Binary

March 1, 2023

1 Binary black hole detections

Matriculation: 2663452m

1.1 Aims

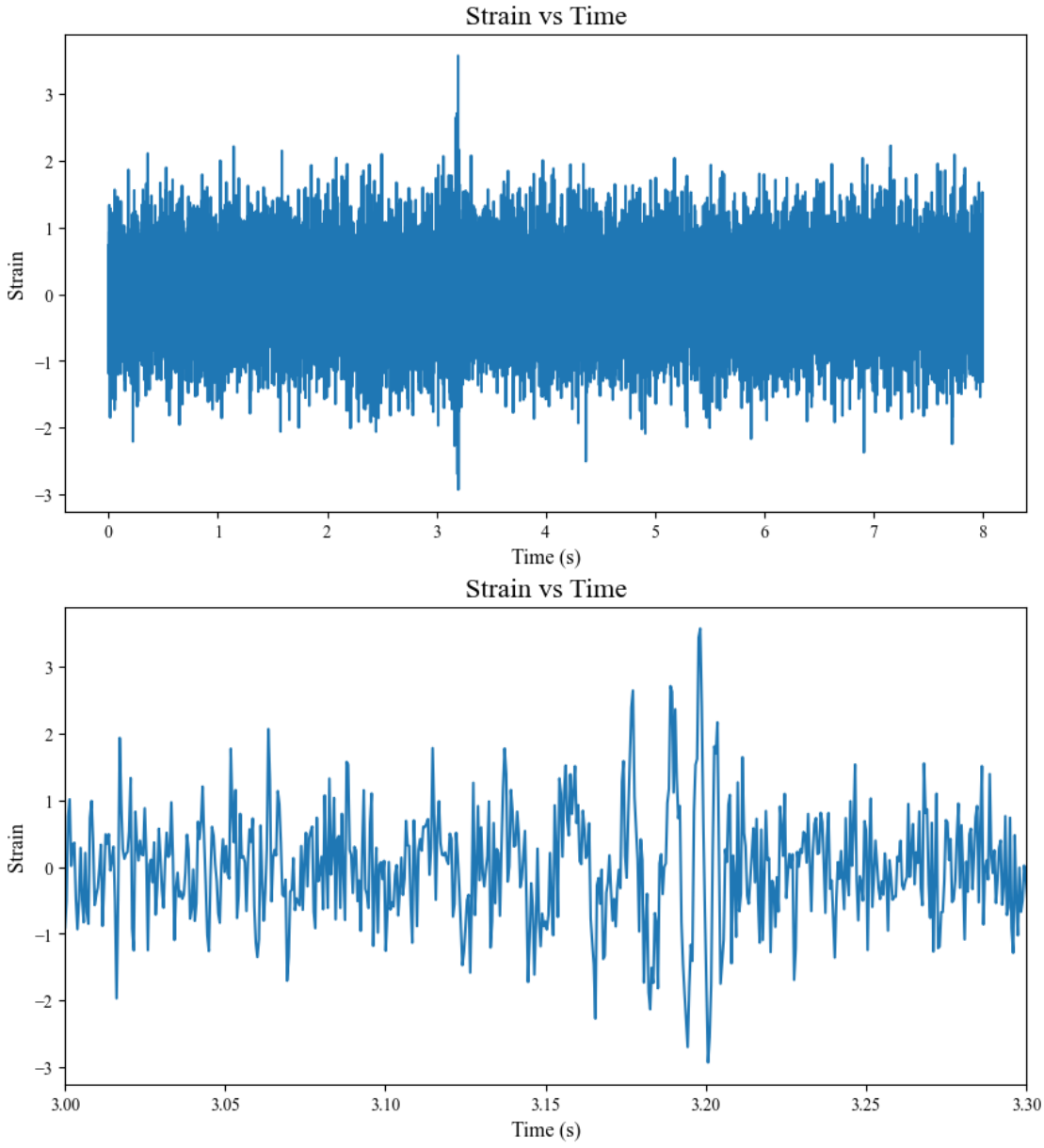
The aims of this experiment are to procees the gravitational wave data from the first and second runs of the LIGO/Virgo gravitational wave detectors and to discover properties of the binary black hole systems that prodiced the signals.

Gravitational waves were first predicted by Einstien in his 1915 paper on General Relativity but in 2015 the first gravitational wave was detected by the LIGO research group.

[219]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.optimize import curve_fit
import scipy.optimize as opt
import scipy.constants as const
import os
import lal as lal
from scipy.signal import spectrogram
import gw_detections_functions as gw
import pandas as pd


title_size = 16
axis_size = 12
plt.rcParams["font.family"] = "Times New Roman"
```

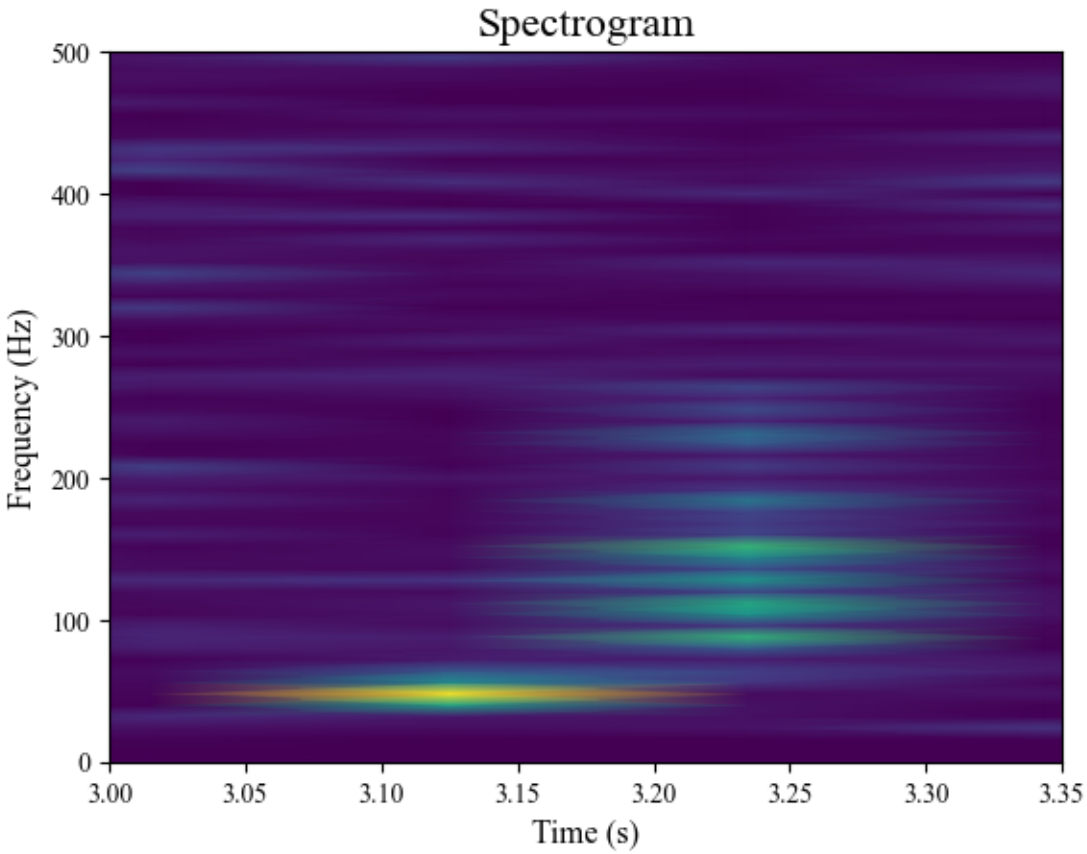
[220]:
```
data = np.loadtxt('strain_data/GW150914_strain.txt')
time = data[:,0]
strain = data[:,1]

plt.figure(figsize = (10,11))
plt.subplot(2,1,1)
plt.plot(time, strain)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.ylabel('Strain', fontsize=axis_size)
```

```
plt.title('Strain vs Time', fontsize=title_size)

plt.subplot(2,1,2)
plt.plot(time, strain)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.ylabel('Strain', fontsize=axis_size)
plt.title('Strain vs Time', fontsize=title_size)
plt.xlim(3,3.3)
plt.show()
```



The siganl shows a peak in the strain likely as the black holes collide releasing a lot of energy in a short amount of time.

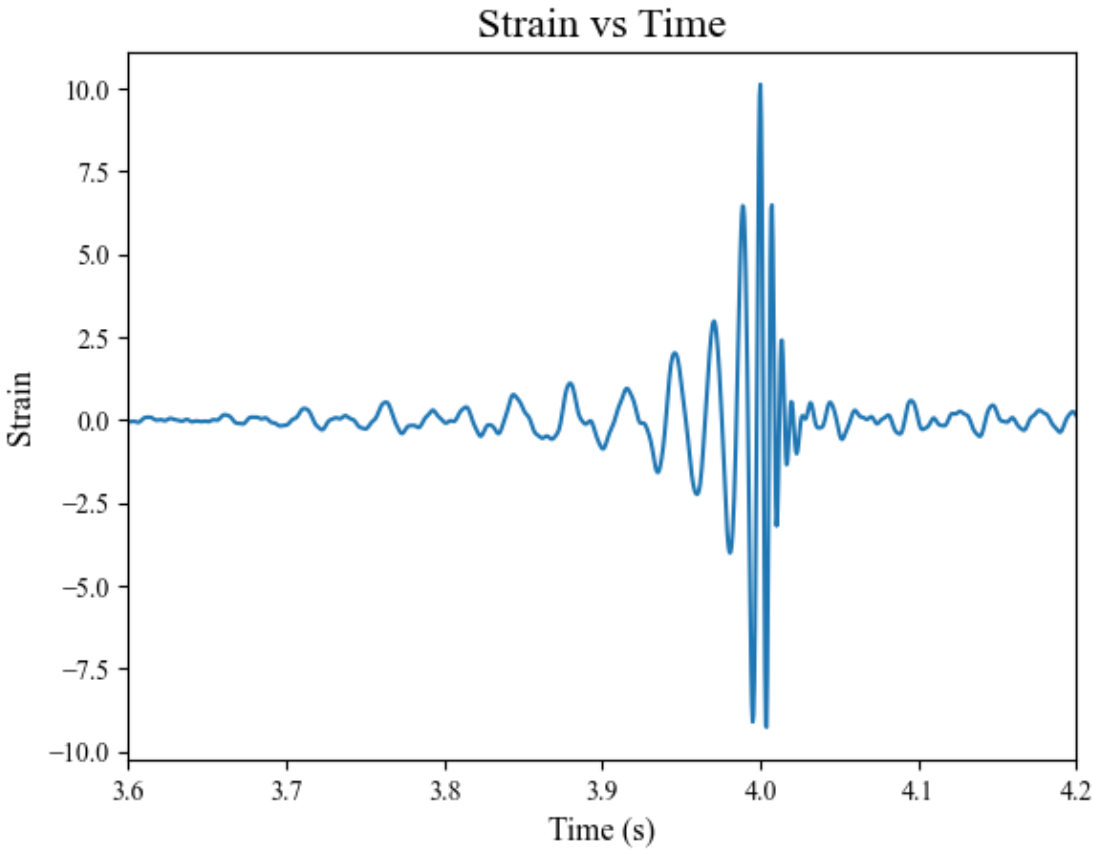
[221]:
```
spec_f, spec_t, spec = sp.signal.spectrogram(strain, 2048)
plt.pcolormesh(spec_t, spec_f, spec, shading='gouraud')
plt.ylabel('Frequency (Hz)', fontsize=axis_size)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.xlim(3,3.35)
plt.ylim(0,500)
plt.title('Spectrogram', fontsize=title_size)
plt.show()
```



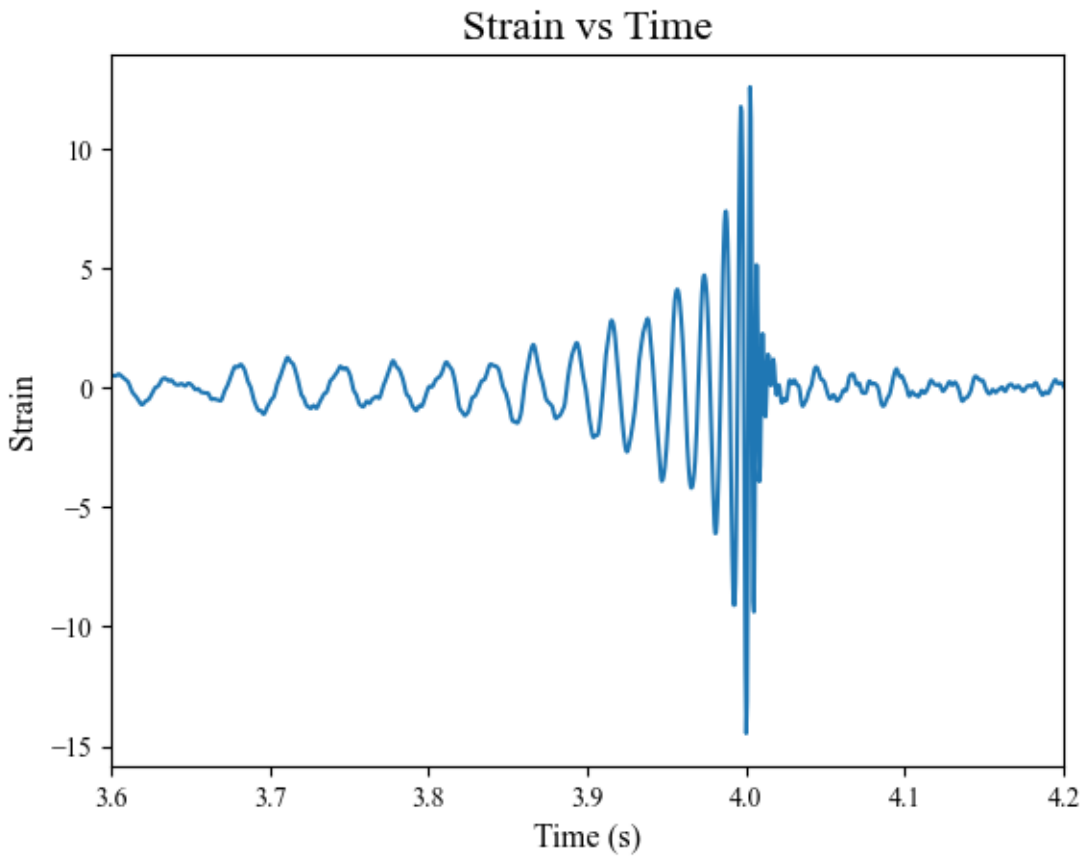
reasonable mass of a black hole in merger 7 - 37 solar masses

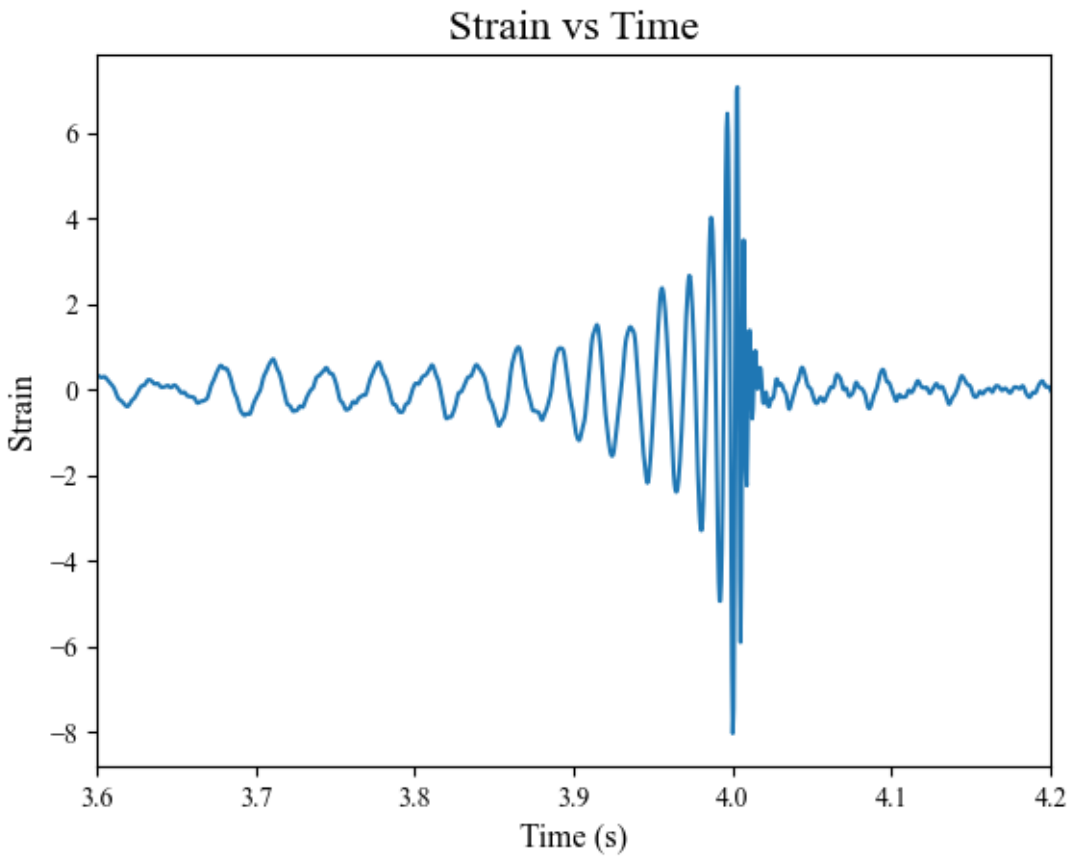
[222]:
```
inv_psd = np.loadtxt('inv_psd/GW150914_inv_psd.txt',usecols=(1,))
t, template = gw.make_template(70,36,2048,8,inv_psd,400)

plt.figure()
plt.plot(t, template)
plt.xlabel('Time (s)', fontsize=axis_size)
```

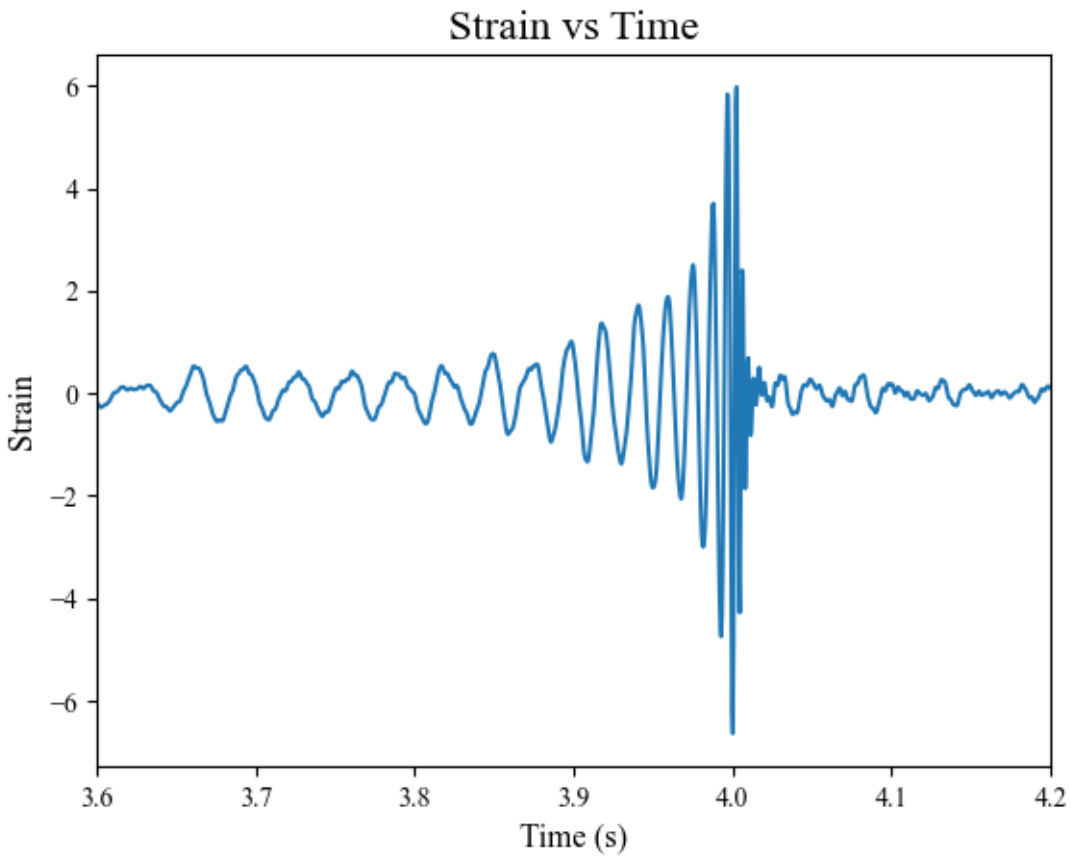
```
plt.xlim(3.6,4.2)
plt.ylabel('Strain', fontsize=axis_size)
plt.title('Strain vs Time', fontsize=title_size)
plt.show()
```

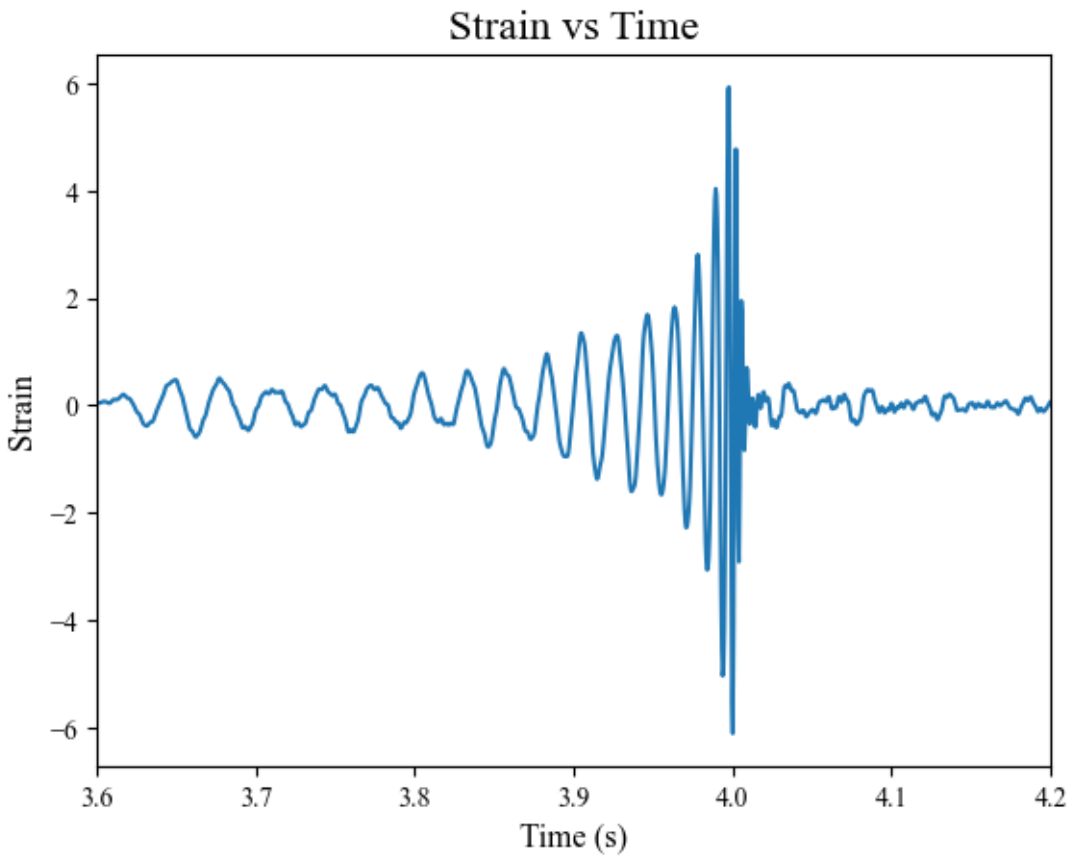


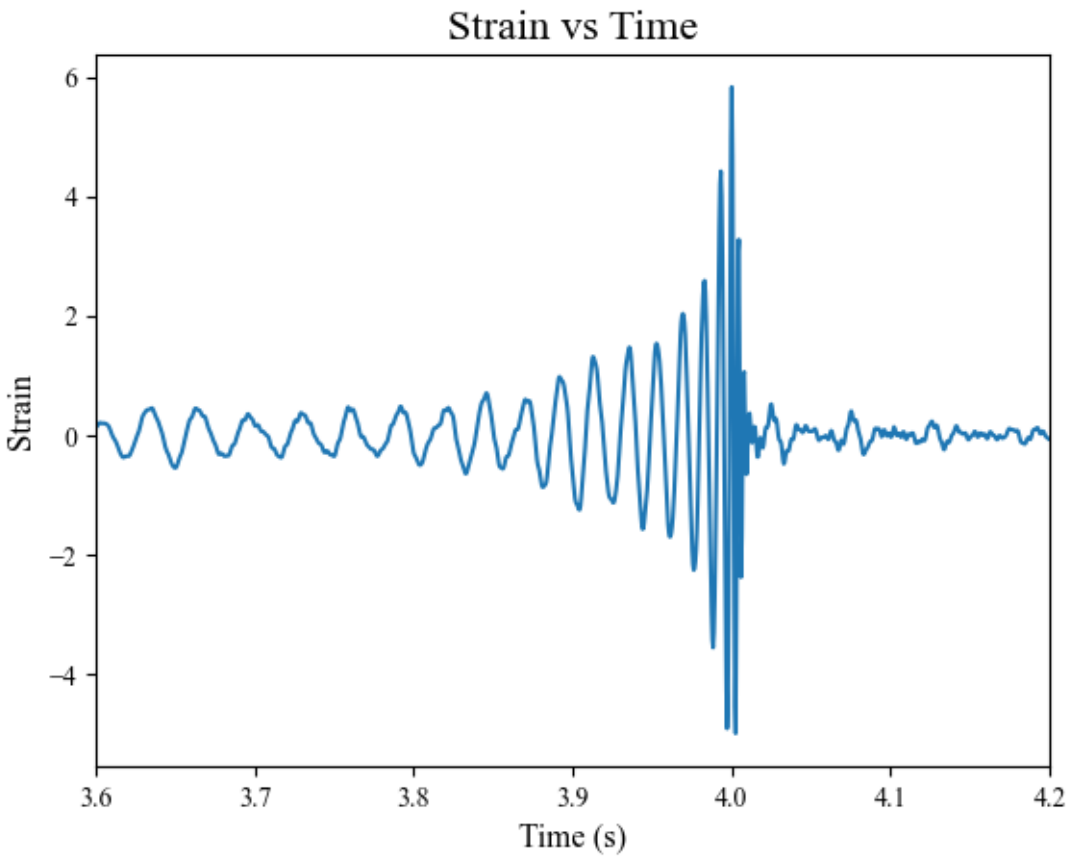
[223]:
```
mass_one = [37,38,36,35,34,33,32,31,25]
mass_two = [24,23,22,21,20,19,18,17,24]
mass_three = np.array(mass_one) - 1
distance = [400,500,350,450,430,510,390,200,420]
distance = np.sort(np.array(distance))
for i in range(len(mass_one)):
        t, template = gw.
 ↪make_template(mass_one[i],mass_two[i],2048,8,inv_psd,distance[i])
        plt.figure()
        plt.plot(t, template)
        plt.xlabel('Time (s)', fontsize=axis_size)
        plt.xlim(3.6,4.2)
        plt.ylabel('Strain', fontsize=axis_size)
        plt.title('Strain vs Time', fontsize=title_size)
        plt.show()
```

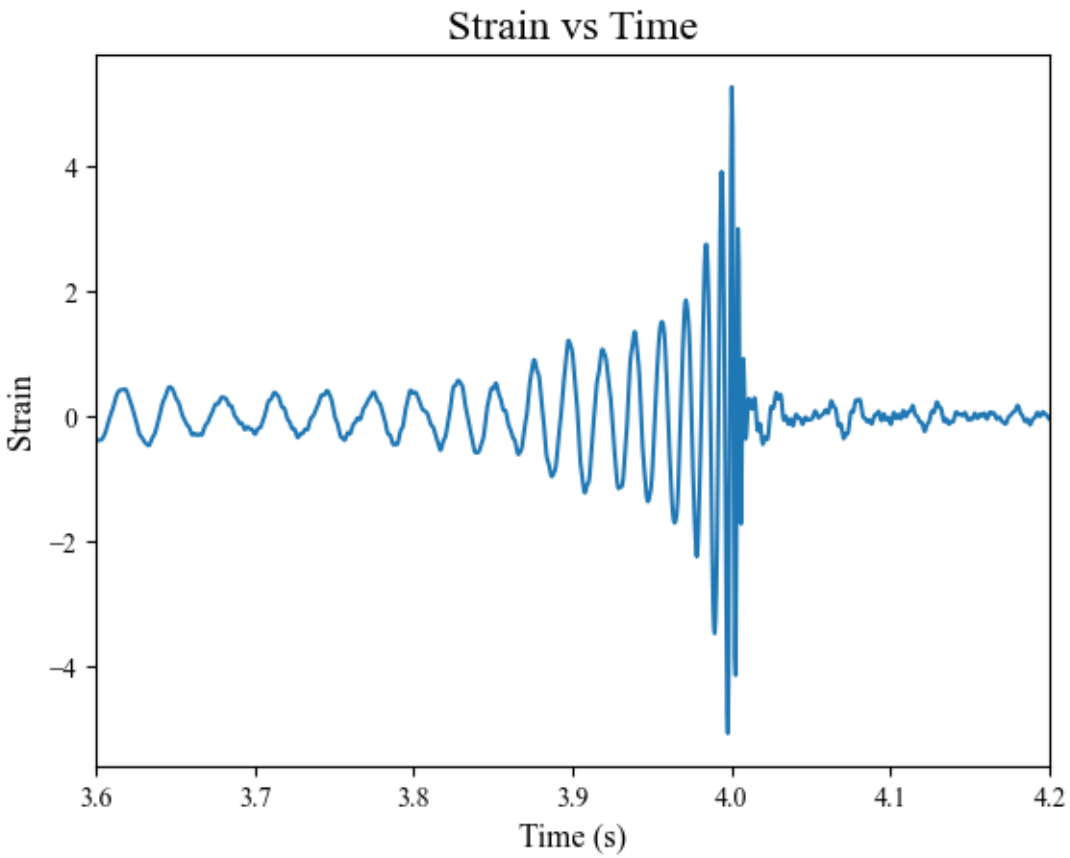

Strain vs Time
Strain
Time (s)
10
5
0
−5
−10
−15
3.6
3.7
3.8
3.9
4.0
4.1
4.2

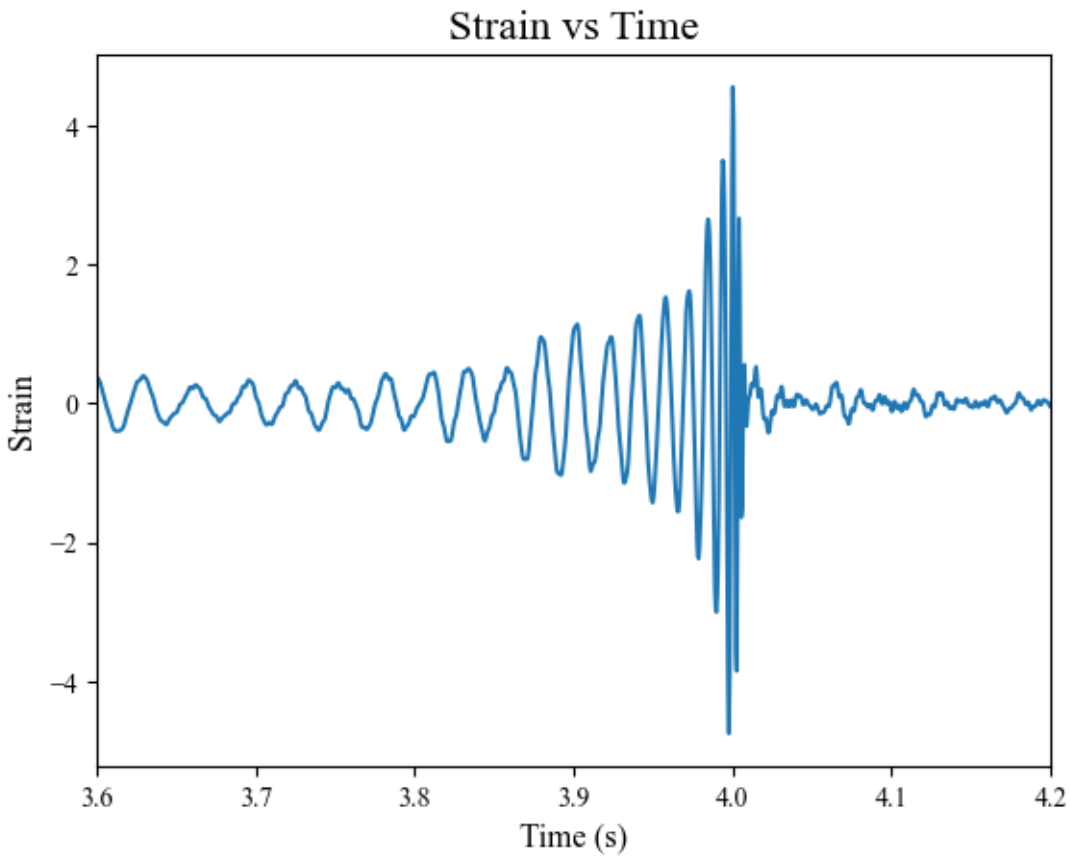

Strain vs Time
Strain
6
4
2
0
−2
−4
−6
−8
3.6
3.7
3.8
3.9
4.0
4.1
4.2
Time (s)

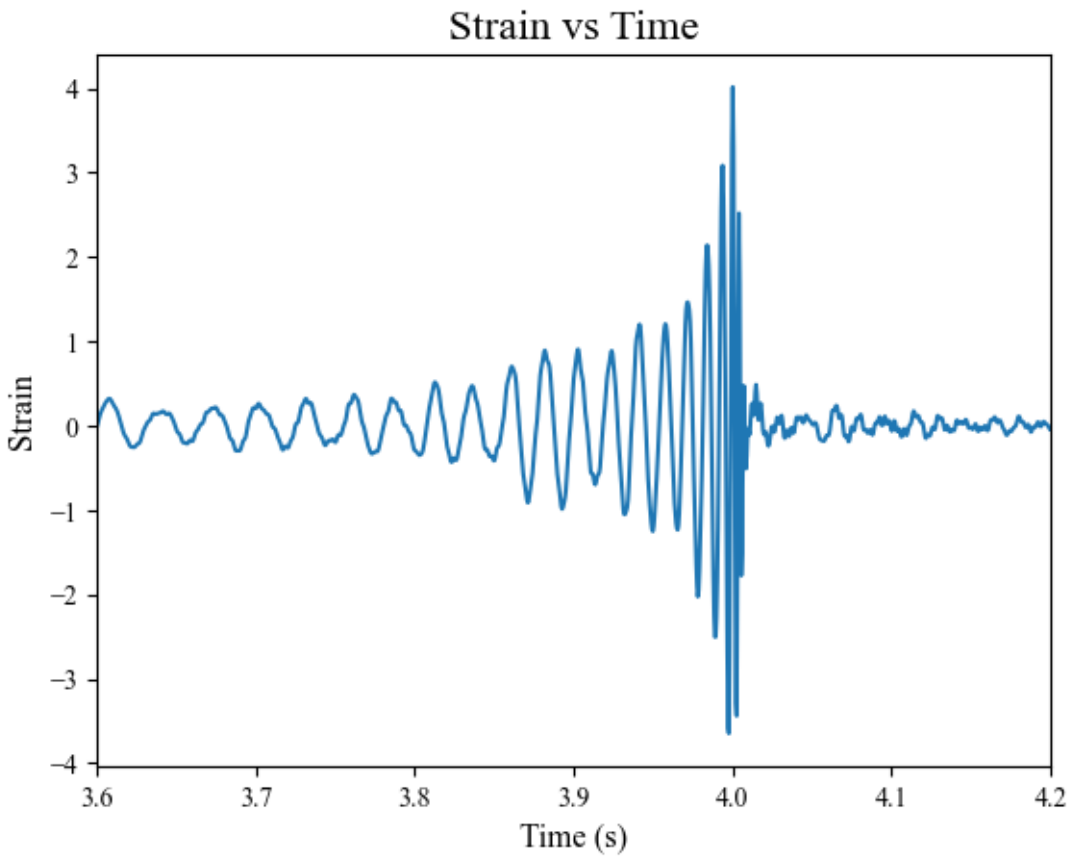

Strain vs Time
Strain
Time (s)
6
4
2
0
−2
−4
−6
3.6
3.7
3.8
3.9
4.0
4.1
4.2

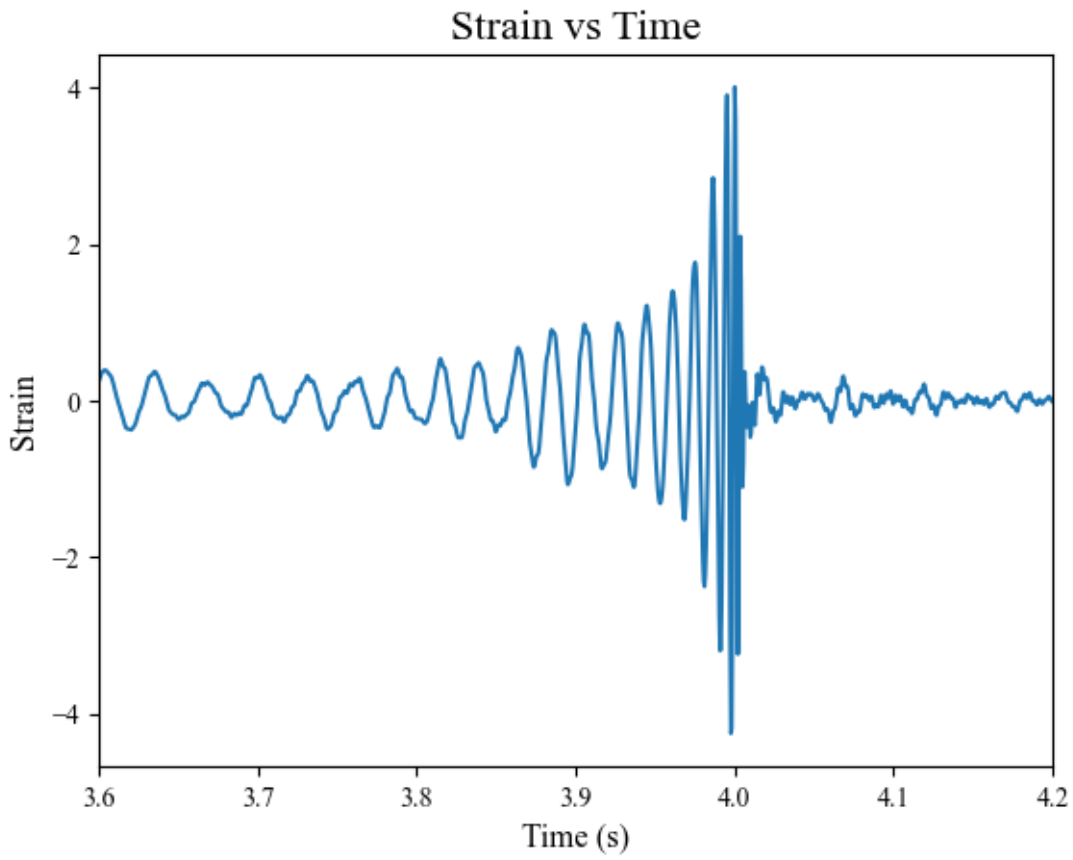

Strain vs Time
Strain
Time (s)
6
4
2
0
−2
−4
−6
3.6
3.7
3.8
3.9
4.0
4.1
4.2


Strain vs Time
Strain
Time (s)
6
4
2
0
−2
−4
3.6
3.7
3.8
3.9
4.0
4.1
4.2


Strain vs Time
Strain
4
2
0
−2
−4
3.6
3.7
3.8
3.9
4.0
4.1
4.2
Time (s)


Strain vs Time
Strain
Time (s)
4
2
0
−2
−4
3.6
3.7
3.8
3.9
4.0
4.1
4.2


Strain vs Time
Strain
Time (s)
4
3
2
1
0
−1
−2
−3
−4
3.6
3.7
3.8
3.9
4.0
4.1
4.2

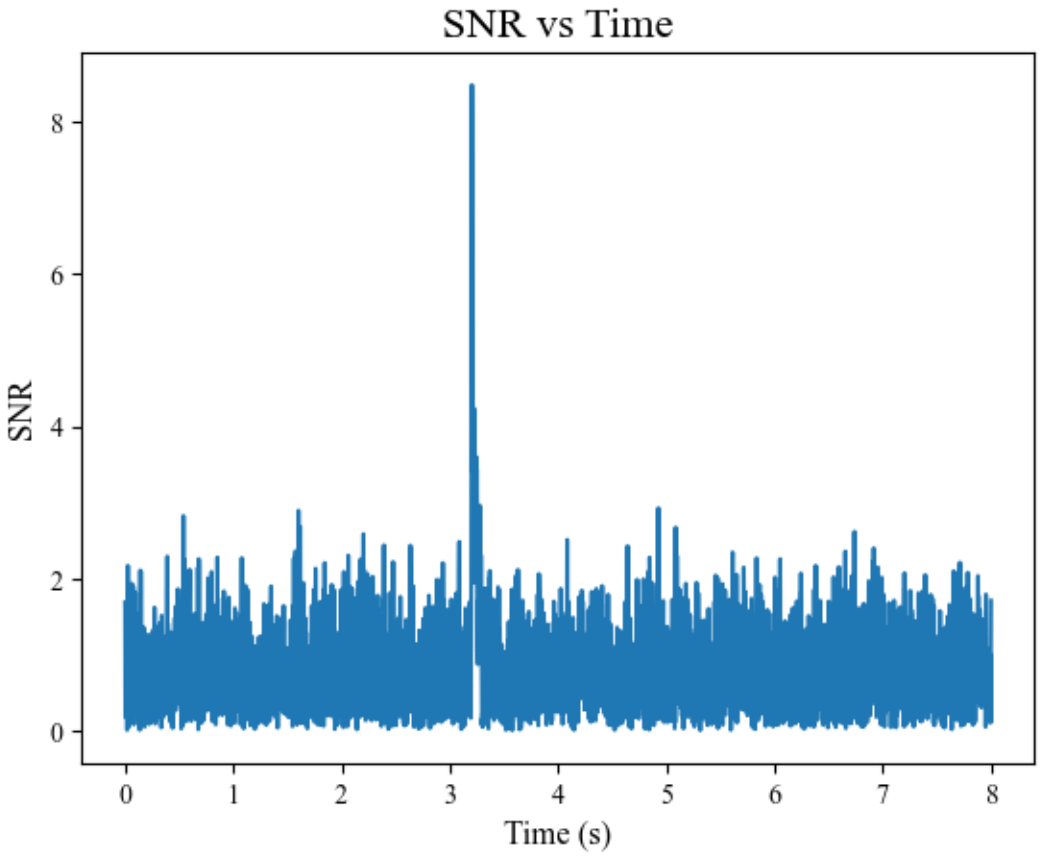


2 Task 2

As distance increases the amplitude of the wave decreases. not much changes for masses that are simlar or different

[224]:
```
snr_ts = gw.get_snr(strain, template, 2048)
plt.figure()
plt.plot(time,snr_ts)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.ylabel('SNR', fontsize=axis_size)
plt.title('SNR vs Time', fontsize=title_size)

plt.show()
print(np.max(snr_ts))
print(np.where(np.max(snr_ts) == snr_ts))
print(snr_ts[np.where(np.max(snr_ts) == snr_ts)],time[np.where(np.max(snr_ts)␣
 ↪== snr_ts)])

min_time = time[np.where(np.max(snr_ts) == snr_ts)]-0.05
```

```
max_time = time[np.where(np.max(snr_ts) == snr_ts)]+0.05
```



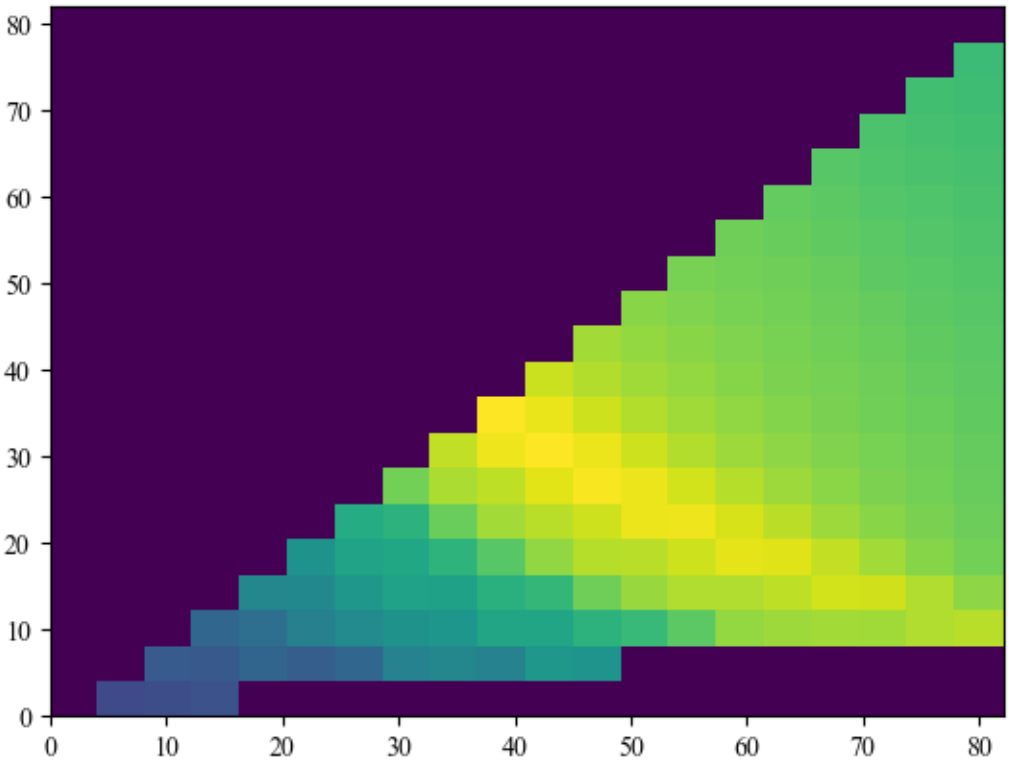

```
8.482283713317145
(array([6561]),)
[8.48228371] [3.20361328]
```

2.0.1 Part a

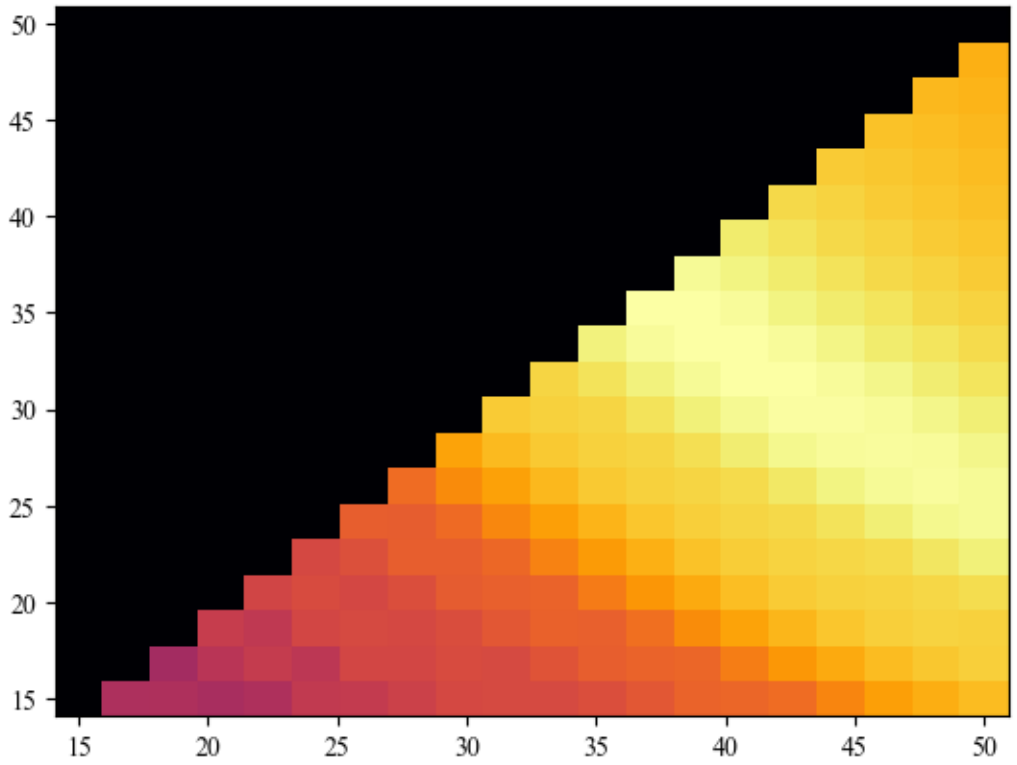
[225]:
```
masses = np.linspace(2,80,20)
currentmax = 0
highest_snr = [[0 for x in range(13)] for y in range(13)]
mass1 = []
mass2 = []
allsnr = []

for m1 in masses:
    for m2 in masses:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
            snr_ts = gw.get_snr(strain, template, 2048)
```

```
            maxsnr = np.max(snr_ts)
            allsnr.append(maxsnr)
            if maxsnr > currentmax:
                currentmax = maxsnr
                M1 = m1
                M2 = m2
        else:
            allsnr.append(0)
print(currentmax, M1, M2)

reshaped = (np.reshape(allsnr, (20,20)).T)
plt.figure()
plt.pcolor(masses, masses, reshaped)
plt.show()

allsnr2 = []
currentmax = 0
masses2 = np.linspace(15,50,20)
maxsnr = []
for m1 in masses2:
    for m2 in masses2:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd)
            snr_ts = gw.get_snr(strain, template, 2048)
            maxsnr = np.max(snr_ts)
            allsnr2.append(maxsnr)
            if maxsnr > currentmax:
                currentmax = maxsnr
                M1 = m1
                M2 = m2
        else:
            allsnr2.append(0)
print(currentmax, M1, M2)



reshaped = (np.reshape(allsnr2, (20,20)).T)
plt.figure()
plt.pcolor(masses2, masses2, reshaped, cmap='inferno')
plt.show()
```

```
13.817943693057638 38.94736842105264 34.8421052631579
```




```
13.826971256612358 37.10526315789474 35.26315789473684
```

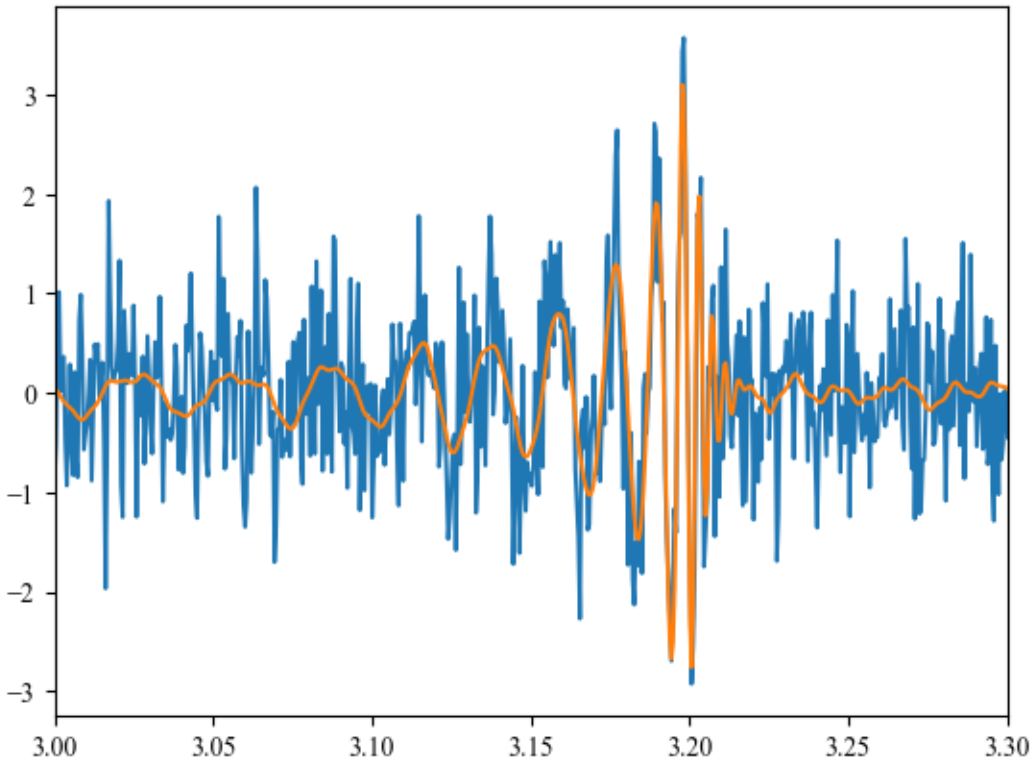


Best fit mass pair for snr result is 36,35 solar masses

2.0.2 Part b

[226]:
```
data = np.loadtxt('strain_data/GW150914_strain.txt')
time = data[:,0]
strain = data[:,1]
inv_psd = np.loadtxt('inv_psd/GW150914_inv_psd.txt',usecols=(1,))
t, template = gw.make_template(36.8,35.5,2048,8,inv_psd,d=1144,tc=3.1978,phic=0)
plt.plot(time, strain)
plt.plot(t, template)
plt.xlim(3,3.3)
```

[226]:
```
(3.0, 3.3)
```



[227]:
```
mass1 = M1
mass2 = M2
m1 = M1
m2 = M2

#generating the template using these masses
t, template = gw.make_template(m1,m2,2048,8,inv_psd)

#generating the snr timeseries
snr_ts = gw.get_snr(strain,template,2048)

def make_signal(t,d,tc,phic):
    _,signal = gw.make_template(mass1,mass2,2048,8,inv_psd,d=d,tc=tc,phic=phic)
    return signal
t = 8
# distances = np.array(np.linspace(100,10000,10000))
tc_min = min_time
tc_max = max_time
print(time[(np.where(np.max(snr_ts) == snr_ts),)])
phic_min = 0
phic_max = 2*np.pi
d_min = 0
```

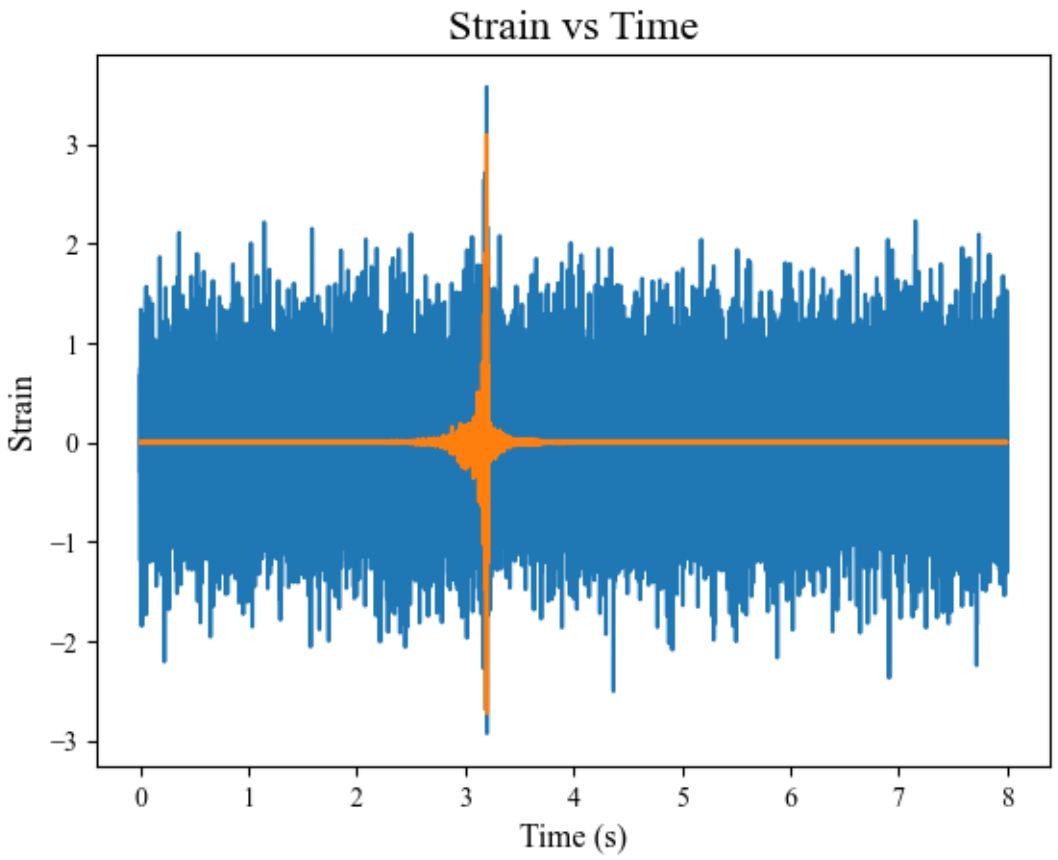
```
d_max = 10000
#p0 = [890,(time[np.where(np.max(snr_ts) == snr_ts)]),5]
p0 = np.array([1000,time[(np.where(np.max(snr_ts) == snr_ts),)],0.5])

ht_err = np.array(np.ones(strain.size))
b = np.array([[d_min,tc_min,phic_min],[d_max,tc_max,phic_max]], dtype = object)
popt, pcov = curve_fit(make_signal,time ,strain, p0 = np.array([1000,time[(np.
 ↪where(np.max(snr_ts) == snr_ts),)],0.5]),bounds = b, sigma=ht_err,␣
 ↪absolute_sigma=True)

t, template = gw.make_template(m1,m2,2048,8,inv_psd,d = popt[0],tc =␣
 ↪popt[1],phic = popt[2])

print(f'The best fit parameters for distance, time, and phase are {popt[0]},␣
 ↪{popt[1]}, and {popt[2]} respectively.')
plt.figure()
plt.plot(time, strain)
plt.plot(t, template)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.ylabel('Strain', fontsize=axis_size)
plt.title('Strain vs Time', fontsize=title_size)
plt.show()

# m1 = M1
# m2 = M2

# #generating the template using these masses
# t, template = gw.make_template(m1,m2,2048,8,inv_psd)

# #generating the snr timeseries
# snr_ts = gw.get_snr(strain,template,2048)

# # without above get wrong answer

# d_est = 1000 # Mpc
# tc_est = time[np.argmax(np.absolute(snr_ts))] # tc estimate at peak of snr␣
 ↪timeseries
# phic_est = 0.5 # radians

# p0 = [d_est, tc_est, phic_est]

# # bounds on the fitting parameters
# d_min, d_max = [0, 10000]
# tc_min, tc_max = [tc_est-0.05, tc_est+0.05]
# phic_min, phic_max = [0,2*np.pi]

# ht_err = np.ones(strain.size)
```

```
# b = [[d_min,tc_min,phic_min,],[d_max,tc_max,phic_max]]
# popt, pcov =␣
 ↪curve_fit(make_signal,time,strain,p0,bounds=b,sigma=ht_err,absolute_sigma=True)

# dist, timeco, phico = popt # fitted distance, time of coalescence and phase␣
 ↪of coalescence parametes

# d_err, tc_err, phic_err = np.sqrt(np.diag(pcov)) # errors on these values

# print(f'Distance: {dist:0.2f} \u00B1 {d_err:0.2f}Mpc')
# print(f'Time of Coalescence:: {timeco:0.4f} \u00B1 {tc_err:0.4f}s')
# print(f'Phase at Coalescence:: {phico:0.1f} \u00B1 {phic_err:0.1f}1')
```

```
[[3.19824219]]
```

```
/var/folders/tn/fqh6631n3p56r504tcsqvss00000gn/T/ipykernel_91564/1711584164.py:2
5: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  p0 = np.array([1000,time[(np.where(np.max(snr_ts) == snr_ts),)],0.5])
/var/folders/tn/fqh6631n3p56r504tcsqvss00000gn/T/ipykernel_91564/1711584164.py:2
9: VisibleDeprecationWarning: Creating an ndarray from ragged nested sequences
(which is a list-or-tuple of lists-or-tuples-or ndarrays with different lengths
or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  popt, pcov = curve_fit(make_signal,time ,strain, p0 =
np.array([1000,time[(np.where(np.max(snr_ts) == snr_ts),)],0.5]),bounds = b,
sigma=ht_err, absolute_sigma=True)
```

```
The best fit parameters for distance, time, and phase are 1151.1513129536315,
3.1977281585687707, and 5.69828940481586e-16 respectively.
```



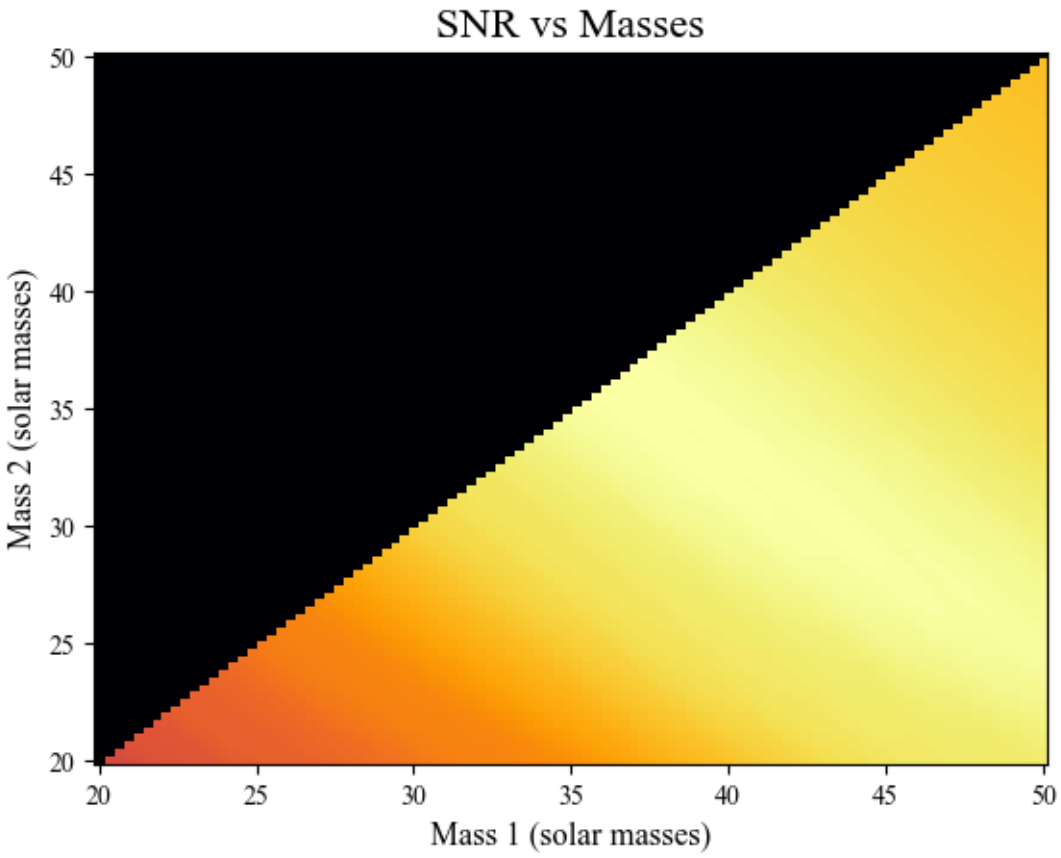
[230]:
```
path = 'strain_data/'
files = np.sort(os.listdir(path))

distance_coel = []
time_coel = []
phase_coel = []
larger_masses = []
smaller_masses = []
names = []
name = ('GW150914', 'GW170823', 'GW170814', 'GW170104', 'GW151226', 'GW151012',
'GW170729', 'GW170809', 'GW170818')

i = 0
j = 0

for file in files:
    filename = path+file
    print('-------------------------------------------------------------')
    print(f'The Binary Black Hole merger event {file}')
    if file.startswith('GW170817'):
```

```
        pass
    else:
        names.append(file)
        if file.endswith('strain.txt'):
            data = np.loadtxt(filename)
            time = data[:,0]
            strain = data[:,1]
        for data in os.listdir('inv_psd/'):
            if data.endswith('inv_psd.txt'):
                inv_psd_data = np.loadtxt('inv_psd/'+ data)
                inv_psd2 = inv_psd_data[:,1]

        t, template = gw.make_template(37,36,2048,8,inv_psd2,400)
        snr_ts = gw.get_snr(strain, template, 2048)
        min_time = time[np.where(np.max(snr_ts) == snr_ts)]-0.05
        max_time = time[np.where(np.max(snr_ts) == snr_ts)]+0.05

        masses = np.linspace(2,80,20)
        currentmax = 0
        highest_snr = [[0 for x in range(13)] for y in range(13)]
        mass1 = []
        mass2 = []
        allsnr = []

        for m1 in masses:
            for m2 in masses:
                if m1>m2 and (m1/m2) < 8:
                    t, template = gw.make_template(m1,m2,2048,8,inv_psd2)
                    snr_ts = gw.get_snr(strain, template, 2048)
                    maxsnr = np.max(snr_ts)
                    allsnr.append(maxsnr)
                    if maxsnr > currentmax:
                        currentmax = maxsnr
                        M1 = m1
                        M2 = m2

                else:
                    allsnr.append(0)
        reshaped = (np.reshape(allsnr, (20,20)).T)

        # plt.figure()
        # plt.title(f'SNR vs Masses {file}', fontsize=title_size)
        # plt.xlabel('Mass 1 (solar masses)', fontsize=axis_size)
        # plt.ylabel('Mass 2 (solar masses)', fontsize=axis_size)
        # plt.pcolor(masses, masses, reshaped)
        # plt.show()
```

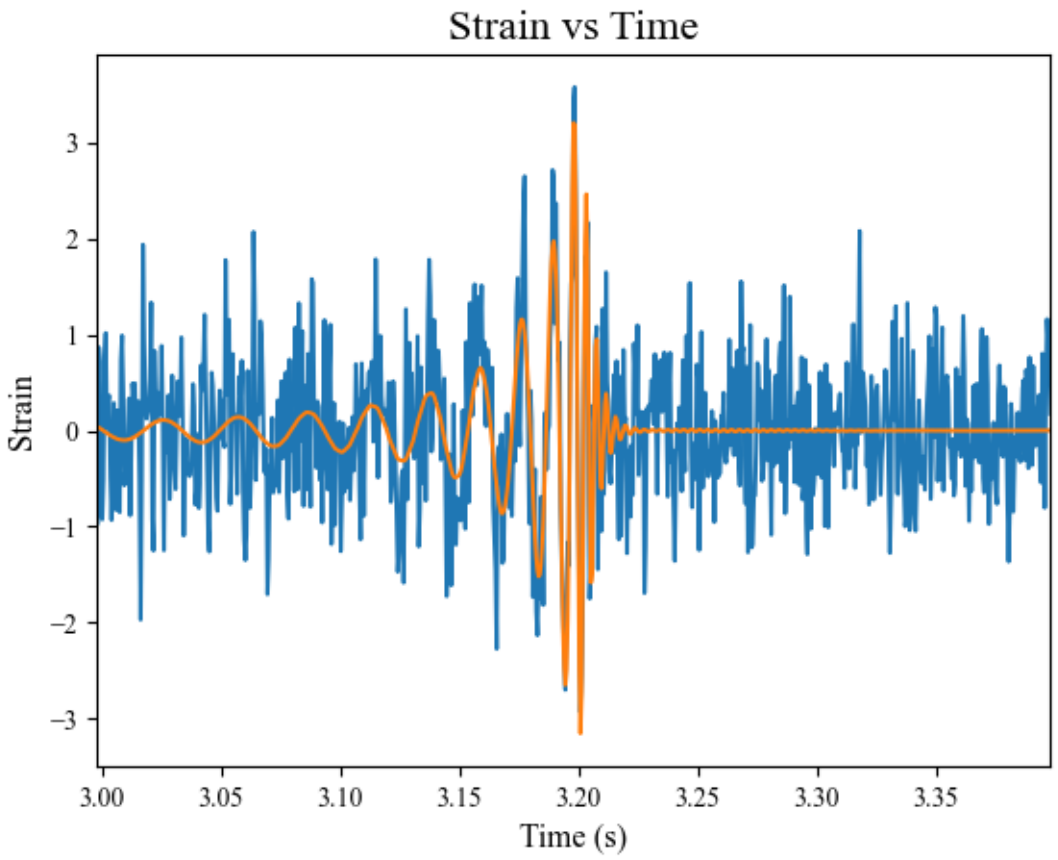
```
set_mass1 = [50, 40, 20, 50, 20, 80, 55, 50, 60, 80]
set_mass2 = [20, 5, 3, 10, 2, 20, 5, 15, 3, 10]
masses2 = []
# for i in set_mass1:
#     for j in set_mass2:
masses2 = np.linspace(set_mass1[i],set_mass2[j],100)

i = i + 1
j = j + 1

allsnr2 = []
currentmax = 0
# masses2 = np.linspace(15,50,20)
maxsnr = []
for m1 in masses2:
    for m2 in masses2:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd2,400)
            snr_ts = gw.get_snr(strain, template, 2048)
            maxsnr = np.max(snr_ts)
            allsnr2.append(maxsnr)
            if maxsnr > currentmax:
                currentmax = maxsnr
                M1 = m1
                M2 = m2
        else:
            allsnr2.append(0)

reshaped = (np.reshape(allsnr2, (100,100)).T)
plt.figure()
plt.title('SNR vs Masses', fontsize=title_size)
plt.xlabel('Mass 1 (solar masses)', fontsize=axis_size)
plt.ylabel('Mass 2 (solar masses)', fontsize=axis_size)
plt.pcolor(masses2, masses2, reshaped, cmap='inferno')
plt.show()

mass1 = M1
mass2 = M2
m1 = M1
m2 = M2
t, template = gw.make_template(m1,m2,2048,8,inv_psd2)

snr_ts = gw.get_snr(strain, template, 2048)
min_time = time[np.where(np.max(snr_ts) == snr_ts)]-0.05
max_time = time[np.where(np.max(snr_ts) == snr_ts)]+0.05

#generating the snr timeseries
```

```
        snr_ts = gw.get_snr(strain,template,2048)

        def make_signal(t,d,tc,phic):
            _,signal = gw.
↪make_template(M1,M2,2048,8,inv_psd,d=d,tc=tc,phic=phic)
            return signal
        t = 8
        distances = np.array(np.linspace(100,10000,10000))
        tc_min = min_time
        tc_max = max_time
        phic_min = 0
        phic_max = 2*np.pi
        d_min = 100
        d_max = 4500

        print(f'The best fit masses for {file} are {M1} and {M2} respectively.')
        print(time[(np.where(np.max(snr_ts) == snr_ts),)])
        ht_err = np.array(np.ones(strain.size))
        b = np.array([[d_min,tc_min,phic_min],[d_max,tc_max,phic_max]], dtype =␣
↪object)
        t_est = time[np.argmax(snr_ts)]
        p0 = [1000,t_est,0.5]

        popt, pcov = curve_fit(make_signal,time ,strain, p0 = [1000,time[(np.
↪where(np.max(snr_ts) == snr_ts),)],0.5],bounds = b, sigma=ht_err,␣
↪absolute_sigma=True)

        distance_coel.append(popt[0])
        time_coel.append(popt[1])
        phase_coel.append(popt[2])
        larger_masses.append(M1)
        smaller_masses.append(M2)

        t, template = gw.make_template(m1,m2,2048,8,inv_psd2,d = popt[0],tc =␣
↪popt[1],phic = popt[2])
        lims = [tc_min-0.15, tc_max+0.15]
        plt.figure()
        plt.plot(time, strain)
        plt.plot(t, template)
        plt.xlabel('Time (s)', fontsize=axis_size)
        plt.xlim(lims[0], lims[1])
        plt.ylabel('Strain', fontsize=axis_size)
        plt.title('Strain vs Time', fontsize=title_size)
        plt.show()
        print(f'The best fit parameters for data set {file} for distance, time,␣
↪and phase are {popt[0]}, {popt[1]}, and {popt[2]} respectively.')
        # print(currentmax, M1, M2)
```

```
------------------------------------------------------------
The Binary Black Hole merger event GW150914_strain.txt
```

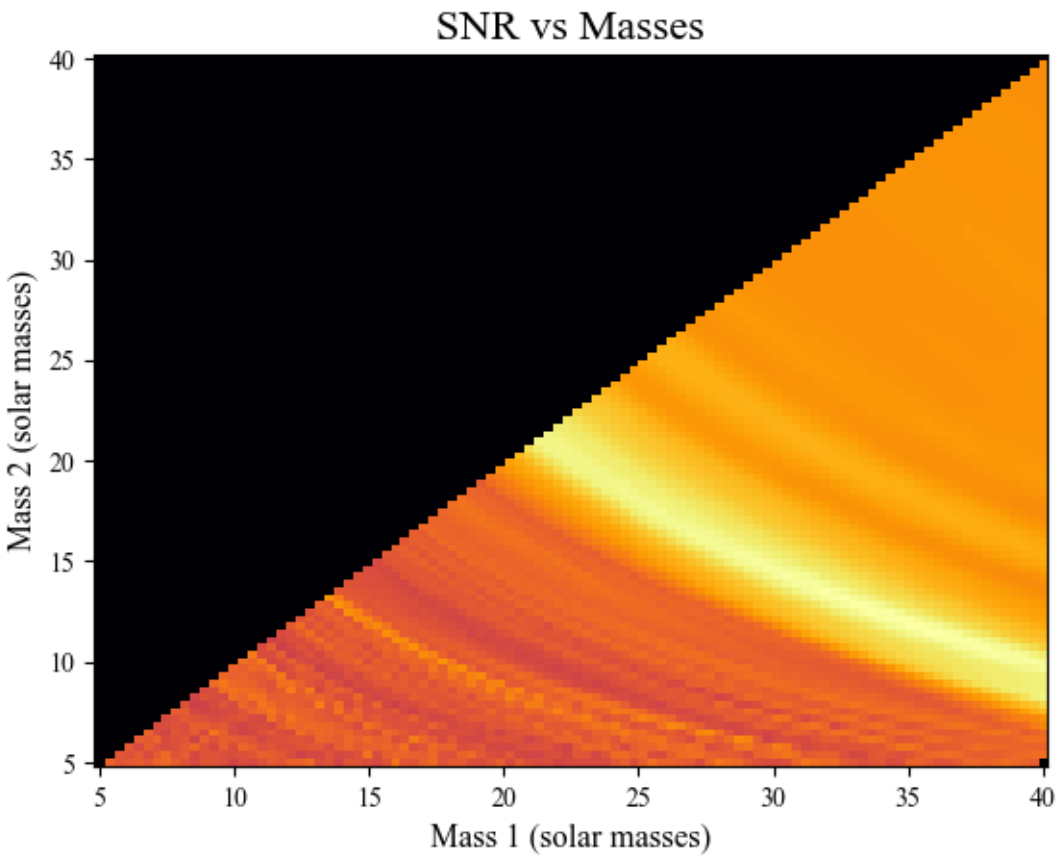



```
The best fit masses for GW150914_strain.txt are 36.06060606060606 and
35.75757575757576 respectively.
[[3.19775391]]
```

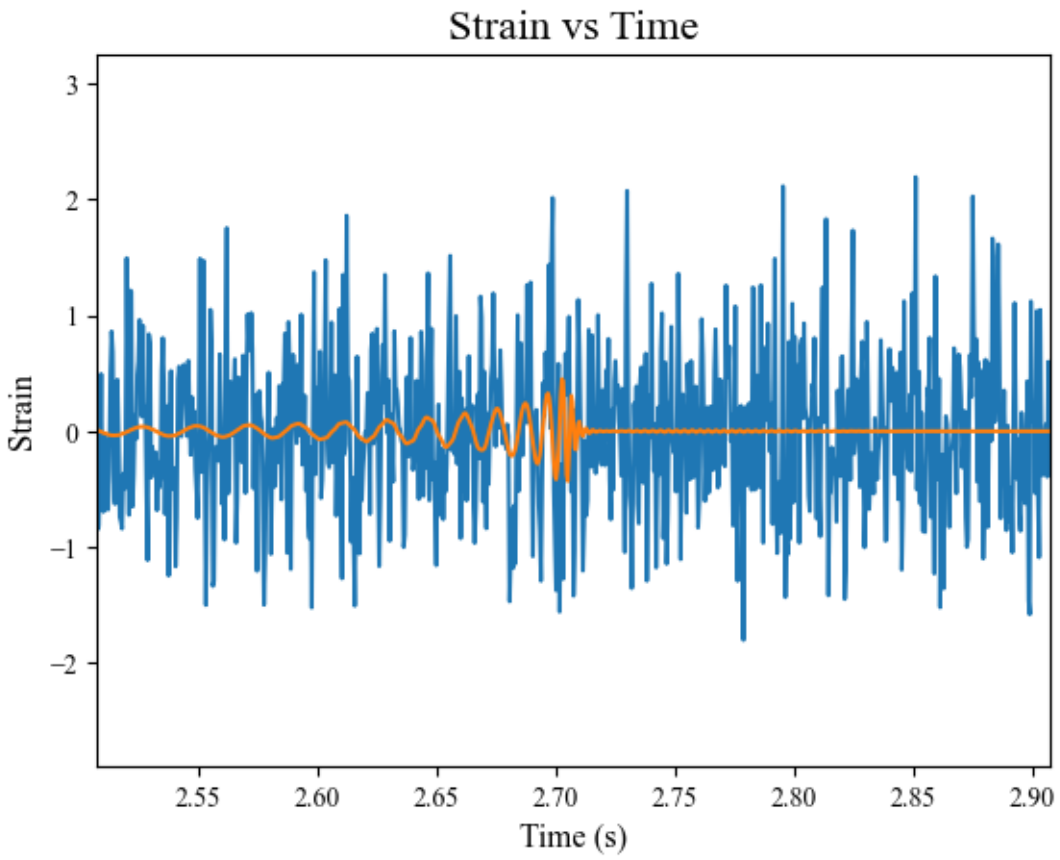


The best fit parameters for data set GW150914_strain.txt for distance, time, and phase are 1127.86146633934, 3.1977708558204716, and 0.030287566634400926 respectively.

The Binary Black Hole merger event GW151012_strain.txt

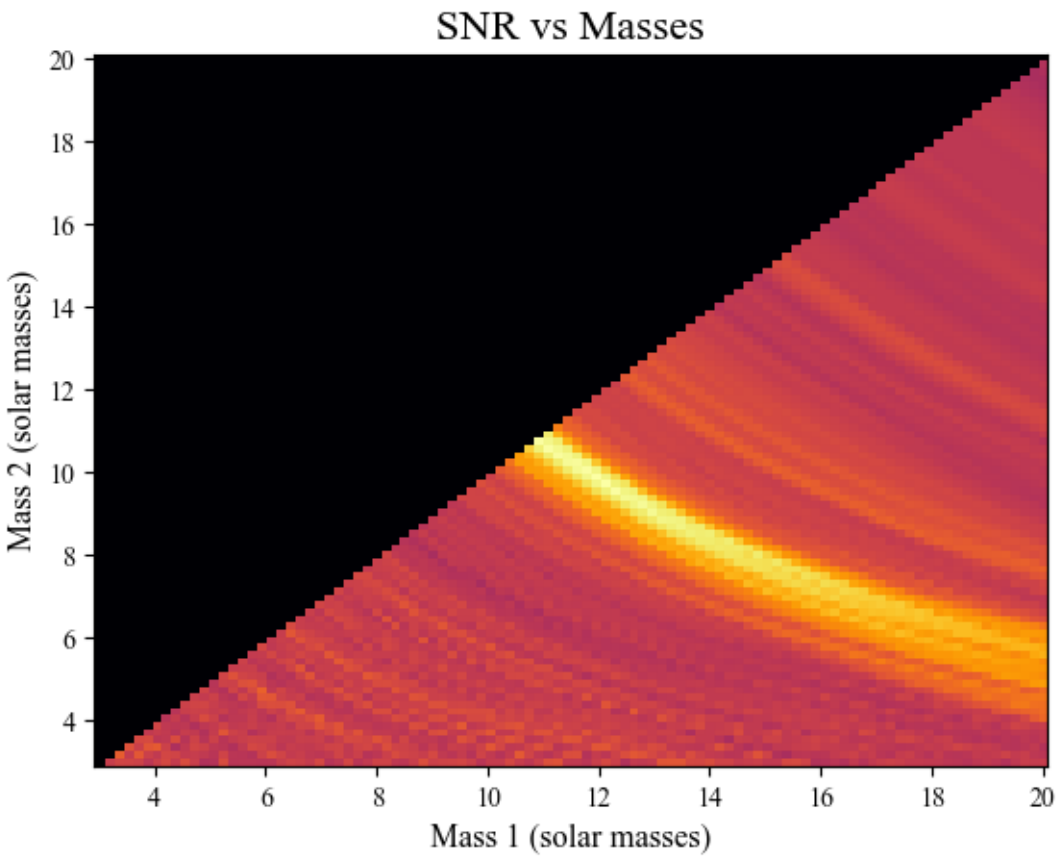



```
The best fit masses for GW151012_strain.txt are 35.4040404040404 and
11.717171717171716 respectively.
[[2.70751953]]
```

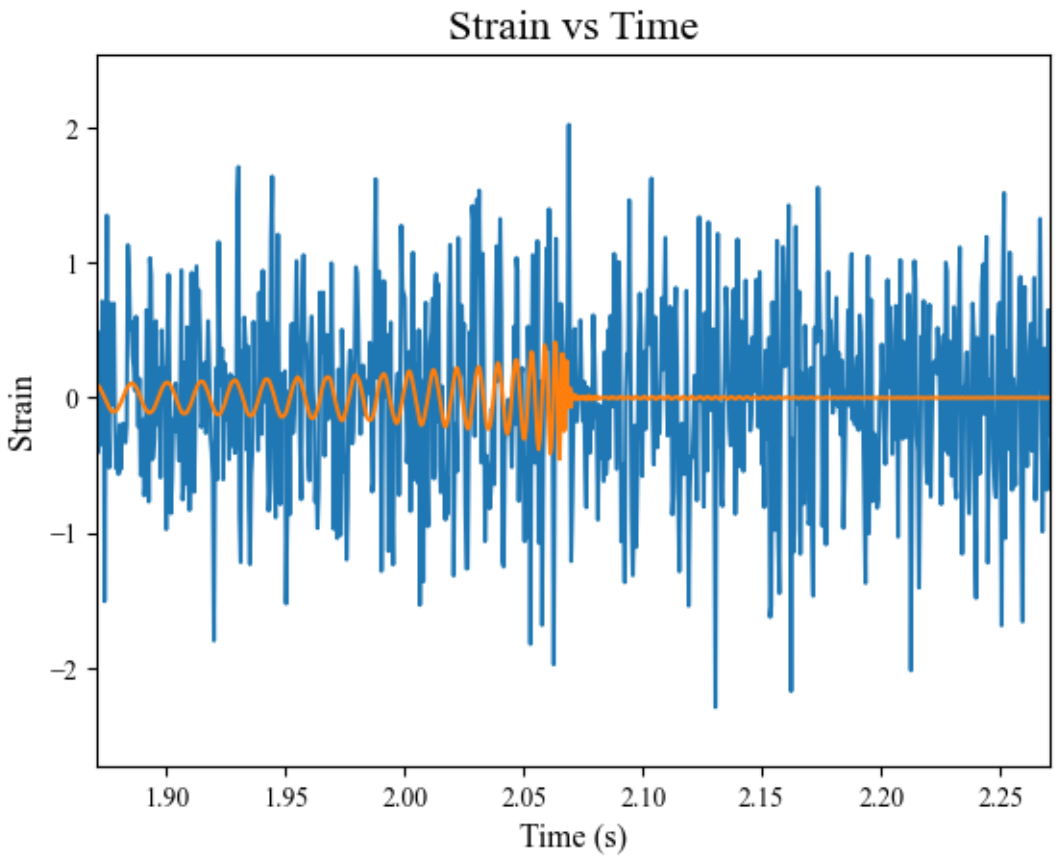


The best fit parameters for data set GW151012_strain.txt for distance, time, and phase are 3782.566299555395, 2.703106823190361, and 8.736722549874701e-20 respectively.

The Binary Black Hole merger event GW151226_strain.txt

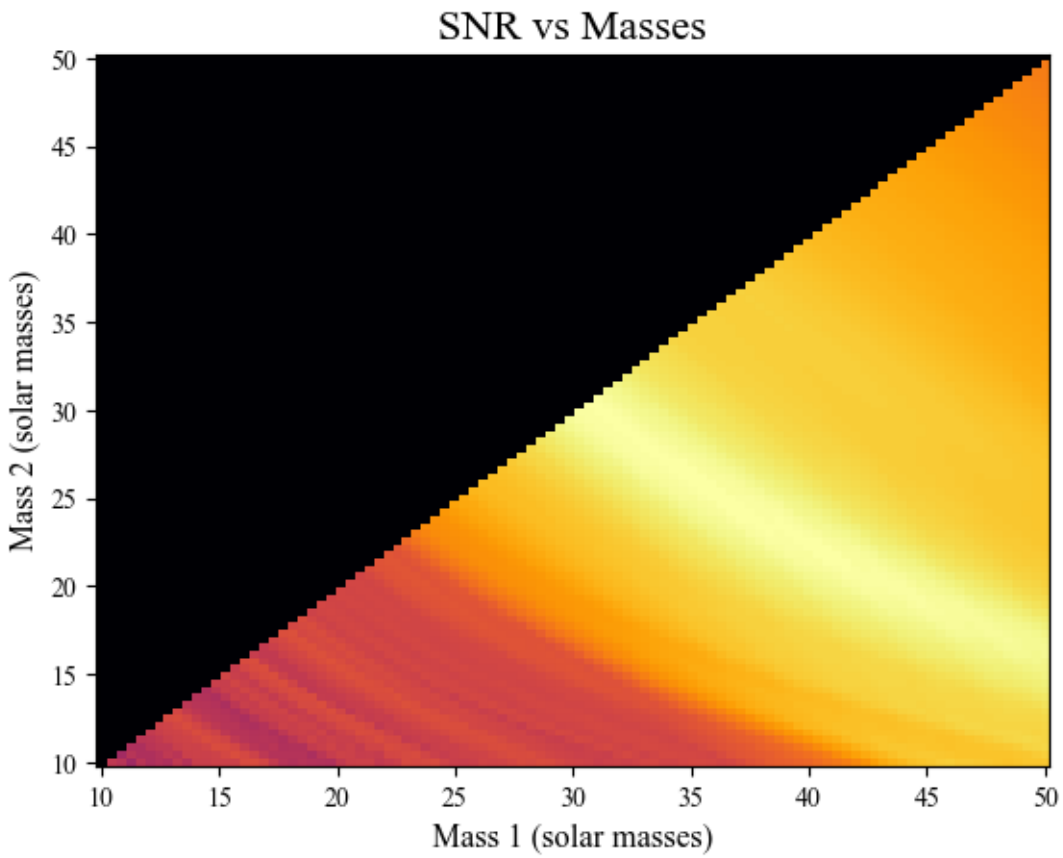



```
The best fit masses for GW151226_strain.txt are 12.1010101010101 and
9.696969696969697 respectively.
[[2.07080078]]
```

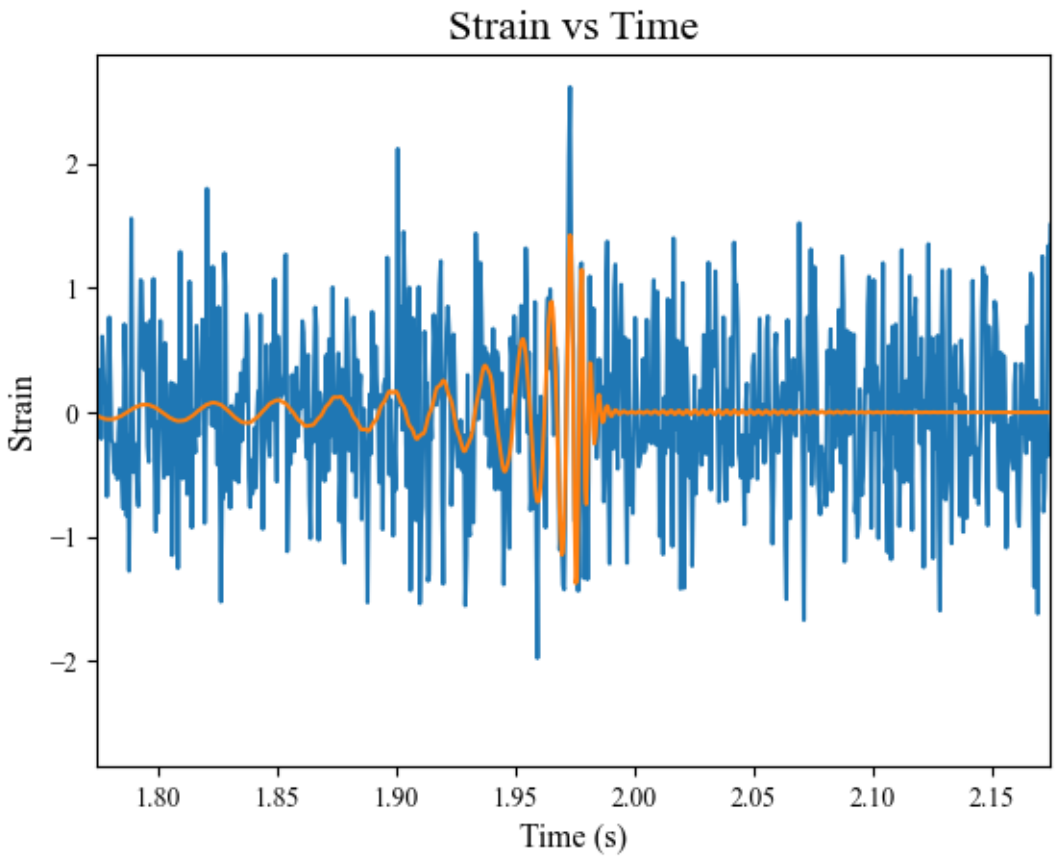


The best fit parameters for data set GW151226_strain.txt for distance, time, and phase are 1811.6016592445792, 2.0671846908126312, and 3.114996141110154e-12 respectively.

The Binary Black Hole merger event GW170104_strain.txt

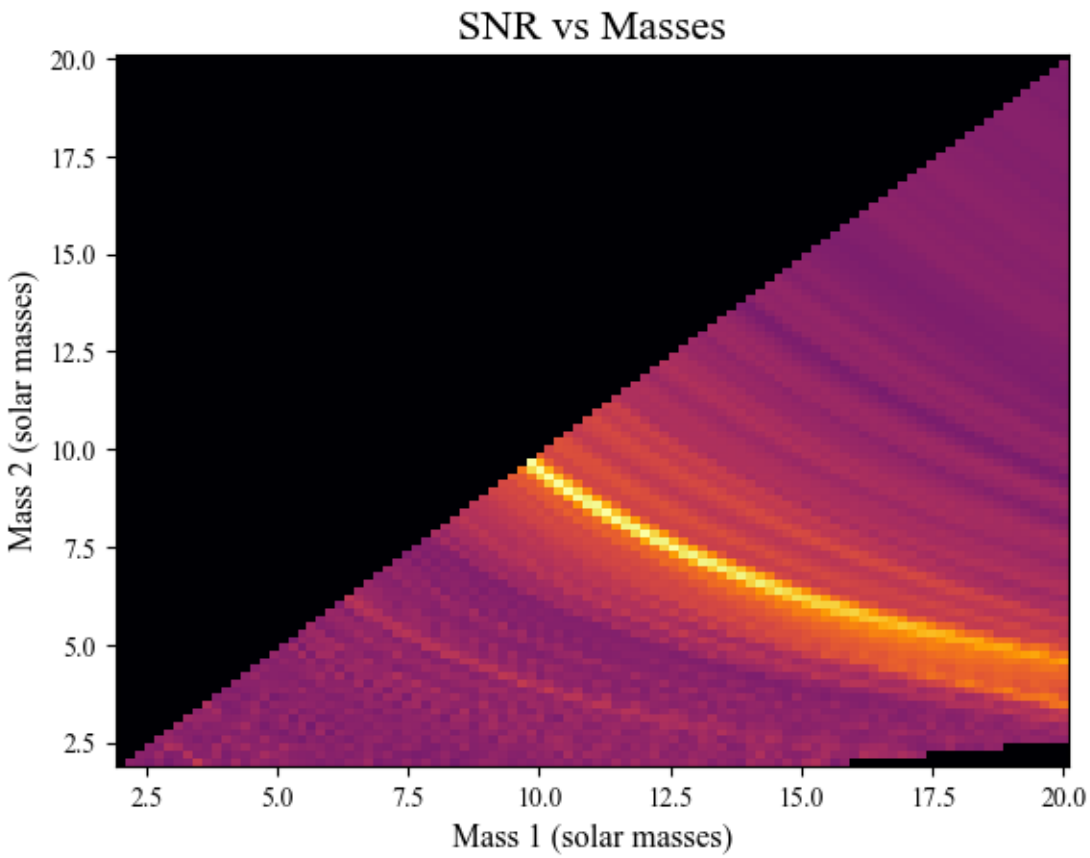



```
The best fit masses for GW170104_strain.txt are 35.858585858585855 and
25.353535353535353 respectively.
[[1.97460938]]
```

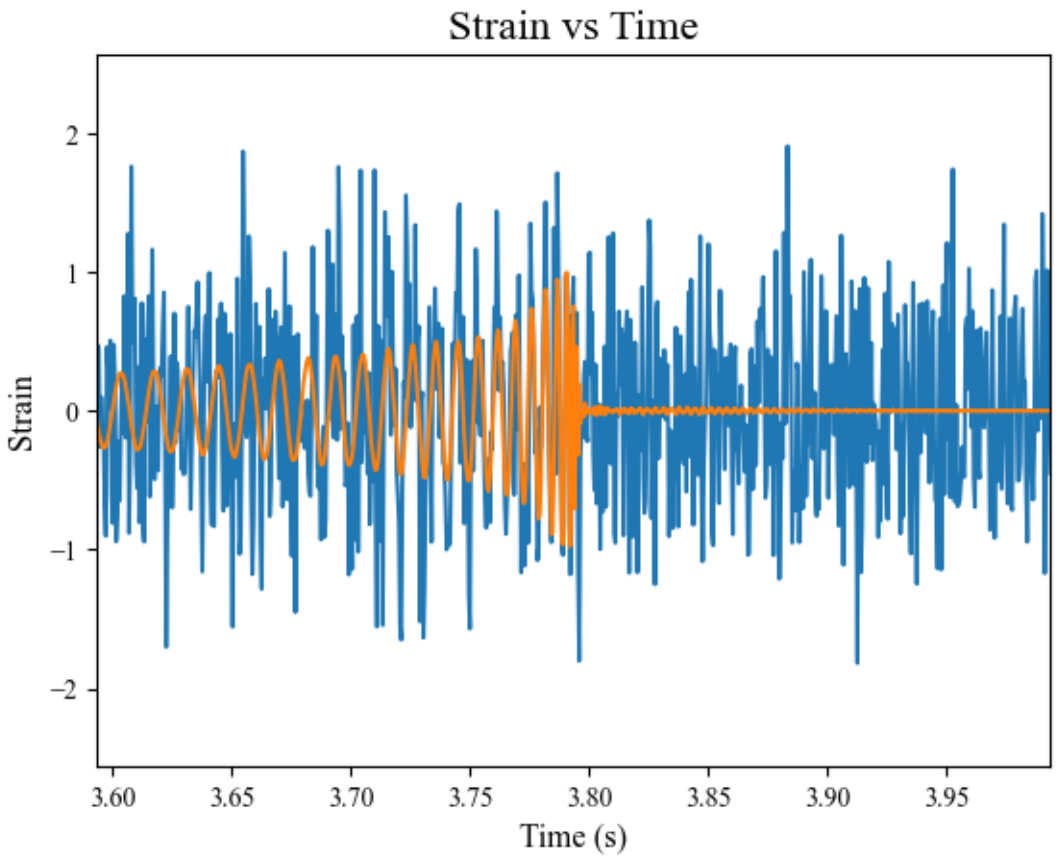


The best fit parameters for data set GW170104_strain.txt for distance, time, and phase are 2119.4825095814767, 1.9745518727859614, and 1.7933210751953568 respectively.

The Binary Black Hole merger event GW170608_strain.txt

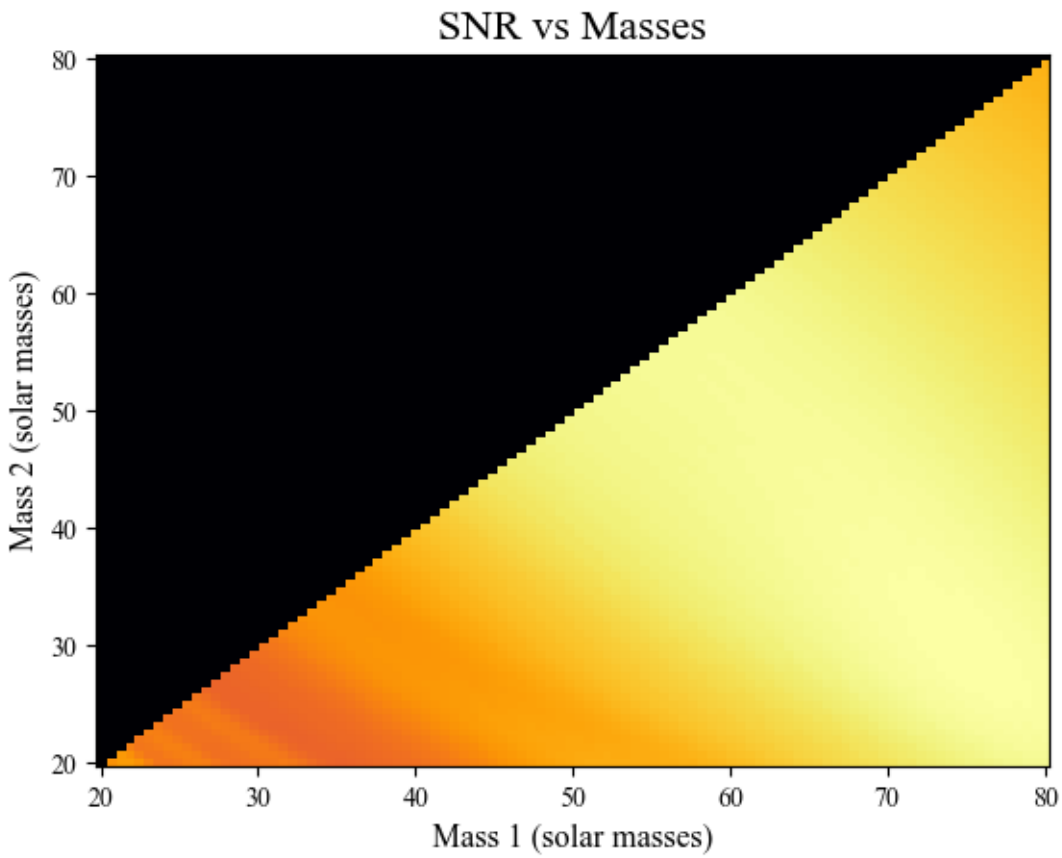



```
The best fit masses for GW170608_strain.txt are 11.272727272727273 and
8.363636363636363 respectively.
[[3.79345703]]
```

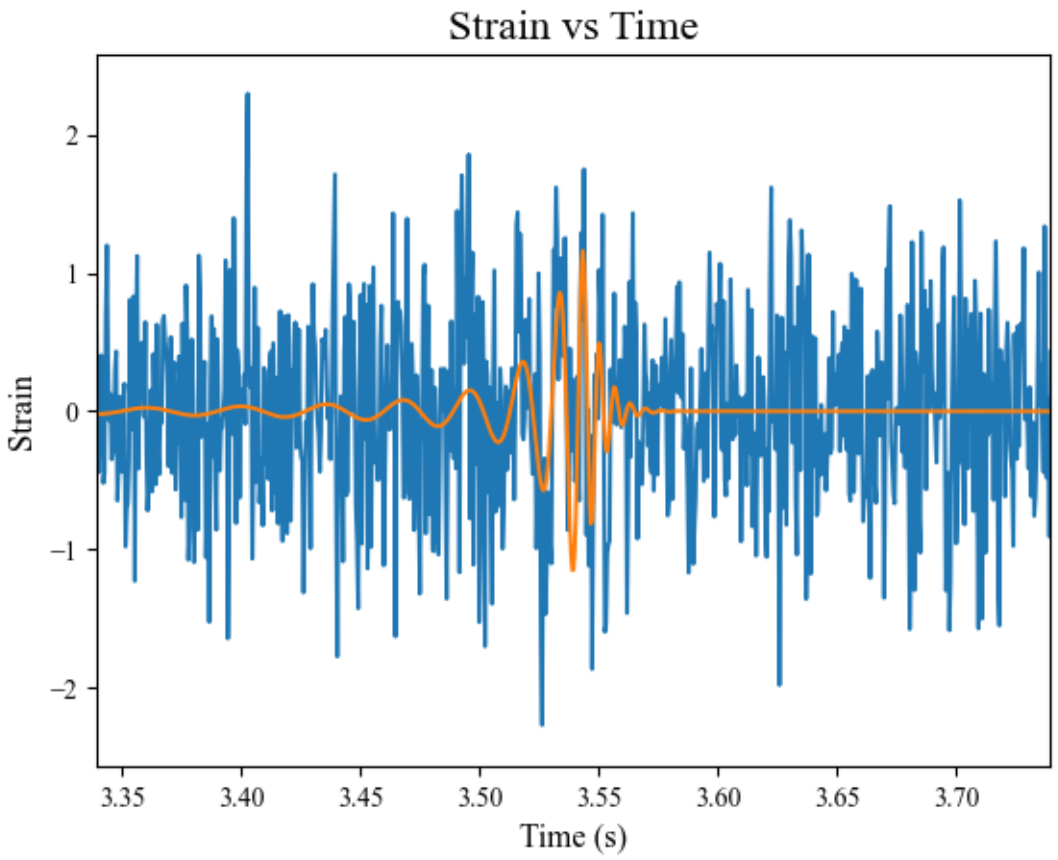


The best fit parameters for data set GW170608_strain.txt for distance, time, and phase are 664.1750278779527, 3.793505875733928, and 2.6743411501251453 respectively.

The Binary Black Hole merger event GW170729_strain.txt

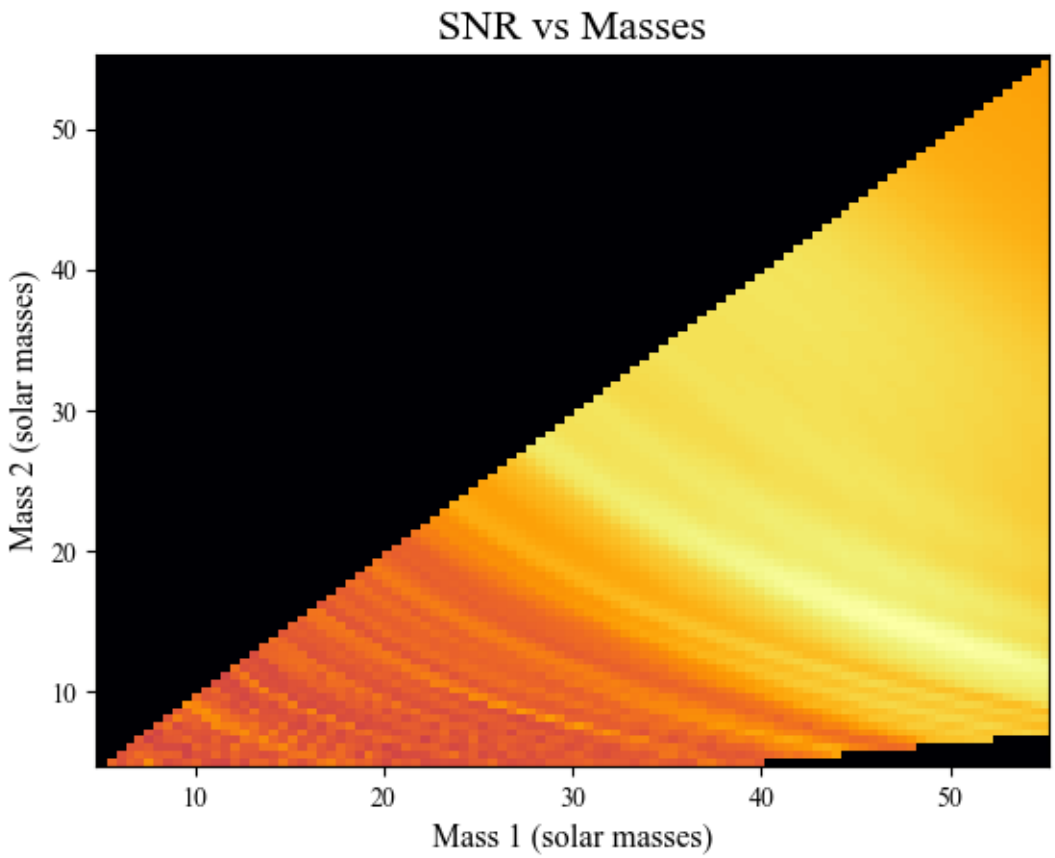



```
The best fit masses for GW170729_strain.txt are 75.15151515151516 and
29.696969696969695 respectively.
[[3.53955078]]
```

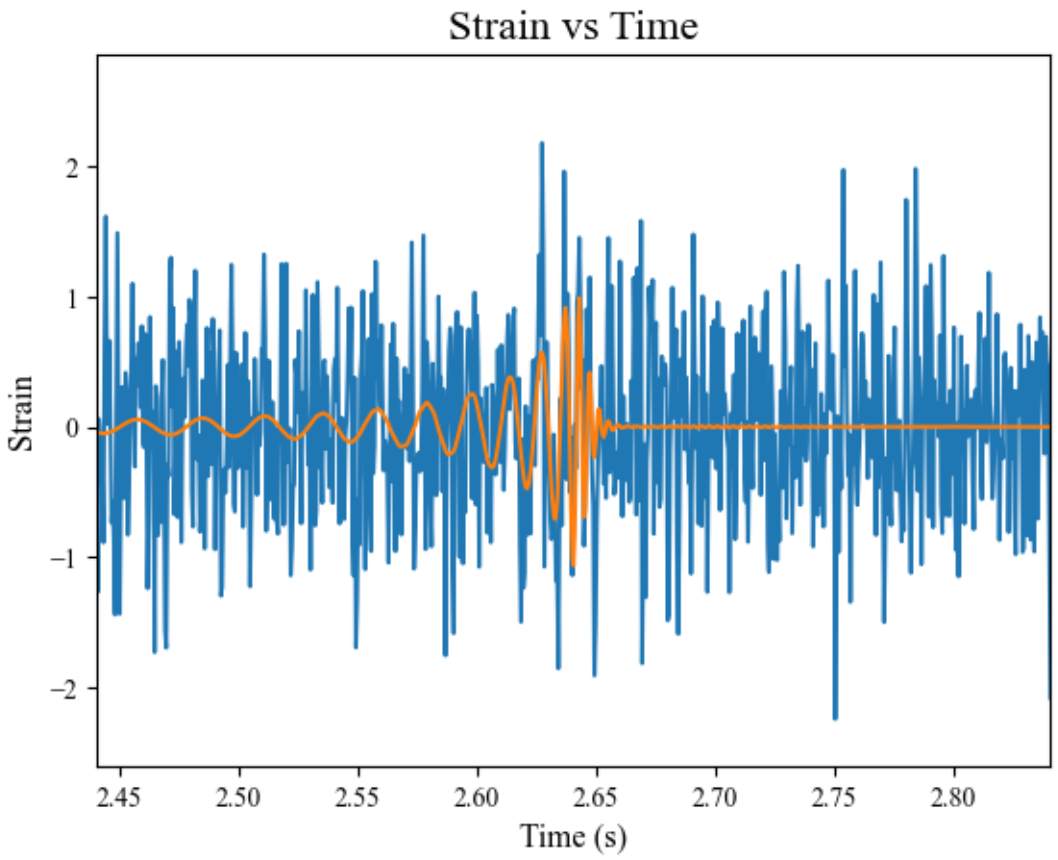


The best fit parameters for data set GW170729_strain.txt for distance, time, and phase are 3150.396091232782, 3.5394629822404595, and 2.8856097578816953 respectively.

The Binary Black Hole merger event GW170809_strain.txt

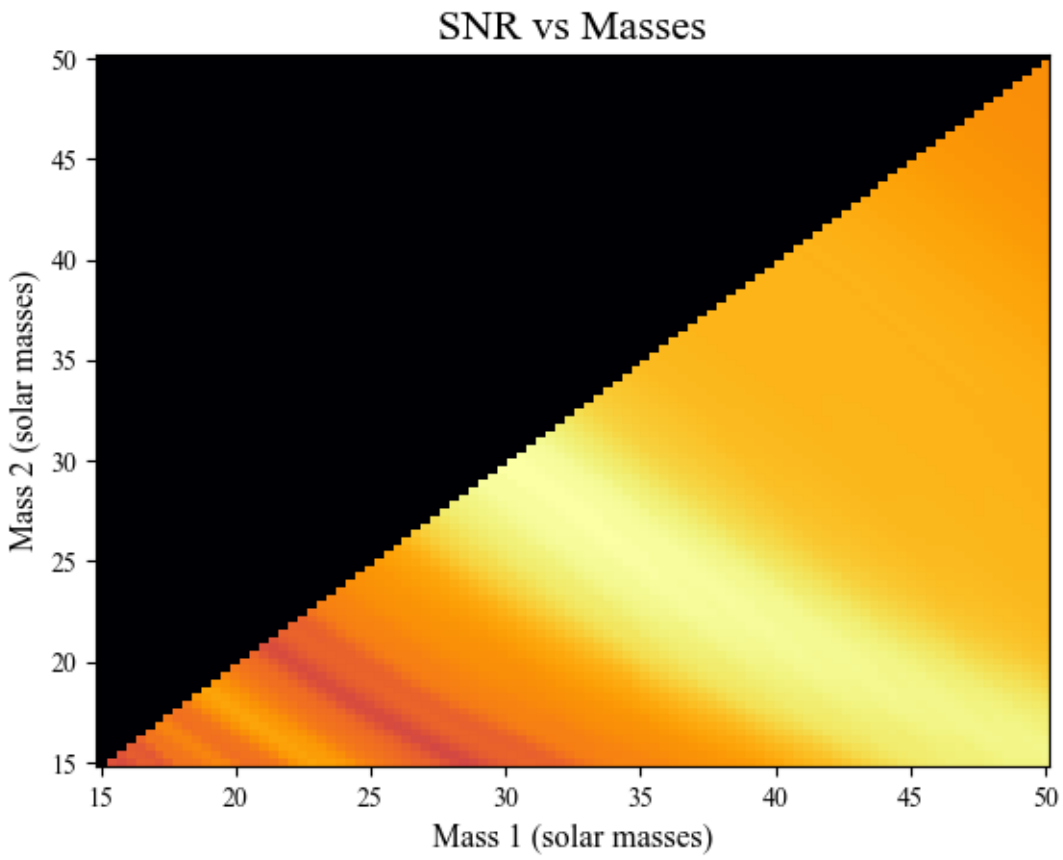



```
The best fit masses for GW170809_strain.txt are 46.41414141414141 and
15.101010101010097 respectively.
[[2.640625]]
```

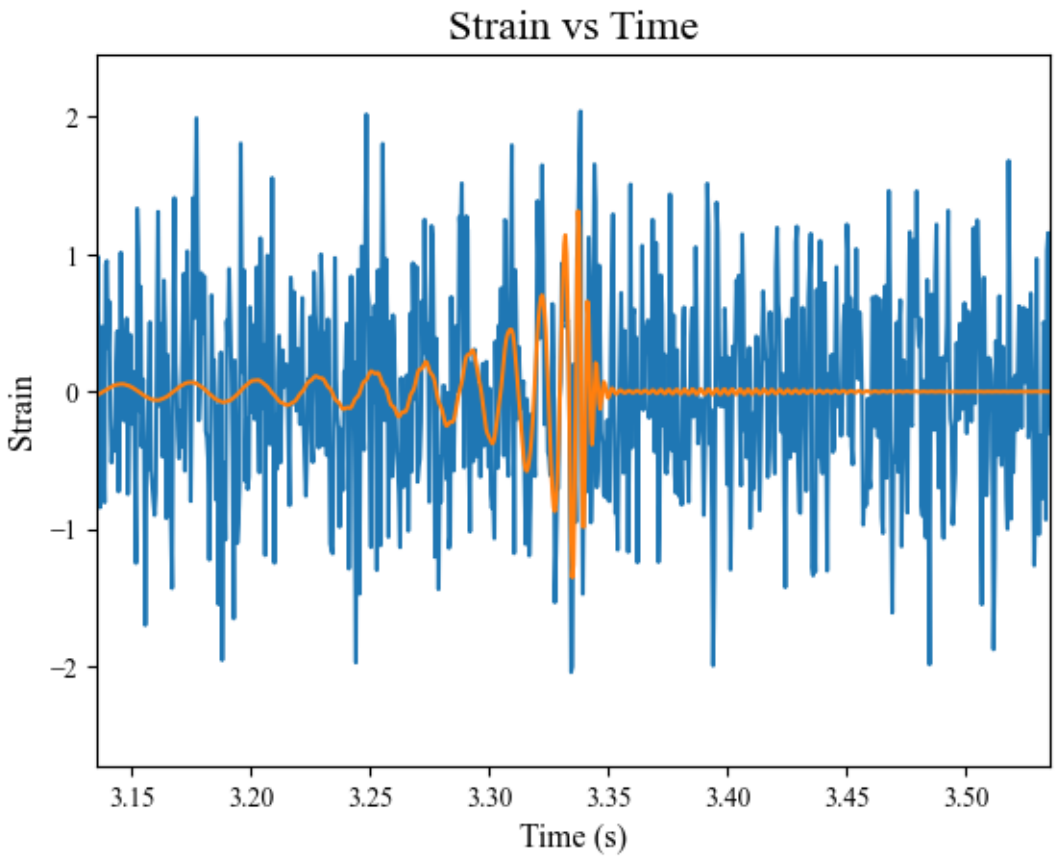


The best fit parameters for data set GW170809_strain.txt for distance, time, and phase are 2145.543375657426, 2.6406004891301014, and 2.7842597216284894 respectively.

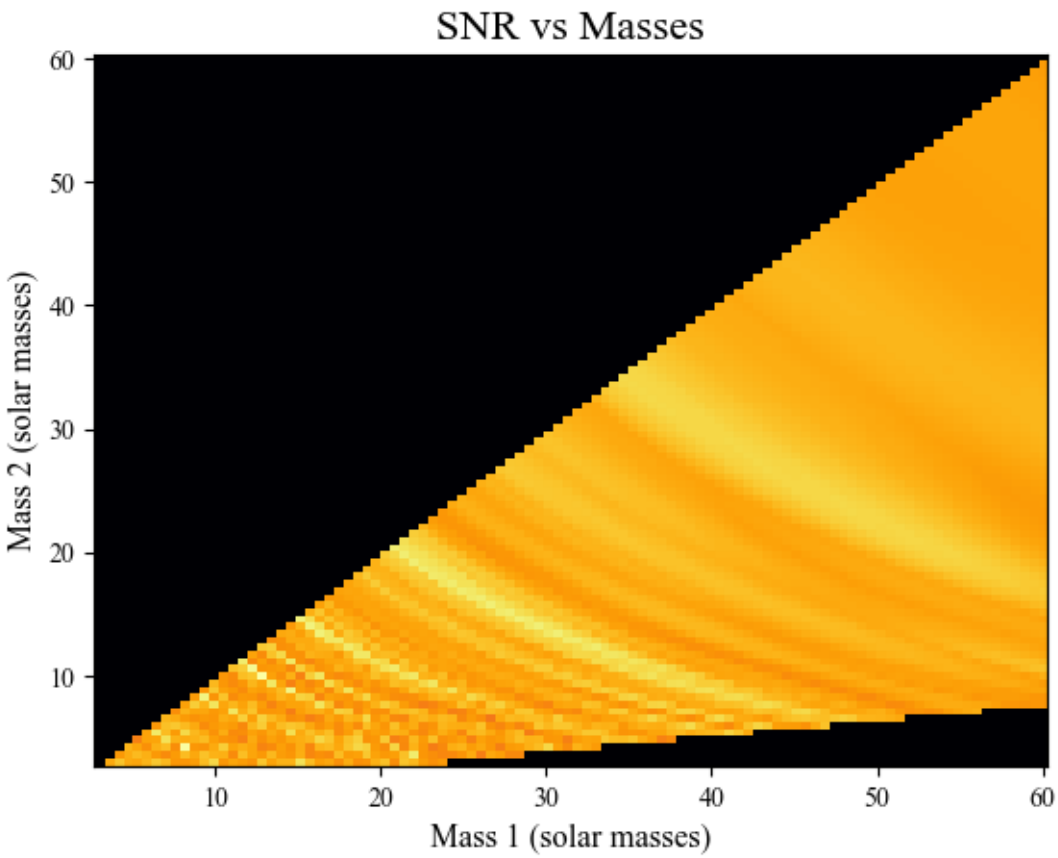
The Binary Black Hole merger event GW170814_strain.txt



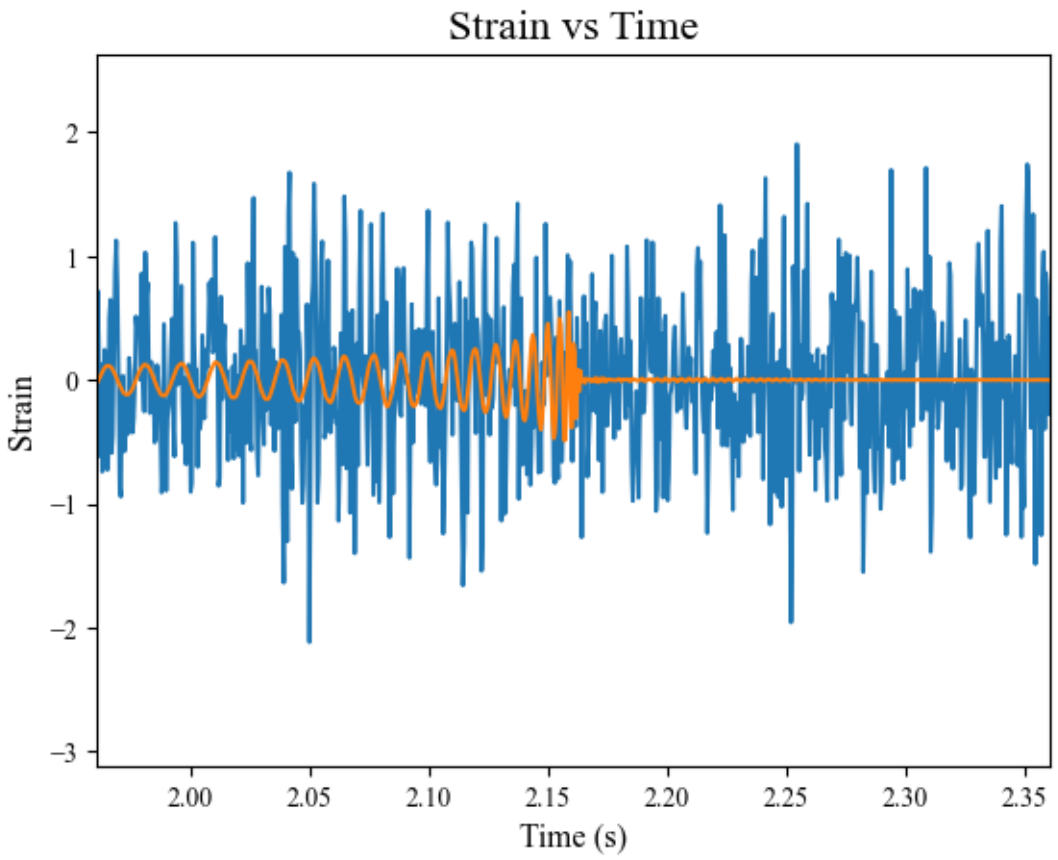

```
The best fit masses for GW170814_strain.txt are 30.2020202020202 and
29.848484848484848 respectively.
[[3.33544922]]
```




```
The best fit parameters for data set GW170814_strain.txt for distance, time, and
phase are 2268.3332639016076, 3.335124797759091, and 1.1587345020445857e-14
respectively.
------------------------------------------------------------
The Binary Black Hole merger event GW170817_strain.txt
------------------------------------------------------------
The Binary Black Hole merger event GW170818_strain.txt
```

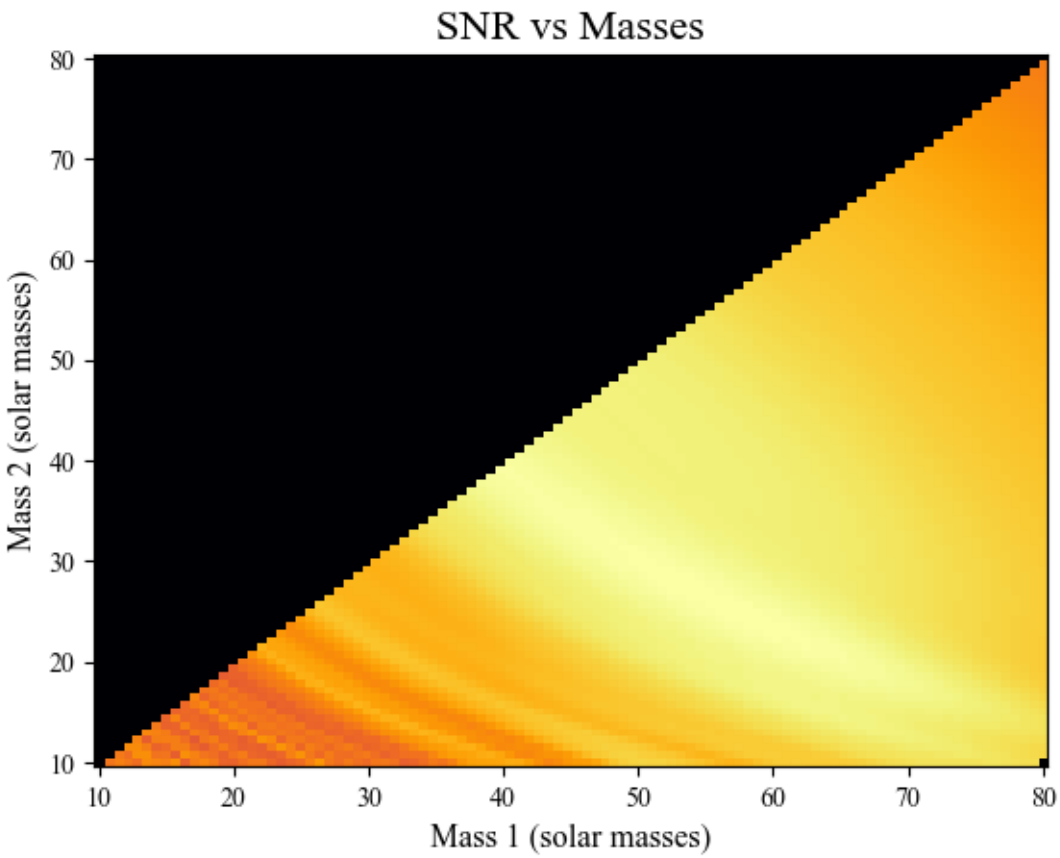



```
The best fit masses for GW170818_strain.txt are 12.787878787878782 and
9.909090909090907 respectively.
[[2.16064453]]
```

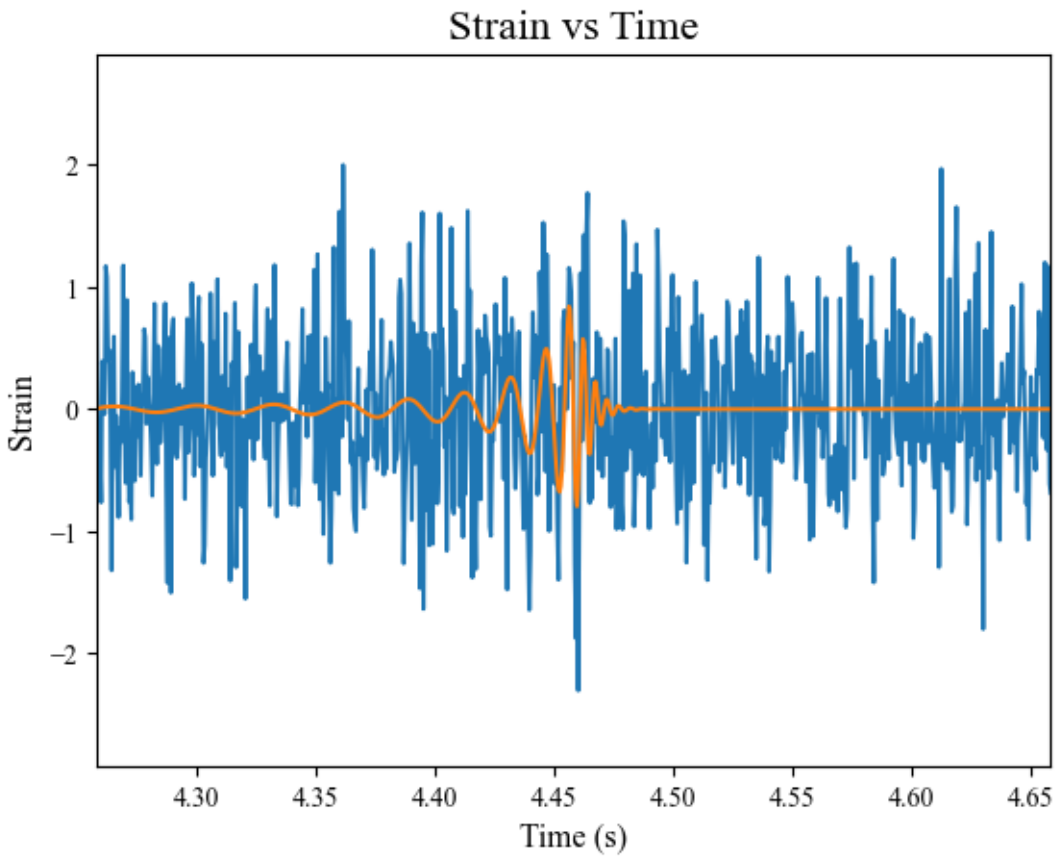


The best fit parameters for data set GW170818_strain.txt for distance, time, and phase are 1595.6538775454962, 2.160806907658626, and 0.7344919387503173 respectively.

The Binary Black Hole merger event GW170823_strain.txt




```
The best fit masses for GW170823_strain.txt are 55.25252525252525 and
26.262626262626263 respectively.
[[4.45800781]]
```



The best fit parameters for data set GW170823_strain.txt for distance, time, and phase are 4111.056369225488, 4.45602328415329, and 6.956297896491246e-11 respectively.

[231]:
```
df = pd.DataFrame({'file':names,'Distance (Mpc)':distance_coel, 'Time (s)':
 ↪time_coel, 'Phase (radians)':phase_coel, 'Larger Mass (solar masses)':
 ↪larger_masses, 'Smaller Mass (solar masses)':smaller_masses})
df.to_csv('coel_data.csv')
df
```

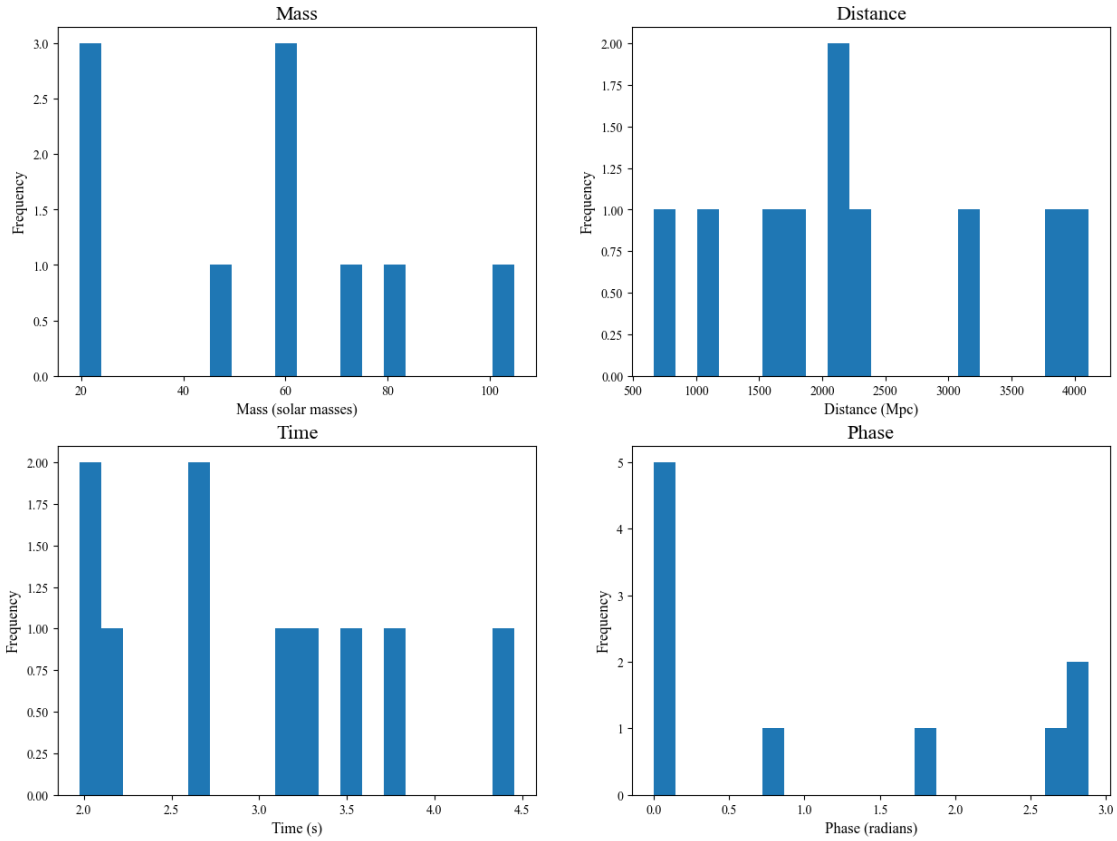
[231]:

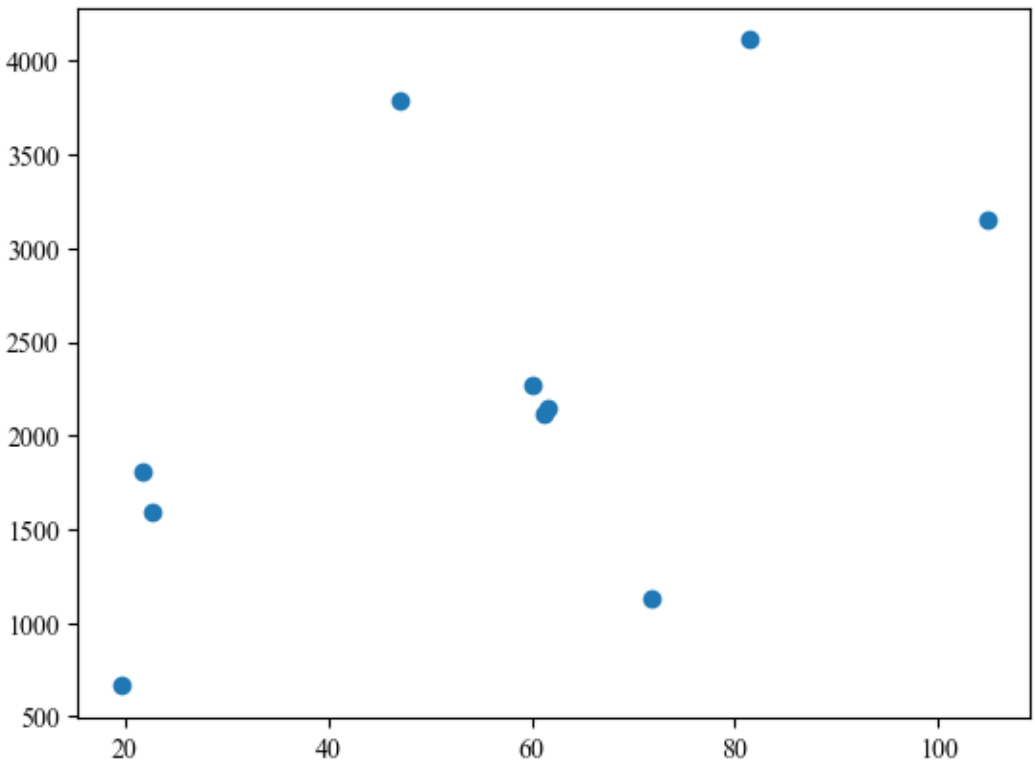
	file	Distance (Mpc)	Time (s)	Phase (radians)	\
0	GW150914_strain.txt	1127.861466	3.197771	3.028757e-02	
1	GW151012_strain.txt	3782.566300	2.703107	8.736723e-20	
2	GW151226_strain.txt	1811.601659	2.067185	3.114996e-12	
3	GW170104_strain.txt	2119.482510	1.974552	1.793321e+00	
4	GW170608_strain.txt	664.175028	3.793506	2.674341e+00	
5	GW170729_strain.txt	3150.396091	3.539463	2.885610e+00	
6	GW170809_strain.txt	2145.543376	2.640600	2.784260e+00	
7	GW170814_strain.txt	2268.333264	3.335125	1.158735e-14	
8	GW170818_strain.txt	1595.653878	2.160807	7.344919e-01	

```
9  GW170823_strain.txt     4111.056369  4.456023     6.956298e-11
```

	Larger Mass (solar masses)	Smaller Mass (solar masses)
0	36.060606	35.757576
1	35.404040	11.717172
2	12.101010	9.696970
3	35.858586	25.353535
4	11.272727	8.363636
5	75.151515	29.696970
6	46.414141	15.101010
7	30.202020	29.848485
8	12.787879	9.909091
9	55.252525	26.262626

[233]:
```
all_masses = np.array(larger_masses) + np.array(smaller_masses)
sort_all_masses = np.sort(all_masses)

plt.figure(figsize = (15,11))
plt.subplot(2,2,1)
plt.hist(all_masses, bins=20)
plt.xlabel('Mass (solar masses)', fontsize=axis_size)
plt.ylabel('Frequency', fontsize=axis_size)
plt.title('Mass', fontsize=title_size)

plt.subplot(2,2,3)
plt.hist(time_coel, bins=20)
plt.xlabel('Time (s)', fontsize=axis_size)
plt.ylabel('Frequency', fontsize=axis_size)
plt.title('Time', fontsize=title_size)

plt.subplot(2,2,2)
plt.hist(distance_coel, bins=20)
plt.xlabel('Distance (Mpc)', fontsize=axis_size)
plt.ylabel('Frequency', fontsize=axis_size)
plt.title('Distance', fontsize=title_size)

plt.subplot(2,2,4)
plt.hist(phase_coel, bins=20)
plt.xlabel('Phase (radians)', fontsize=axis_size)
plt.ylabel('Frequency', fontsize=axis_size)
plt.title('Phase', fontsize=title_size)
plt.show()

plt.figure()
plt.scatter(all_masses, distance_coel)
plt.show()
```


Mass
Frequency
Mass (solar masses)
Distance
Frequency
Distance (Mpc)
Time
Frequency
Time (s)
Phase
Frequency
Phase (radians)



[ ]:

[ ]:

[ ]:
```
# data = np.loadtxt('data_files/GW150914_strain.txt')
# time = data[:,0]
# strain = data[:,1]


# inv_psd = np.loadtxt('data_files/GW150914_inv_psd.txt',usecols=(1,))
# t, template = gw.make_template(37,36,2048,8,inv_psd,400)



# snr_ts = gw.get_snr(strain, template, 2048)
# min_time = time[np.where(np.max(snr_ts) == snr_ts)]-1
# max_time = time[np.where(np.max(snr_ts) == snr_ts)]+1



# masses = np.linspace(2,80,50)
```

```
# currentmax = 0
# highest_snr = [[0 for x in range(13)] for y in range(13)]
# mass1 = []
# mass2 = []
# allsnr = []

# for m1 in masses:
#     for m2 in masses:
#         if m1>m2 and (m1/m2) < 8:
#             t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
#             snr_ts = gw.get_snr(strain, template, 2048)
#             maxsnr = np.max(snr_ts)
#             allsnr.append(maxsnr)
#             if maxsnr > currentmax:
#                 currentmax = maxsnr
#                 M1 = m1
#                 M2 = m2
#         else:
#             allsnr.append(0)
# print(currentmax, M1, M2)

# reshaped = (np.reshape(allsnr, (50,50)).T)
# plt.figure()
# plt.pcolor(masses, masses, reshaped)
# plt.show()

# allsnr2 = []
# currentmax = 0
# masses2 = np.linspace(15,50,50)
# maxsnr = []
# for m1 in masses2:
#     for m2 in masses2:
#         if m1>m2 and (m1/m2) < 8:
#             t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
#             snr_ts = gw.get_snr(strain, template, 2048)
#             maxsnr = np.max(snr_ts)
#             allsnr2.append(maxsnr)
#             if maxsnr > currentmax:
#                 currentmax = maxsnr
#                 M1 = m1
#                 M2 = m2
#         else:
#             allsnr2.append(0)
# print(currentmax, M1, M2)
```

```
    # reshaped = (np.reshape(allsnr2, (50,50)).T)
    # plt.figure()
    # plt.pcolor(masses2, masses2, reshaped, cmap='inferno')
    # plt.show()


    # mass1 = M1
    # mass2 = M2
    # def make_signal(t,d,tc,phic):
    #     _,signal = gw.
↪make_template(mass1,mass2,2048,8,inv_psd,d=d,tc=tc,phic=phic)
    #     return signal
    # t = 8
    # distances = np.array(np.linspace(100,10000,10000))
    # tc_min = min_time
    # tc_max = max_time
    # phic_min = 0
    # phic_max = 2*np.pi
    # d_min = 100
    # d_max = 2000
    # #p0 = [890,(time[np.where(np.max(snr_ts) == snr_ts)]),5]


    # ht_err = np.array(np.ones(strain.size))
    # b = np.array([[d_min,tc_min,phic_min],[d_max,tc_max,phic_max]], dtype =␣
↪object)
    # popt, pcov = curve_fit(make_signal,time ,strain, p0 = [890,time[(np.
↪where(np.max(snr_ts) == snr_ts),)],5],bounds = b, sigma=ht_err,␣
↪absolute_sigma=True)

    # t, template = gw.
↪make_template(mass1,mass2,2048,8,inv_psd,popt[0],popt[1],popt[2])

    # print(f'The best fit parameters for distance, time, and phase are␣
↪{popt[0]}, {popt[1]}, and {popt[2]} respectively.')
    # plt.figure()
    # plt.plot(time, strain)
    # plt.plot(time, template)
    # plt.xlabel('Time (s)', fontsize=axis_size)
    # plt.ylabel('Strain', fontsize=axis_size)
    # plt.title('Strain vs Time', fontsize=title_size)
    # #plt.xlim(2.9,3.4)
    # plt.show()
```

3 Task 2 part a brute force method

[ ]:
```
masses = np.linspace(2,80,13)
highest_snr = []
mass1 = []
mass2 = []


for m1 in masses:
    for m2 in masses:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
            snr_ts = gw.get_snr(strain, template, 2048)
            highest_snr.append(np.max(snr_ts))
            mass1.append(m1)
            mass2.append(m2)

#print(len(highest_snr))
sorted_snr = np.sort(highest_snr)
n=20
rslt = sorted_snr[-n:]

# print(rslt)
for i in range(n):
    indexes = np.where(highest_snr == rslt[i])
    final_mass1
    print(mass1[indexes[0][0]],mass2[indexes[0][0]])
    # print(mass1[indexes[0][0]],mass2[indexes[0][0]])

masses = np.linspace(15,70,20)
highest_snr = []
mass1 = []
mass2 = []


for m1 in masses:
    for m2 in masses:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
            snr_ts = gw.get_snr(strain, template, 2048)
            highest_snr.append(np.max(snr_ts))
            mass1.append(m1)
            mass2.append(m2)

#print(len(highest_snr))
sorted_snr = np.sort(highest_snr)
n=20
```

```
rslt = sorted_snr[-n:]

# print(rslt)
for i in range(n):
    indexes = np.where(highest_snr == rslt[i])
    #print(mass1[indexes[0][0]],mass2[indexes[0][0]])
    # print(mass1[indexes[0][0]],mass2[indexes[0][0]])



masses = np.linspace(33,40,200)
highest_snr = []
mass1 = []
mass2 = []


for m1 in masses:
    for m2 in masses:
        if m1>m2 and (m1/m2) < 8:
            t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
            snr_ts = gw.get_snr(strain, template, 2048)
            highest_snr.append(np.max(snr_ts))
            mass1.append(m1)
            mass2.append(m2)

#print(len(highest_snr))
sorted_snr = np.sort(highest_snr)
n=20
rslt = sorted_snr[-n:]

# print(rslt)
for i in range(n):
    indexes = np.where(highest_snr == rslt[i])
    print(mass1[indexes[0][0]],mass2[indexes[0][0]])
    # print(mass1[indexes[0][0]],mass2[indexes[0][0]])
```

[ ]:
```
# for m1 in mass1:
#     for m2 in mass2:
#           if m1>m2 and (m1/m2) < 8:
#             t, template = gw.make_template(m1,m2,2048,8,inv_psd,400)
#             snr_ts = gw.get_snr(strain, template, 2048)
#             highest_snr.append(np.max(snr_ts))
#             mass1_2.append(m1)
#             mass2_2.append(m2)
```