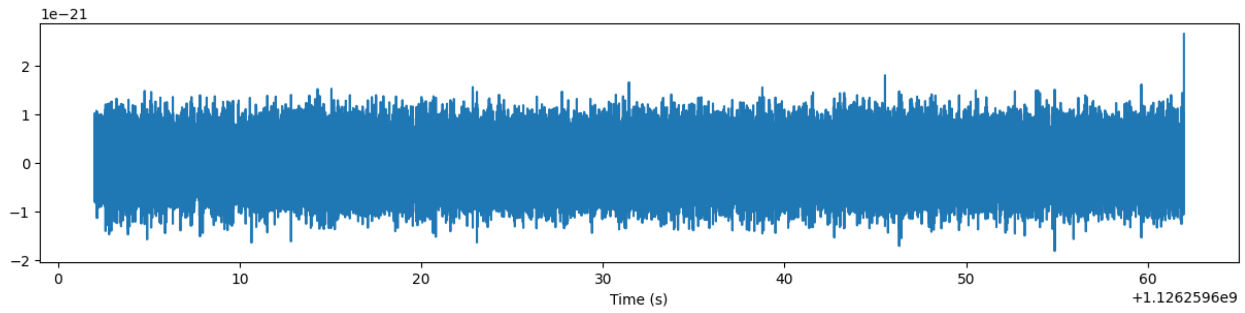
[2] Gravitational Wave Open Data Workshop. *Matched Filtering in Action - ODW 2020*. URL: https://github.com/gw-odw/odw-2020/blob/master/Day_2/Tuto_2.2_Matched_Filtering_In_action.ipynb.

Importing Libraries

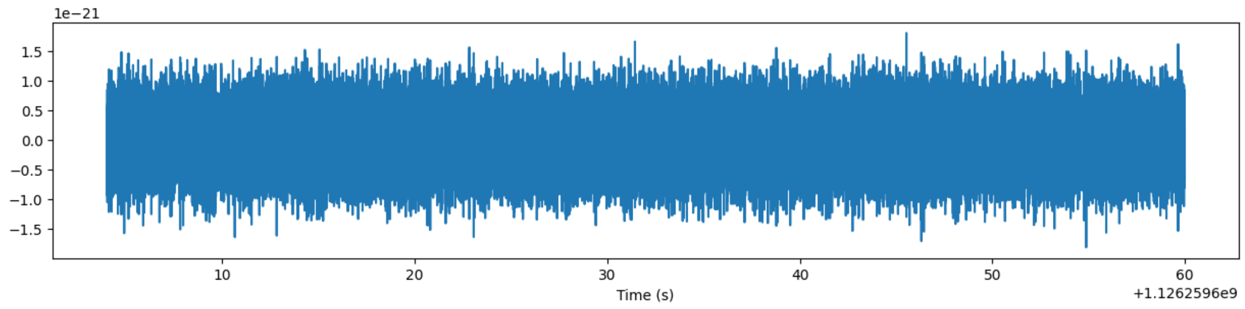
```
%matplotlib inline
import warnings
warnings.filterwarnings("ignore", "Wswiglal-redir-stdio")
import pylab
from pycbc.filter import resample_to_delta_t, highpass,
matched_filter, sigma
from pycbc.frame import read_frame
from pycbc.waveform import get_td_waveform
from pycbc.psd import interpolate, inverse_spectrum_truncation
import numpy
import pandas as pd
```

Read the data frame

```
strain = read_frame('det2.gwf',channels=['V1:FAKE_STRAIN'])[0]

# Remove the low frequency content and downsample the data to 2048Hz
strain = highpass(strain, 50.0)
strain = resample_to_delta_t(strain, 1.0/2048)
pylab.figure(figsize=[15, 3])
pylab.plot(strain.sample_times, strain)
pylab.xlabel('Time (s)')
pylab.savefig('plots/highpass_strain.png',dpi=300,
bbox_inches='tight')
pylab.show()
```

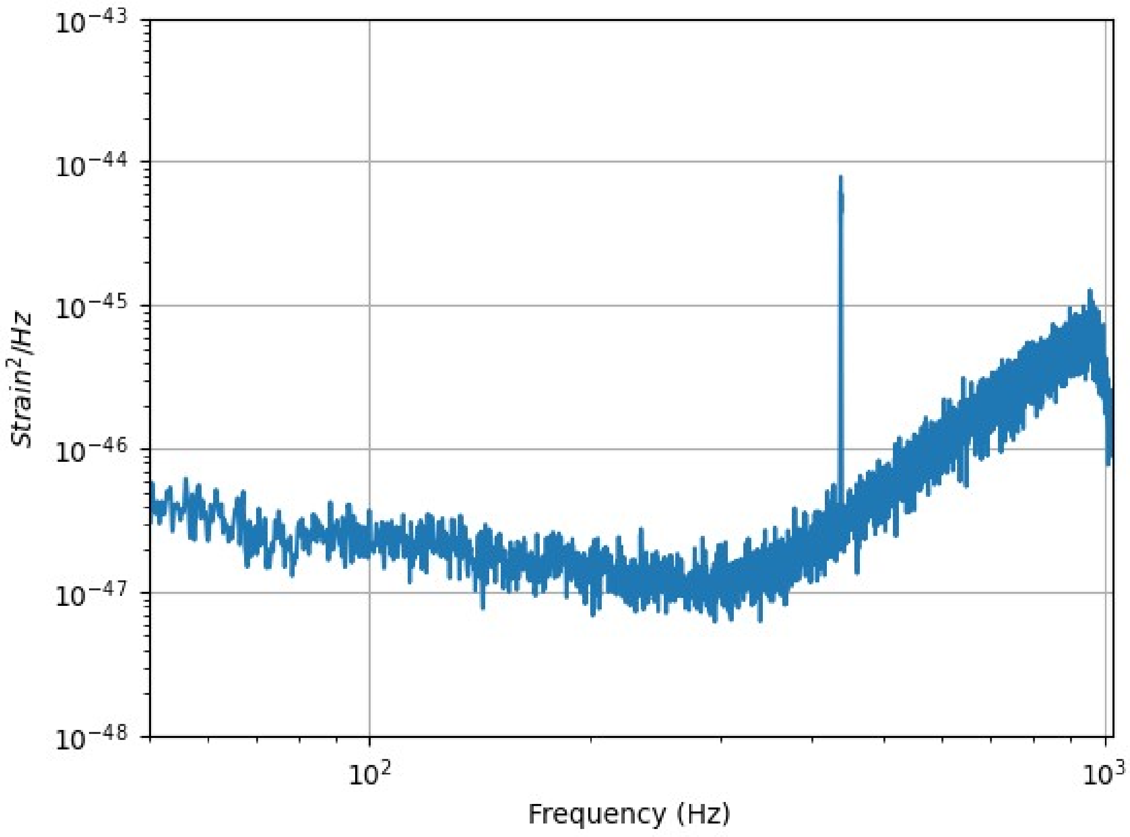


```
# Remove 2 seconds of data from both the beginning and end
conditioned = strain.crop(2, 2)
pylab.figure(figsize=[15, 3])
pylab.plot(conditioned.sample_times, conditioned)
pylab.xlabel('Time (s)')
pylab.savefig('plots/cropped_strain.png',dpi=300, bbox_inches='tight')
pylab.show()
```



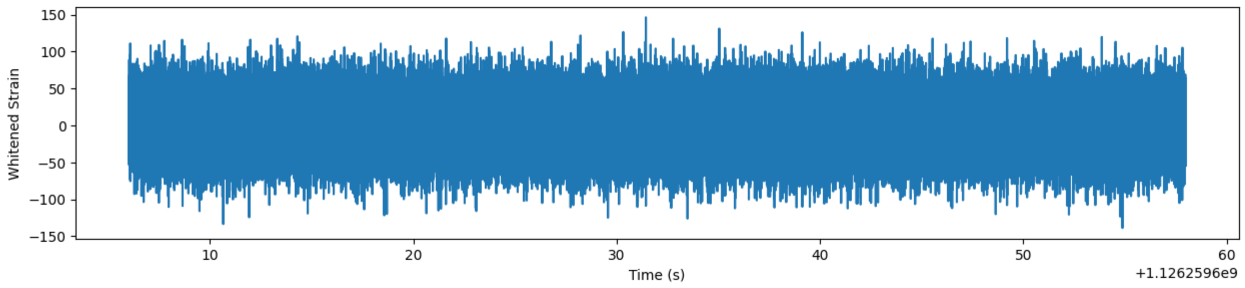
Estimate the power spectral density

```
# 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=50)

# Note that the psd is a FrequencySeries!
pylab.loglog(psd.sample_frequencies, psd)

pylab.ylabel('$Strain^2 / Hz$')
pylab.xlabel('Frequency (Hz)')
pylab.grid()
pylab.xlim(50, 1024)
pylab.ylim(1e-48, 1e-43)
pylab.savefig('plots/psd.png',dpi=300, bbox_inches='tight')
pylab.show()
```

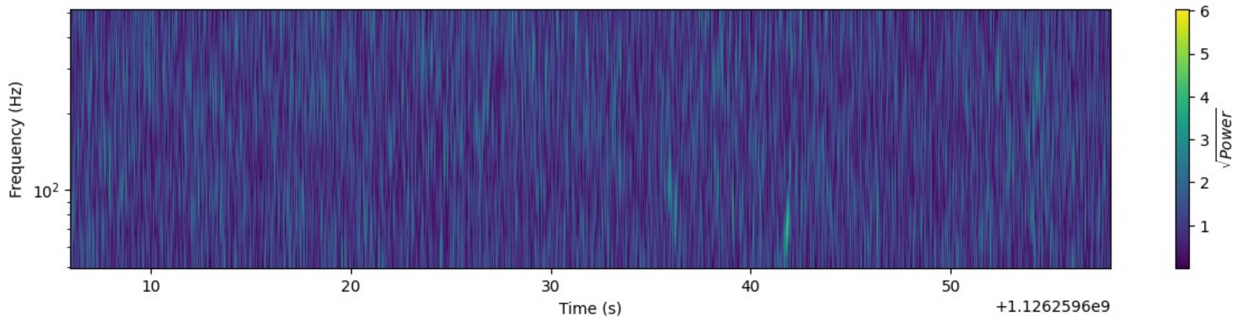


Whiten the data

```
# This produces a whitened set.
# This works by estimating the power spectral density from the
# data and then flattening the frequency response.
# (1) The first option sets the duration in seconds of each
#     sample of the data used as part of the PSD estimate.
# (2) The second option sets the duration of the filter to apply
whitened = conditioned.whiten(4, 4)

pylab.figure(figsize=[15, 3])
pylab.plot(whitened.sample_times, whitened)
pylab.ylabel('Whitened Strain')
pylab.xlabel('Time (s)')
pylab.savefig('plots/whitened_strain.png',dpi=300,
bbox_inches='tight')
pylab.show()
```

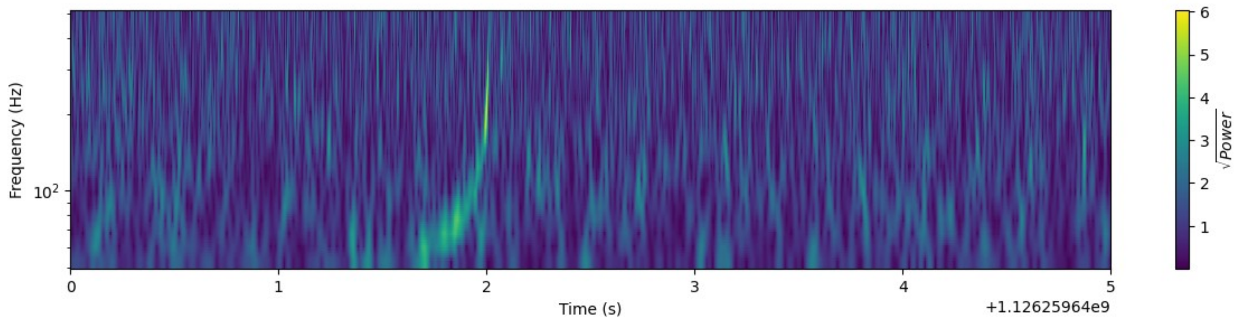


```
# The qtransform method returns a vector of the sample times,
frequencies, and a 2-d vector of the
# power in each time-frequency bin. The free parameter is the choice
of the Q-value. Larger Q-values
# are generally more appropriate for viewing long duration features of
the data and vice versa.

# The options here:
# (1) The time spacing for the output image (i.e. 1 ms in this case)
# (2) The number of frequency bins in the output, logarithmically
spaced
# (3) The qrange to maximize over. We'll pick a constant at 8 here
#     Typically higher values will be more appropriate for longer
duration
#     signals
# (4) The frequency range to output
times, freqs, power = whitened.qtransform(.001, logfsteps=100,
                                           qrange=(8, 8),
                                           frange=(50, 512),
                                          )
pylab.figure(figsize=[15, 3])
pylab.pcolormesh(times, freqs, power**0.5)
# pylab.xlim(m_time - 0.5, m_time + 0.3)
pylab.yscale('log')
pylab.xlabel('Time (s)')
pylab.ylabel('Frequency (Hz)')
pylab.colorbar(label='$\\sqrt{Power}$')
pylab.savefig('plots/qtransform.png',dpi=300, bbox_inches='tight')
pylab.show()
```



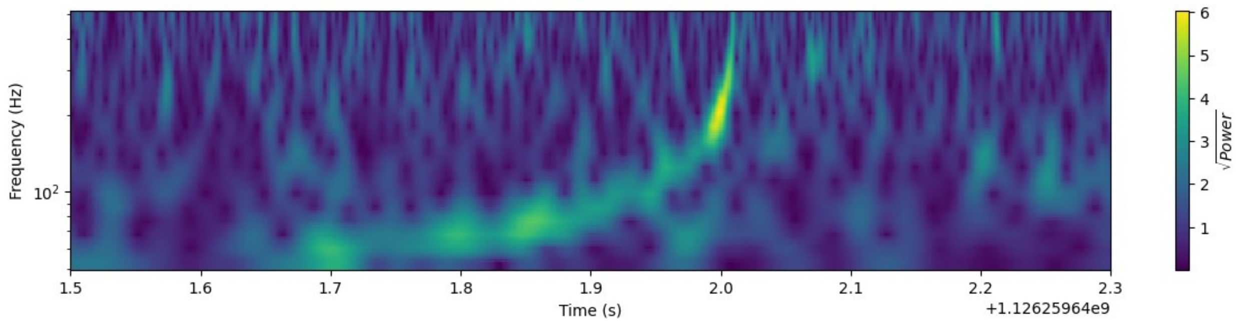
Enhance! Zoom to the signal in the data

```
pylab.figure(figsize=[15, 3])
pylab.pcolormesh(times, freqs, power**0.5)
pylab.xlim(1126259640, 1126259645)
pylab.yscale('log')
pylab.xlabel('Time (s)')
pylab.ylabel('Frequency (Hz)')
pylab.colorbar(label='$\\sqrt{Power}$')
pylab.savefig('plots/qtransform_zoomed.png',dpi=300,
bbox_inches='tight')
pylab.show()
```



Its clearer now! One final push!

```
m_time = 1126259642.0
pylab.figure(figsize=[15, 3])
pylab.pcolormesh(times, freqs, power**0.5)
pylab.xlim(m_time - 0.5, m_time + 0.3)
pylab.yscale('log')
pylab.xlabel('Time (s)')
pylab.ylabel('Frequency (Hz)')
pylab.colorbar(label='$\\sqrt{Power}$')
pylab.savefig('plots/qtransform_zoomed2.png',dpi=300,
bbox_inches='tight')
pylab.show()
```



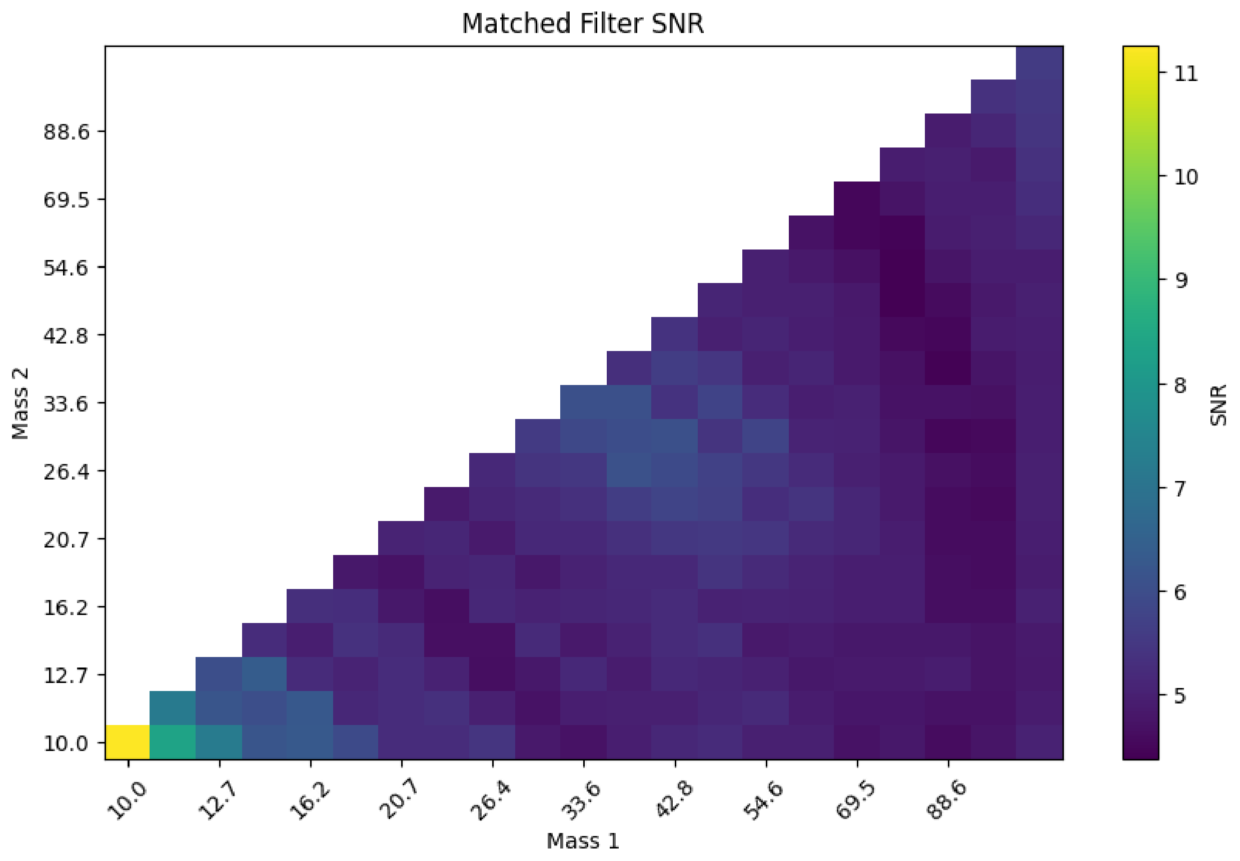
Searching the parameter space using matched filtering

```
def search_mass_space(masses,output_file):
    df = []
    for idx1 in range(len(masses)):
        for idx2 in range(idx1, len(masses)):
            m1 = masses[idx2]
            m2 = masses[idx1]
            hp, hc = get_td_waveform(approximant="SEOBNRv4",
                                     mass1=m1,
                                     mass2=m2,
                                     delta_t=conditioned.delta_t,
                                     f_lower=50)
            hp.resize(len(conditioned))
            template = hp.cyclic_time_shift(hp.start_time)
            snr = matched_filter(template, conditioned,
                                     psd=psd, low_frequency_cutoff=50)
            snr = snr.crop(4 + 4, 4)
            peak = abs(snr).numpy().argmax()
            snrp = snr[peak]
            time = snr.sample_times[peak]
            print("Found a signal at {}s with SNR
{}".format(time,abs(snrp)))
            df.append({'mass1': m1, 'mass2': m2, 'time': time, 'snr':
abs(snrp)})
    df = pd.DataFrame(df)
    # save to results.csv
    df.to_csv(output_file, index=False)
    return df

def plot_mass_space(df, masses, output_file):
    # use imshow to plot the SNR values with mass1 on the x-axis and
mass2 on the y-axis (note only mass2 >= mass1 are valid)
    pylab.figure(figsize=[10, 6])
    pylab.imshow(df.pivot(index='mass2', columns='mass1',
values='snr'), aspect='auto', origin='lower')
    pylab.colorbar(label='SNR')
    pylab.xlabel('Mass 1')
    pylab.xticks(range(0, len(masses), 2), ['%.1f' % m for m in
masses[::2]], rotation=45)
    pylab.ylabel('Mass 2')
    pylab.yticks(range(0, len(masses), 2), ['%.1f' % m for m in
masses[::2]])
    pylab.title('Matched Filter SNR')
    pylab.savefig(output_file,dpi=300, bbox_inches='tight')
    pylab.show()
```

Coarse search (from 10 to 100 $M_\odot$)

```
# mass range for typical compact binary systems is 10-100 solar masses
masses_coarse = numpy.logspace(1, 2, 20)
# df_coarse = search_mass_space(numpy.logspace(1, 2, 20),
'results_coarse.csv')
df_coarse = pd.read_csv('results_coarse.csv')
# find the (m1, m2) pair with the highest SNR from df
idx = df_coarse['snr'].idxmax()
m1 = df_coarse.loc[idx, 'mass1']
m2 = df_coarse.loc[idx, 'mass2']
t = df_coarse.loc[idx, 'time']
print(f"The highest SNR is {df_coarse.loc[idx, 'snr']:.2f} at time
{t:.2f}s with m1={m1:.2f} and m2={m2:.2f}")
plot_mass_space(df_coarse, masses_coarse,
'plots/mass_space_coarse.png')
```

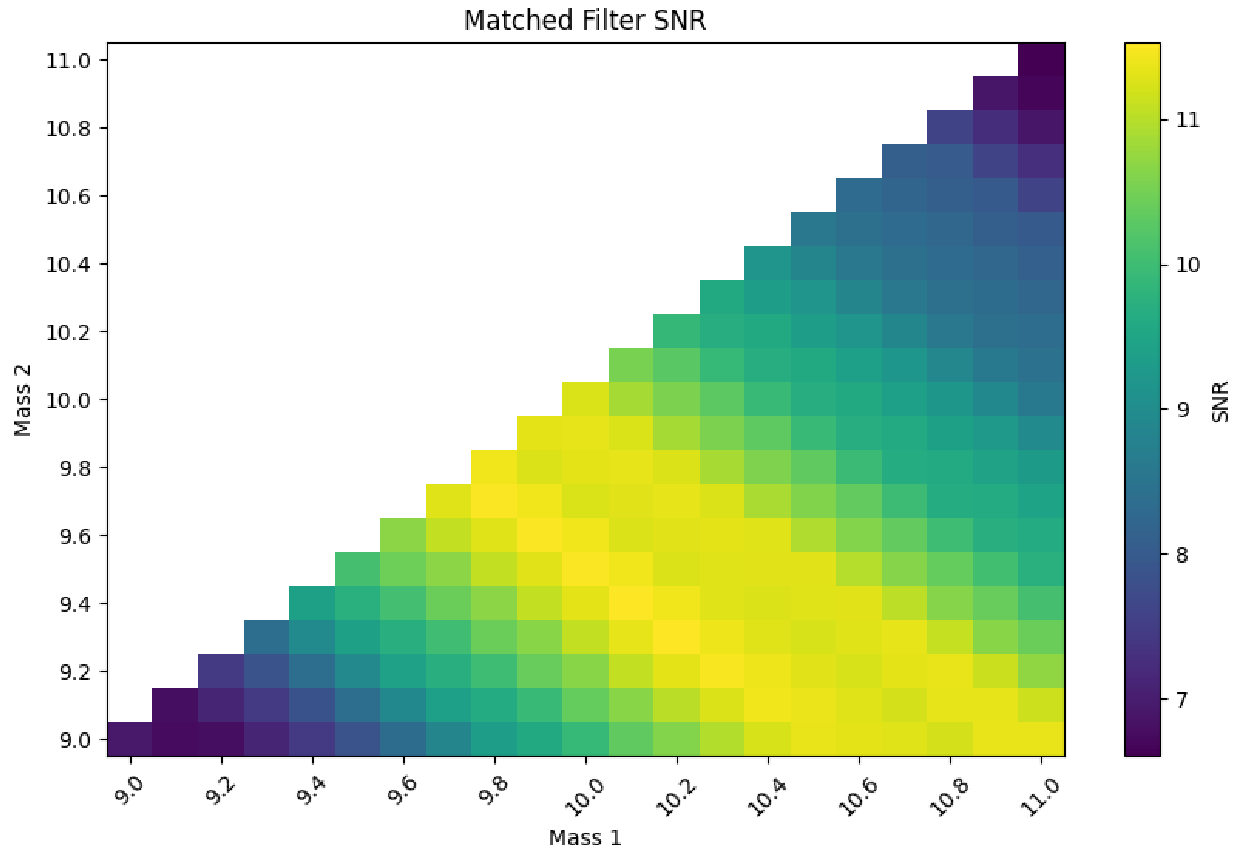
```
The highest SNR is 11.25 at time 1126259642.00s with m1=10.00 and
m2=10.00
```



Fine search (from 9 to 11 $M_\odot$)

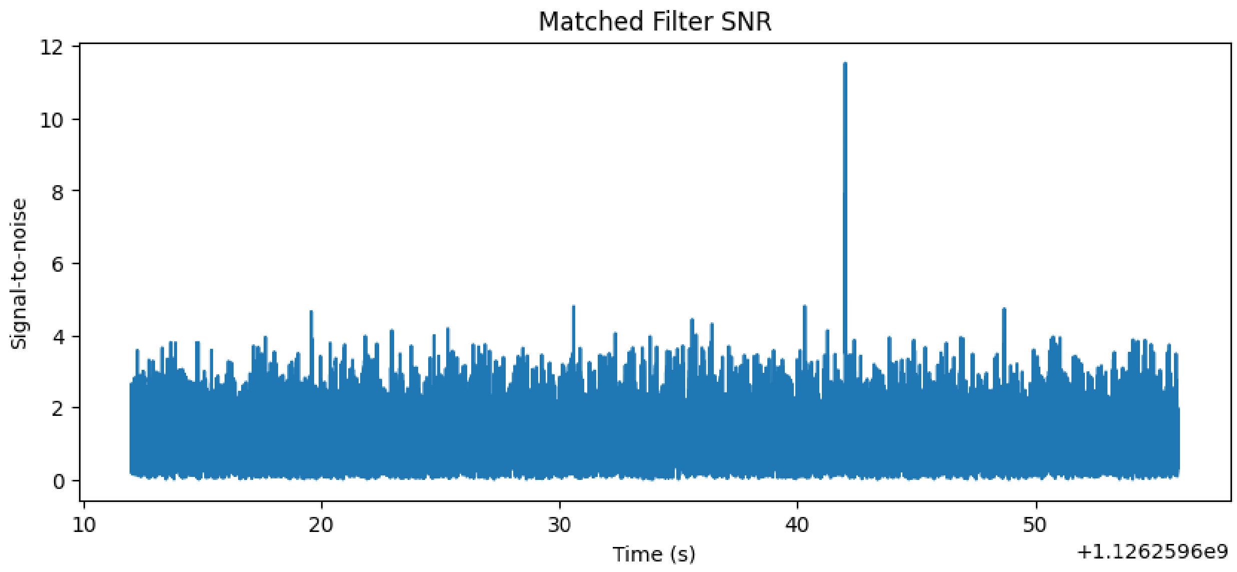
```
masses_fine = numpy.linspace(9,11,21)
# df_fine = search_mass_space(masses_fine, 'results_fine.csv')
df_fine = pd.read_csv('results_fine.csv')
# find the (m1, m2) pair with the highest SNR from df
idx = df_fine['snr'].idxmax()
m1 = df_fine.loc[idx, 'mass1']
m2 = df_fine.loc[idx, 'mass2']
t = df_fine.loc[idx, 'time']
print(f"The highest SNR is {df_fine.loc[idx, 'snr']:.2f} at time
{t:.2f}s with m1={m1:.2f} and m2={m2:.2f}")
plot_mass_space(df_fine, masses_fine, 'plots/mass_space_fine.png')
```

```
The highest SNR is 11.53 at time 1126259642.01s with m1=10.10 and
m2=9.40
```



Create template for optimal matched filter

```
hp, hc = get_td_waveform(approximant="SEOBNRv4",
                         mass1=m1,
                    mass2=m2,
                    delta_t=conditioned.delta_t,
                    f_lower=50)
hp.resize(len(conditioned))
template = hp.cyclic_time_shift(hp.start_time)
snr = matched_filter(template, conditioned,
                    psd=psd, low_frequency_cutoff=50)
snr = snr.crop(4 + 4, 4)
peak = abs(snr).numpy().argmax()
snrp = snr[peak]
time = snr.sample_times[peak]

pylab.figure(figsize=[10, 4])
pylab.plot(snr.sample_times, abs(snr))
pylab.ylabel('Signal-to-noise')
pylab.xlabel('Time (s)')
pylab.title('Matched Filter SNR')
pylab.savefig('plots/matched_filter_snr.png',dpi=300,
bbox_inches='tight')
pylab.show()
```



Aligning and Subtracting the Proposed Signal

In the previous section we found a peak in the signal-to-noise for a proposed binary black hole merger. We can use this SNR peak to align our proposal to the data, and to also subtract our proposal from the data.

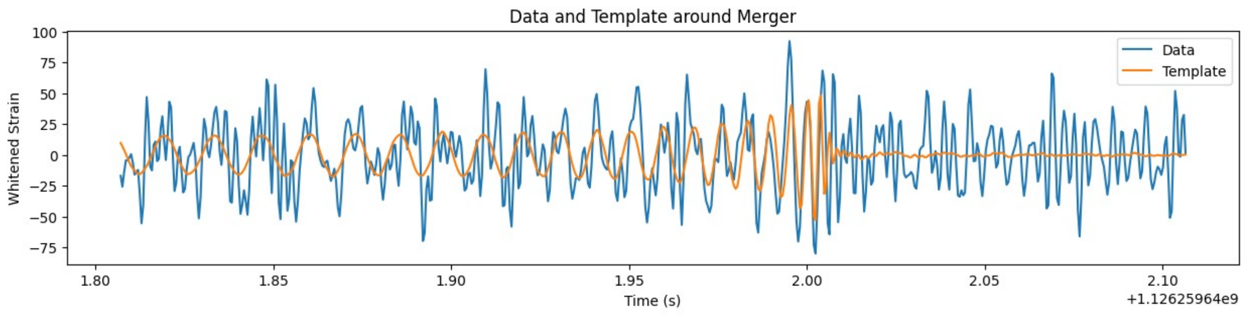
```
# The time, amplitude, and phase of the SNR peak tell us how to align
# our proposed signal with the data.

# 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=50.0,
high_frequency_cutoff=500.0)

# Scale the template amplitude and phase to the peak value
aligned = (aligned.to_frequencyseries() * snrp).to_timeseries()
aligned.start_time = conditioned.start_time
```

Visualize the overlap between the signal and data

To compare the data an signal on equal footing, and to concentrate on the frequency range that is important. We will whiten both the template and the data, and then bandpass both the data and template between 50-500 Hz. In this way, any signal that is in the data is transformed in the same way that the template is.

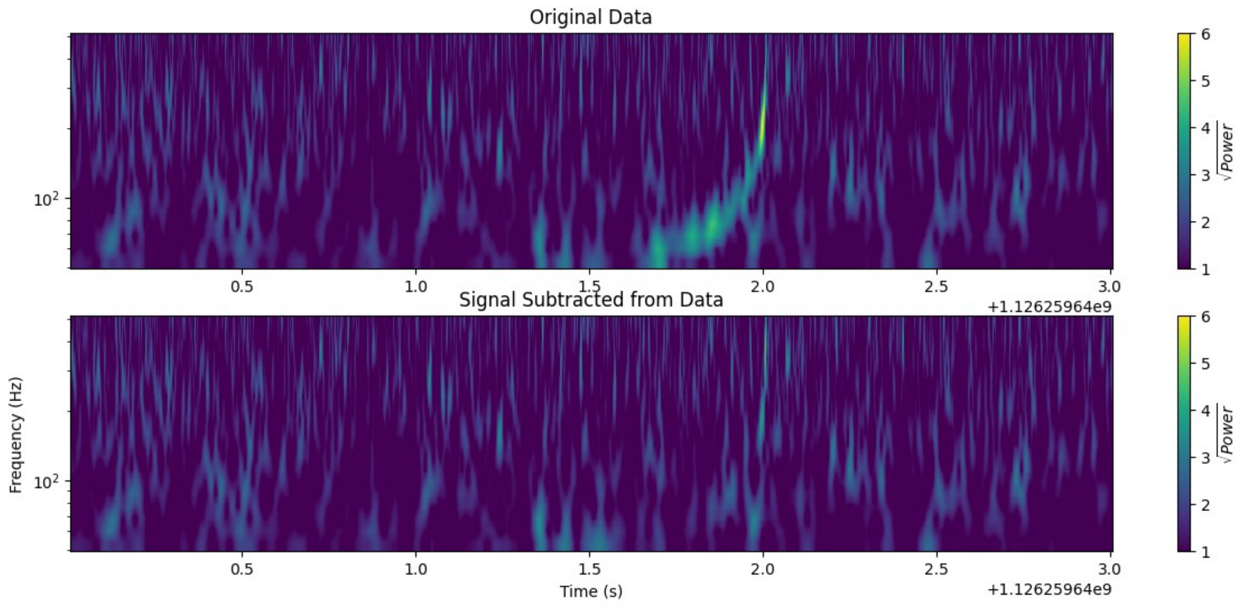
```
# We do it this way so that we can whiten both the template and the
data
white_data = (conditioned.to_frequencyseries() /
psd**0.5).to_timeseries()
white_template = (aligned.to_frequencyseries() /
psd**0.5).to_timeseries()

white_data = white_data.highpass_fir(50., 512).lowpass_fir(500, 512)
white_template = white_template.highpass_fir(50, 512).lowpass_fir(500,
512)

# Select the time around the merger
white_data = white_data.time_slice(time-.2, time+.1)
white_template = white_template.time_slice(time-.2, time+.1)

pylab.figure(figsize=[15, 3])
pylab.plot(white_data.sample_times, white_data, label="Data")
pylab.plot(white_template.sample_times, white_template,
label="Template")
pylab.legend()
pylab.ylabel('Whitened Strain')
pylab.xlabel('Time (s)')
pylab.title('Data and Template around Merger')
pylab.savefig('plots/overlay_data_template.png',dpi=300,
bbox_inches='tight')
pylab.show()
```



Subtracting the signal from the data

Now that we've aligned the template we can simply subtract it. Let's see below how that looks in the time-frequency plots!

```
subtracted = conditioned - aligned

# Plot the original data and the subtracted signal data
pylab.figure(figsize=[15, 6])

for i in [1,0]:
    data, title = [(conditioned, 'Original Data'),
                    (subtracted, 'Signal Subtracted from Data')][i]
    t, f, p = data.whiten(4, 4).qtransform(.001,
                                                  logfsteps=100,
                                                  qrange=(8, 8),
                                                  frange=(50, 512))
    if i==1:
        ax = pylab.subplot(2, 1, i+1)
    else:
        ax = pylab.subplot(2, 1, i+1, sharex=ax_bottom)
    if i==1:
        ax_bottom = ax
    ax.set_title(title)
    mesh = ax.pcolormesh(t, f, p**0.5, vmin=1, vmax=6)
    pylab.colorbar(mesh,label='$\\sqrt{Power}$')
    # add a color bar
    ax.set_yscale('log')
    if i==1:
        ax.set_xlabel('Time (s)')
        ax.set_ylabel('Frequency (Hz)')
        ax.set_xlim(time - 2, time + 1)
pylab.savefig('plots/subtracted.png',dpi=300, bbox_inches='tight')
pylab.show()
```


Original Data
Signal Subtracted from Data
Frequency (Hz)
10^2
Time (s)
0.5
1.0
1.5
2.0
2.5
3.0
+1.12625964e9
$\sqrt{Power}$
1
2
3
4
5
6