Fishing for Exotic Compact Objects in the LIGO datset

Kushagra N. Nag*
Department of Physics and Astronomy, Texas Tech University, USA
(Dated: May 9, 2025)

With the profound advent of gravitational wave astronomy; general relativity, alternative theories of gravity, and nature of astrophysical compact objects are under great scrutiny. This project explores the possibility of searching for gravitational wave echoes in the LIGO dataset, which are predicted to provide signatures for the existence of exotic compact objects (ECOs) in our universe. In this work, we adapt an analytical ringdown + echo template describing gravitational wave signals from ECOs and perform Bayesian parameter inference on the GW150914 Hanford data. I first performed an injection study that successfully recovered the signal that justifies the construction of this method. Then, focusing on the GW150914 event, I found a signal-to-noise Bayes factor of around 2 and a signal-to-noise ratio of around 1.8 which provides no evidence towards an ECO signal in that event.

Keywords: Exotic Compact Objects, Gravitational Waves, Bayesian inference

I. INTRODUCTION

The most puzzling and debated predictions of Einstein's General Relativity (GR) are black hole event horizon and consequent information paradox [1]. They have puzzled scientists for more than a century, and the variety of approaches taken to explain them [2]. A clear signature from the final states of compact binary mergers one way or another would settle this matter.

Gravitational-wave (GW) astronomy provides a unique opportunity to test the nature of the compact object. So far, the field has evolved rapidly with the observation of a large number of compact binary merger events. Although it is difficult to probe the near-horizon nature in the inspiral phase of these merger events, the post-merger ringdown phase carries a definite signature of the compact remnant. While Einstein's GR predicts a spectrum of quasinormal modes characterized by the mass and angular momentum of the black hole [3, 4], it was predicted that due to the near-horizon geometry of Exotic Compact Objects (ECOs), the perturbed ECO will emit a series of delayed 'echoes' after the ringdown signal [5]. This is a distinct feature which can be probed if we observe a loud ringdown signal. While the current Advanced detectors do have enough ringdown signal-to-noise ratio (SNR), the third generation detectors such as the Cosmic Explorer [6] and Einstein Telescope [7] are designed to have sufficient ringdown SNR for compact objects specifically in the intermediate mass range [8].

The report is organized as follows. Section I A discusses the analytical template that I consider in this work, describing the ringdown + echo signals from ECOs along with simulations to better understand the signal construction. Section I B, discusses the method for obtaining the GW150914 event data. Section II will discuss the steps that I took to construct my waveform, likelihood, and priors with the intention of performing Bayesian inference. Finally, in section III I discuss the findings where in III A the method is verified with an injection study and in III B we ask whether we saw an ECO in GW150914?

A. ECO Template

Echoes in the post-merger ringdown GW signal are believed to provide evidence for the existence of ECOs. These ECOs are objects preventing the formation of a singularity and have a boundary/surface at a Planckian distance away from the would-be event horizon (EH) of a black hole (BH), $r_{ECO} = r_{BH}(1+\epsilon)$, i.e. they are horizonless. The location of the boundary of a given ECO (r_{ECO}) differs from that of a BH by a Planckian scale [9].

An interesting feature of these ECOs and BHs is that the GWs or high-frequency electromagnetic radiation can orbit them in circular motion [9]. According to GR, the location of this orbit is only possible at $r = \frac{3}{2} r_{BH}$ and is called the Photon Sphere (PS). The PS plays a major role in the space-time response to any type of wave and hence describes the behavior of high-frequency GWs near the horizon. Therefore, the presence of an additional horizonless boundary, with nonzero reflectivity, at r_{ECO} can modify the GW signal [5]. The higher frequency GWs manifesting from the ringdown of a compact object would transmit through the PS while reflecting the lower frequency waves toward the horizon. If a boundary exists, then the reflected low-frequency wave gets trapped between this wall and the PS. This trapped wave would then gradually leak out

* Correspondence email address: kunag@ttu.edu

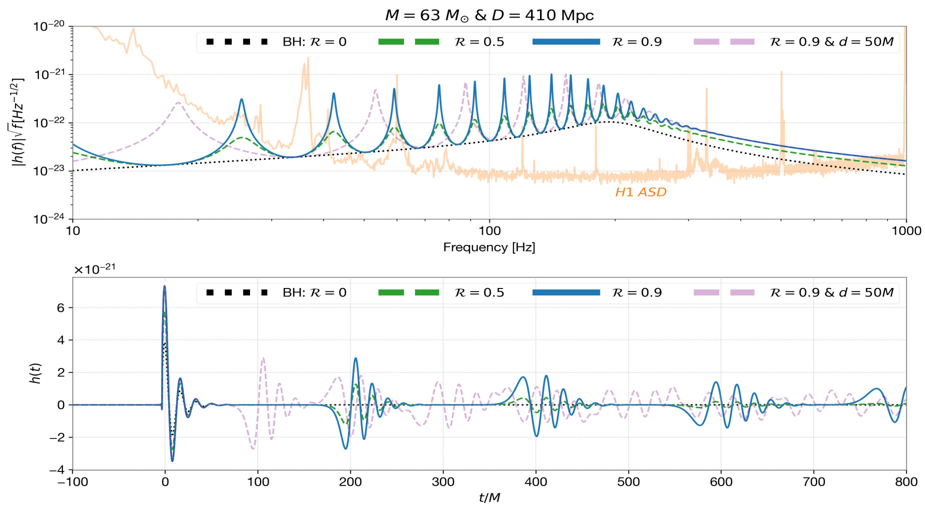

Figure 1. Simulation of the template 1 with different Echo parameters. *Top panel* : Frequency domain signal where higher peaks correspond to increasing $\mathcal{R}$. The orange curve corresponds to the H1 amplitude spectral density (see I B). *Bottom panel* : Time domain obtained by an inverse fourier transform of the top panel where we see that d characterizes the arrival time of subsequent echoes. See the discussion in I A.

during each iteration as repeating damped echoes [10].

Adriano Testa and Paolo Pani presented an analytical template, focusing on the physical properties and parameters, describing the ringdown + echo signal of nonspinning ECOs [11]. They expect the reflective boundary of the ECO to be in a position $x = x_0$ near the would-be EH of a BH that is enclosed by a light ring. The properties of this boundary are characterized by the complex and frequency-dependent reflection coefficient $\mathcal{R}$ while the compactness d characterizes the width of the cavity formed by the boundary and PS. In this picture of the ECO, the space-time configuration is modified in the sense that apart from the PS, the boundary at x_0 also acts as a scattering barrier to the GWs. This results in a modification of the condition of wave propagation, i.e. the in-going waves upon reflection gain an additional factor of reflectivity (due to the boundary) and then transmit through the PS to reach the outside observer. Their model for post-merger excitations is given by Eq. 1.

$$\tilde{Z}^+(\omega) = \sqrt{\frac{\pi}{2}}\mathcal{A}\frac{e^{i(\omega-\omega_I)t_0}(1+\mathcal{R})\Gamma\left(1-\frac{i\omega}{\alpha}\right)\left(\omega_R\sin(\omega_R t_0+\phi)+i(\omega+i\omega_I)\cos(\omega_R t_0+\phi)\right)}{[(\omega+i\omega_I)^2-\omega_R^2]\left[\pi\Gamma\left(1-\frac{i\omega}{\alpha}\right)+e^{2id\omega}\mathcal{R}\cosh\left(\frac{\pi\omega_R}{\alpha}\right)\Gamma\left(\frac{1}{2}-i\frac{\omega+\omega_R}{\alpha}\right)\Gamma\left(\frac{1}{2}-i\frac{\omega-\omega_R}{\alpha}\right)\Gamma\left(1+\frac{i\omega}{\alpha}\right)\right]} \tag{1}$$

which is characterized by the parameters p given in table I. They arrive at this conclusion by finding solutions to how a source would respond to this modified space-time geometry where they assume the PS and the reflective boundary as potentials which behave as barriers to incoming and outgoing GWs. A detailed discussion and analysis of their model is beyond the scope of this work, and here we just focus on adopting their template with a motive to fish for ECOs using Bayesian inference.

In eq. 1, ω_R denotes the real part of the QNM frequency, ω_I corresponds to the imaginary part of the complex QNM frequency (the inverse of which denotes the damping time) and α corresponds to the second derivative of the PS potential. These parameters are a function of the source mass M and were fixed to their corresponding fundamental QNM terms (see Table I).

With a motive to understand the dependence of the template 1 on the Echo parameters defined in I, Fig 1 shows the signal simulation for a source with $63M_\odot$ at a luminosity distance $D = 410$ Mpc. This is similar to the case of the final black hole system observed in the GW150914 event. The top panel

Table I. Ringdown+echo parameters p characterizing the analytical template 1 adopted from [11].

Ringdown	M	total mass of the object
	$\mathcal{A}$	ringdown amplitude $\sim M/D$
	ϕ	ringdown phase (fixed in our analysis)
	t_0	start time of the ringdown (fixed in our analysis)
	ω_R	$0.3737/M$
	ω_I	$-0.08896/M$
	α	$0.2161/M$
Echo	d	width of cavity
	$\mathcal{R}(\omega)$	reflection coefficient at the surface

shows the construction of the frequency domain signal in comparison to the amplitude spectral density of the Hanford (H1) interferometer (discussed in I B). We can see that for $\mathcal{R} = 0$ (BH case) the template reduces to the usual BH ringdown case. With increasing reflectivity ($\mathcal{R}$), we observe sharper peaks at lower frequencies, which accounts for the behavior of low-frequency GWs trapped in the cavity with peaks formed due to excitations of QNMs.

In addition, the time domain waveform was obtained by performing an inverse Fourier transform defined by Eq. 2,

$$h(t) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \tilde{h}(\omega) e^{-i\omega t} d\omega \qquad (2)$$

The effect of the cavity in the time domain can be seen in the bottom panel of Fig 1 where we see that after the merger at $t/M = 0$, "extra" signals in the form of echoes are visible. The effect of d is much clearer here which governs the arrival times of subsequent echoes. Also, an increase in $\mathcal{R}$, increases the amplitude visible both in top and bottom panel, suggesting the feasibility of detection with higher $\mathcal{R}$. The simulation code is given in Appendix A.

B. GW150914 data

In this work, I focus on the GW150914 which was the first observed binary black hole merger event. The motivation to choose it was because of the presence of a small chunk of the ringdown signal observed in the event [12]. I use the $fetch\_open\_data()$ function of the $TimeSeries$ object in the $GWpy$ package [13] to access it. The focus is on the H1 interferometer data. I accessed 2 seconds of data centered around the event with a sampling rate of 4096 Hz. As is done in many gravitational wave data analysis, I assume the detector noise to be stationary Gaussian noise. To estimate the noise, I accessed 64 seconds of data before the merger event and allowed it to pass through the $.psd()$ function of the time-series data, which outputs the power spectral density (PSD) estimate of the H1 data by applying a Tukey window and following the welch method. This splits the data into overlapping segments, window is applied to each segment to reduce spectral leakage, fast Fourier transform (fft) is computed on each segment and finally the squared magnitudes of ffts are averaged over each windowed segment to obtain the PSD estimate. The code corresponding to this is given in Appendix D.

II. METHOD

I use the powerful yet elegant package *bilby* [14] to perform Bayesian parameter estimation using the template defined in I A and the GW150914 event data and the noise estimate defined in I B.

I code the template defined by eq. 1 which I pass to the $WaveformGenerator$ function of the $waveform\_generator$ object in *bilby* as a $frequency\_domain\_source\_model$ with the same *duration* as that of the data. This allows us to store any given template as a $waveform$ object suitable for sampling in *bilby* [15]. In addition, for the suitability of the *bilby* sampling, the data and the corresponding noise estimates are stored in the $interferometer$ object of the *bilby* library [14].

Then I define a $My\_Likelihood$ class, passing the *bilby.Likelihood* environment suitable for *bibly* sampling. This class contains 2 functions, $log\_likelihood()$ and $noise\_log\_likelihood()$. The former is defined by eq. 3 which is the noise-weighted inner product of the strain data $d(f)$ minus the waveform/template $h(f;\theta)$ and $S_n(f)$ is the power spectral density.

$$lnL =< d - h|d - h >= -2 \sum_f \frac{|d(f) - h(f;\theta)|^2}{S_n(f)} \Delta f \qquad (3)$$

The latter is defined by eq. 4 which is the noise-weighted inner product of the data with itself (assuming that there is no signal). Defining the $noise\_log\_likelihood()$ function is very useful, as *bilby* automatically computes the signal-to-noise Bayes factor ($BF_{S/N}$) using the respective evidence calculation during sampling [15].

$$lnL_{noise} =< d|d >= -2 \sum_f \frac{|d(f)|^2}{S_n(f)} \Delta f \qquad (4)$$

Finally, I pass the interferometer object (containing data and noise) and the waveform object (containing the signal) to *My_likelihood* which are used in the calculations of 3 and 4. And I set uniform priors on the template parameters, $\mathcal{R} \in [0, 1]$, $d \in [0, 70]$ M, $M \in [10, 80]$ $M_\odot$ and $D \in [0.1, 1.0]$ Gpc. The code that reproduces all the remaining results is given in Appendix B and Appendix C.

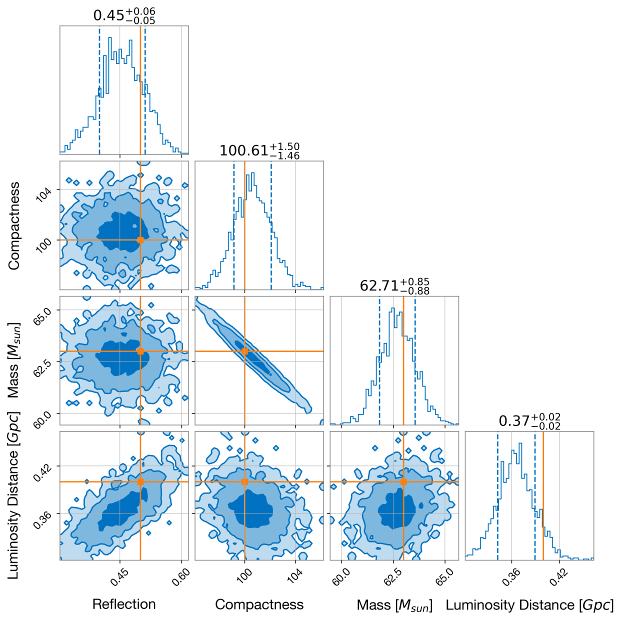

Figure 2. The corner plot showing the $1-d$ and $2-d$ posterior distributions of the template parameters. Here we can see that the injected values for all the parameters were successfully recovered. See the discussion in III A.

III. RESULTS AND CONCLUSIONS

A. Injection Study

Before jumping on the data, I did an injection analysis in which I inject the signal into the data and ask if my code structure and implementation could recover it? *bilby* uses *dynesty* as its default sampler [15]. The stopping condition of the sampling is based on the change in the evidence and I specify that to a condition of $dlogz = 0.01$. Fig. 2 shows the corner plot of our parameters where we can see that all parameters are recovered to their injected value ($\mathcal{R} = 0.5, d = 100, M = 63$ and $D = 0.4$). Note that I set a different uniform prior for d here with $\in [0, 200]$. In addition, the $2-d$ posterior contours of $d - M$ and $D - \mathcal{R}$ interestingly show a negative and positive correlation between the parameters respectively. The obtained $BF_{S/N} \approx 282$, provides very strong evidence (according to the Jeffrey scale) towards recovery.

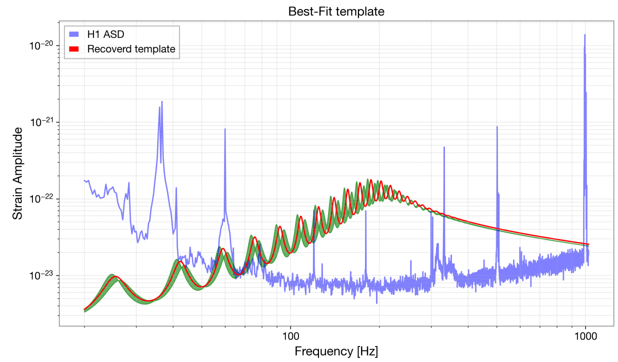

Figure 3. The recovered signal (red) with its 1σ uncertainty (green) as a comparison to the H1 ASD. We can see that the recovered signal is well above the H1 sensitivity curve. See the discussion in III A.

$$\rho = 2\sqrt{\sum_f \frac{|d(f) - h(f;\hat{\theta})|^2}{S_n(f)} \Delta f} \tag{5}$$

Fig 3 is where I plot the recovered waveform with uncertainty of 1σ as a comparison to $\sqrt{PSD}$ of the H1 interferometer showing the loudness of the signal. To access loudness, I further compute the Signal-to-Noise ratio (SNR) defined as the noise-weighted inner product of the data with the best-fit recovered template given by 5 . For the injection study I obtained an SNR of ≈ 21 which further quantifies the result.

B. Is GW150914 an ECO event?

With the successful injection study, I focus on the GW150914 event and perform Bayesian parameter estimation using 2 seconds of data around the event and the same priors defined in section II.

Fig 4 shows the corner plot showing the posterior distributions of the parameters. Surprisingly, I see a constraint on the "echo" parameters, namely $\mathcal{R}$ and d. The negative correlation of M and d is still visible in their $2-d$ posterior contours. Also, we can see that the inferred estimate of mass M and luminosity distance D is very off than what was observed in the GW150914 event, which can be explained (as observed in III A) by the correlations of these parameters with compactness d and reflection coefficient $\mathcal{R}$ respectively.

Fig 5 shows the signal construction, with best-fit template parameters obtained by Bayesian infer-

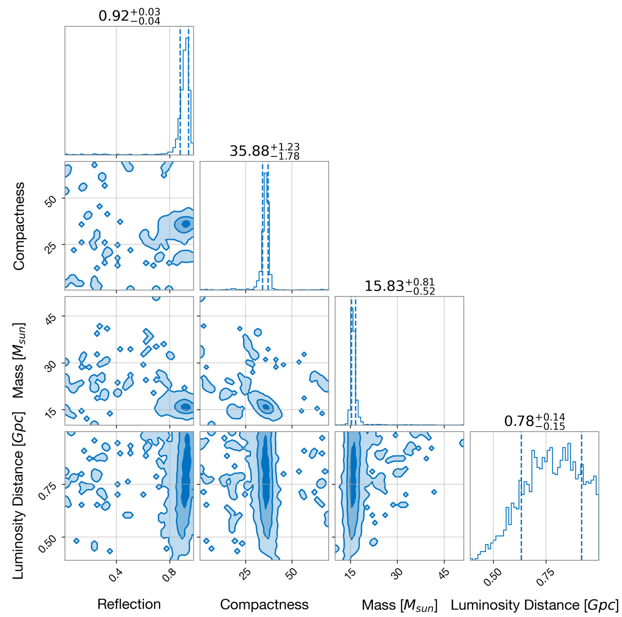

Figure 4. The corner plot showing $1-d$ and $2-d$ posterior distribution of the template parameters. We see that the inferred values of M and D does not agree with what was observed for GW150914 because of the correlation of these parameters with the Echo parameters. See discussions in III A and III B.

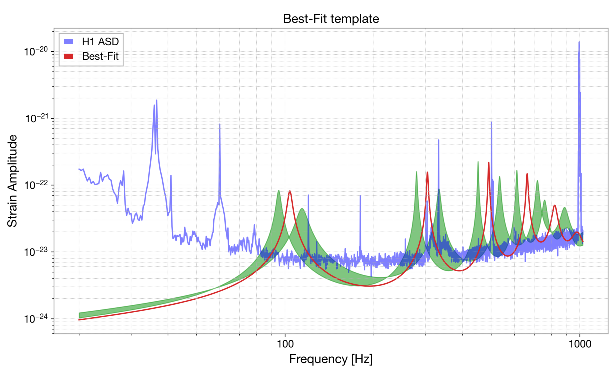

Figure 5. The best-fit signal (red), obtained by using the median values of the inferred parameters, with its 1σ uncertainty (green) as a comparison t the H1 ASD. The observed $BF_{S/N} \approx 2$ which is not substantial enough (according to Jeffry scale) and the observed SNR is around 1.8 (see section III B).

ence, as a comparison to the amplitude spectrum of the H1 interferometer. We can see that the peaks in the signal or it's uncertainty is very close to the H1 noise peaks, suggesting that it is fitting more for noise. However, to quantify the search, it is important to look at the $BF_{S/N}$ and for this I obtained that to be ≈ 2, which is not substantial enough (according to Jeffry scale) to say anything about this ECO model. Also, the SNR is around 1.8, which further quantifies to the fact that we see NO EVIDENCE towards GW150914 being an ECO.

IV. DISCUSSION

In this work I performed a Bayesian data analysis of the GW150914 data to fish for ECOs. The approach was first tested in the form of an injection study in which it successfully recovered the signal with $BF_{S/N} = 282$. When focusing on GW150914, the analysis resulted in a $BF_{S/N} = 2$ which according to the Jeffrey scale is not substantial enough to conclude evidence towards an ECO model. In the injection study, an interesting feature regarding the correlations between the ringdown and echo parameters was observed. This was further reflected in the GW150914 analysis, where the posterior distributions of the ringdown parameters were different than what was observed.

However, it would be interesting to search for these ECOs in the events where the ringdown were significant/loud. Also, it would be interesting to fish for these ECOs using a non-phenomenological template in which the model depends on the appearances of the echoes rather than focusing on the properties of the objects itself. There are different techniques of testing GR with methods including insprial-merger-ringdown consistency checks, BH spectroscopy, and more [16]. This involves a need for loud information in the ringdown portion of GWs and such analysis would play a crucial role in understanding the formation process of compact objects, fundamental physics, and cosmology. With the profound development of the next generation interferometers, it would be possible to capture a loud ringdown of GW events leading to a better constrain and possibility towards fishing for these ECOs.

ACKNOWLEDGEMENTS

I would like to thank Dr. Nihan Pol for educating us about the many data analysis techniques involved in Astronomy/Astrophysics. Specifically on Bayesian parameter inference which was the core technique used in this work.

[1] S. W. Hawking, Phys. Rev. D **14**, 2460 (1976).

[2] W. G. Unruh and R. M. Wald, Reports on Progress in Physics **80**, 092002 (2017), arXiv:1703.02140

[hep-th].
[3] H.-P. Nollert, Classical and Quantum Gravity **16**, R159 (1999).
[4] E. Berti, V. Cardoso, and A. O. Starinets, Classical and Quantum Gravity **26**, 163001 (2009), arXiv:0905.2975 [gr-qc].
[5] V. Cardoso, E. Franzin, and P. Pani, Phys. Rev. Lett. **116**, 171101 (2016), arXiv:1602.07309 [gr-qc].
[6] B P Abbott et al., Classical and Quantum Gravity **34**, 044001 (2017), arXiv:1607.08697 [gr-qc].
[7] S. Hild et al., Classical and Quantum Gravity **28**, 094013 (2011), arXiv:1012.0908 [gr-qc].
[8] V. e. a. Kalogera, (2021), 10.48550/ARXIV.2111.06990, arXiv:2111.06990 [gr-qc].
[9] V. Cardoso and P. Pani, Nature Astronomy **1**, 586 (2017), arXiv:1709.01525 [gr-qc].
[10] V. Cardoso and P. Pani, Living Reviews in Relativity **22** (2019), 10.1007/s41114-019-0020-4, arXiv:1904.05363 [hep-ph].
[11] A. Testa and P. Pani, Physical Review D **98** (2018), 10.1103/physrevd.98.044018, arXiv:1806.04253 [gr-qc].
[12] R. Cotesta, G. Carullo, E. Berti, and V. Cardoso, Phys. Rev. Lett. **129**, 111102 (2022).
[13] D. M. Macleod, J. S. Areeda, S. B. Coughlin, T. J. Massinger, and A. L. Urban, SoftwareX **13**, 100657 (2021).
[14] G. Ashton *et al.*, Astrophys. J. Suppl. **241**, 27 (2019), arXiv:1811.02042 [astro-ph.IM].
[15] G. Ashton and C. Talbot, Mon. Not. Roy. Astron. Soc. **507**, 2037 (2021), arXiv:2106.08730 [gr-qc].
[16] B. P. e. a. Abbott (LIGO Scientific and Virgo Collaborations), Phys. Rev. Lett. **116**, 221101 (2016).

Appendix A: Simulating the ECO model

```python

import numpy as np
import matplotlib.pyplot as plt
from astropy import constants as const
import scipy.special as special
import tqdm as tqdm


#### Our model
pc = (const.pc).value
Gpc = pow(10,9)*pc
M_sun = (const.M_sun).value
G = (const.G).value
c = (const.c).value
#### Defining the model
def echo(frequency_array, R, d, M, D, wr, wi, alpha, t_0, phi):


    w = 2 * np.pi * frequency_array


    Zp = np.sqrt(np.pi/2) * (M/D) * (M_sun/Gpc) * np.float64(G/(c**2)) * (np.exp(1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3)))))*t_0)*(1+R)*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3)))))))*(((wr/(M*(M_sun*(G/(c**3)))))*np.sin(((wr/(M*(M_sun*(G/(c**3)))))*t_0)+phi)+((1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))*np.cos((wr/(M*(M_sun*(G/(c**3))))*t_0)+phi))))/
                                        ((((w-(1j*wi)/(M*(M_sun*(G/(c**3)))))*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))-((wr/(M*(M_sun*(G/(c**3)))))**2))*((np.pi*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))+(np.exp(2j*d*(M*(M_sun*(G/(c**3))))*w)*R*np.cosh((np.pi*wr)/alpha)*special.gamma((1/2)-(1j*(w+(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3))))))*special.gamma((1/2)-(1j*(w-(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3)))))))*special.gamma(1+((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))))

    return Zp


#### Defining a frequency range and other required variables to store the signal
freq_range = np.linspace(1, 10000, 100000)
bh_signal = np.zeros(freq_range.size, dtype='complex')
signal_rp5 = np.zeros(freq_range.size, dtype='complex')
signal_rp9 = np.zeros(freq_range.size, dtype='complex')
signal_rp9_d50 = np.zeros(freq_range.size, dtype='complex')

#### Simulating...
for i in tqdm.tqdm(range(len(freq_range))):
    bh_signal[i] = echo(frequency_array=freq_range[i], R=0.0, d=0, M=63, D=0.4, wr=0.3737, wi=-0.08896, alpha=0.2161, t_0=-0.001, phi=0)
    signal_rp5[i] = echo(frequency_array=freq_range[i], R=0.5, d=100, M=63, D=0.4, wr=0.3737, wi=-0.08896, alpha=0.2161, t_0=-0.001, phi=0) #np.zeros(freq_range.size)
    signal_rp9[i] = echo(frequency_array=freq_range[i], R=0.9, d=100, M=63, D=0.4, wr=0.3737, wi=-0.08896, alpha=0.2161, t_0=-0.001, phi=0)
    signal_rp9_d50[i] = echo(frequency_array=freq_range[i], R=0.9, d=50, M=63, D=0.4, wr=0.3737, wi=-0.08896, alpha=0.2161, t_0=-0.001, phi=0)


#### Defining the time array and required variables to store the signal
t_range = np.linspace(-20000, 20000, 100000)*(63.0*(M_sun*(G/(c**3))))
w_range = 2*np.pi*freq_range
dw = (w_range[len(w_range)-1]-w_range[0])/len(w_range)

signal_time_bh = np.zeros(t_range.size)
signal_time_rp5 = np.zeros(t_range.size)
signal_time_rp9 = np.zeros(t_range.size)
```

```
signal_time_rp9_d50 = np.zeros(t_range.size)

for i in tqdm.tqdm(range(len(t_range)), desc='Cal. time domain', leave=False):

    signal_time_bh[i] = (1/np.sqrt(2*np.pi))*(bh_signal*np.exp(-1j*w_range*t_range[i])*dw).sum()
    signal_time_rp5[i] = (1/np.sqrt(2*np.pi))*(signal_rp5*np.exp(-1j*w_range*t_range[i])*dw).sum()
    signal_time_rp9[i] = (1/np.sqrt(2*np.pi))*(signal_rp9*np.exp(-1j*w_range*t_range[i])*dw).sum()
    signal_time_rp9_d50[i] = (1/np.sqrt(2*np.pi))*(signal_rp9_d50*np.exp(-1j*w_range*t_range[i])*dw).sum()



#### Plotting the simulations
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(13, 8))

plt.subplots_adjust(hspace=0.3)

ax1.set_title('$M = 63$ $M_{\\odot}$ $&$ $D = 410$ Mpc')

ax1.plot(H1.frequency_array, H1.amplitude_spectral_density_array, ls='-', c='tab:orange', alpha=0.3)
ax1.plot(freq_range, np.sqrt(freq_range)*np.abs(bh_signal), ls=':', c='k', label='BH: $\\mathcal{R} = 0$')
ax1.plot(freq_range, np.sqrt(freq_range)*np.abs(signal_rp5), ls='--', c='tab:green', label='$\\mathcal{R} = 0.5$')
ax1.plot(freq_range, np.sqrt(freq_range)*np.abs(signal_rp9), ls='-', c='tab:blue', label='$\\mathcal{R} = 0.9$')
ax1.plot(freq_range, np.sqrt(freq_range)*np.abs(signal_rp9_d50), c='purple', ls='--', alpha=0.3, label='$\\mathcal{R} = 0.9$ $&$ $d=50M$')

ax1.set_xlabel('Frequency [Hz]', fontsize=12)
ax1.set_ylabel('$|h(f)| \\sqrt{f} [Hz^{-1/2}]$', fontsize=12)
ax1.set_xscale('log')
ax1.set_yscale('log')
ax1.legend(framealpha=0.1, handlelength=5, ncols=5, fancybox=True)#handles=custom_lines_ax1, frameon=False)
ax1.set_xlim(10, 1000)
ax1.set_ylim(1e-24, 1e-20)
ax1.text(200, 3e-24, '$H1$ $ASD$', c='tab:orange')
ax1.grid(ls=':')



ax2.plot(t_range/(63.0*(M_sun*(G/(c**3)))), signal_time_bh, ls=':', c='k', label='BH: $\\mathcal{R}=0$')
ax2.plot(t_range/(63.0*(M_sun*(G/(c**3)))), signal_time_rp5, ls='--', c='tab:green', label='$\\mathcal{R}=0.5$')
ax2.plot(t_range/(63.0*(M_sun*(G/(c**3)))), signal_time_rp9, c='tab:blue', label='$\\mathcal{R}=0.9$')
ax2.plot(t_range/(63.0*(M_sun*(G/(c**3)))), signal_time_rp9_d50, ls='--', c='purple', label='$\\mathcal{R}=0.9$ $&$ $d=50M$', alpha=0.3)
ax2.set_xlim(-0.1e3, 0.8e3)
ax2.set_xlabel('$t/M$', fontsize=12)
ax2.set_ylabel('$h(t)$', fontsize=12)
ax2.legend(framealpha=0.1, handlelength=5, ncols=4, fancybox=True)
ax2.grid(ls=':')

#plt.savefig('./Simulation_63M_410Mpc.png', dpi=300)

plt.show()
```

Listing 1. This code will perform simulations of the model considered in this work.

Appendix B: Injection recovery

```
import numpy as np
import matplotlib.pyplot as plt
from astropy import constants as const
import scipy.special as special
import bilby
import H1 as get_interf

#### Define time of event in GPS
time_of_event=1126259462.4
#### Get the data and PSD
H1 = get_interf.get_H1(time_of_event=1126259462.4, post_trigger_duration=2, duration=3, psd_duration_multi=32)


#### Our model
pc = (const.pc).value
Gpc = pow(10,9)*pc
M_sun = (const.M_sun).value
G = (const.G).value
c = (const.c).value
#### Defining the model
def echo(frequency_array, R, d, M, D, wr, wi, alpha, t_0, phi):


    w = 2 * np.pi * frequency_array


    Zp = np.sqrt(np.pi/2) * (M/D) * (M_sun/Gpc) * np.float64(G/(c**2)) * (np.exp(1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3)))))*t_0)*(1+R)*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3)))))))*(((wr/(M*(M_sun*(G/(c**3)))))*np.sin(((wr/(M*(M_sun*(G/(c**3)))))*t_0)+phi)+((1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))*np.cos((wr/(M*(M_sun*(G/(c**3))))*t_0)+phi))))/
                                            ((((w-(1j*
```

```python
    wi)/(M*(M_sun*(G/(c**3)))))*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))-((wr/(M*(M_sun*(G/(c**3)))))**2))*((np.pi*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))+(np.exp(2j*d*(M*(M_sun*(G/(c**3))))*w)*R*np.cosh((np.pi*wr)/alpha)*special.gamma((1/2)-(1j*(w+(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3)))))))*special.gamma((1/2)-(1j*(w-(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3))))))))*special.gamma(1+((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))))

    cross = np.zeros(len(frequency_array))
    return {"plus": Zp, "cross": cross}


#### Define the Likelihood according to what bilby likes
class My_Likelihood(bilby.Likelihood):

    def __init__(self, interferometers, waveform_generator, priors=None):

        super(My_Likelihood, self).__init__(dict())
        self.interferometers = interferometers[0]
        self.waveform_generator = waveform_generator
        self.priors = priors

    def priors(self):
        return self.priors

    def log_likelihood(self):

        waveform = self.waveform_generator.frequency_domain_strain(self.parameters)
        residual = self.interferometers.frequency_domain_strain - \
                    self.interferometers.get_detector_response(waveform, self.parameters)
        psd = self.interferometers.power_spectral_density_array
        duration = self.waveform_generator.duration

        log_l = -2.0 / duration * np.sum((np.conj(residual)*residual) / psd)

        return log_l.real

    def noise_log_likelihood(self):

        noise = self.interferometers.frequency_domain_strain
        psd = self.interferometers.power_spectral_density_array
        duration = self.waveform_generator.duration

        log_l = -2.0 / duration * np.sum(np.abs(noise)**2 / psd)

        return log_l.real


#### Define the sampling frequency and the data duration
sampling_frequency = H1.sampling_frequency
duration = H1.duration


#### Call the waveform_generator to create our waveform model.
waveform = bilby.gw.waveform_generator.WaveformGenerator(
    duration=duration,
    sampling_frequency=sampling_frequency,
    frequency_domain_source_model=echo,
)


#### define parameters to inject.
injection_parameters = dict(
    R=0.5,
    d=100,
    M=63,
    D=0.4,
    wr=0.3737,
    wi=-0.08896,
    alpha=0.2161,
    t_0=-0.001,
    phi=0.0,
    ra=2.19432,
    dec=-1.2232,
    psi=0.532268,
    geocent_time=1126259462.4
)


#### Inject the signal
H1.inject_signal(
    waveform_generator=waveform, parameters=injection_parameters, raise_error=False
)


#### Define the prior for our parameters
prior = injection_parameters.copy()
prior['R'] = bilby.core.prior.Uniform(name='Reflection', minimum=0.3,maximum=0.8)
prior['d'] = bilby.core.prior.Uniform(name='Compactness', minimum=80, maximum=120)
prior['M'] = bilby.core.prior.Uniform(name="Mass", minimum=40, maximum=80, unit="$M_{sun}$")
prior['D'] = bilby.core.prior.Uniform(name="Luminosity Distance", minimum=0.2, maximum=0.6, unit="$Gpc$")


#### Instantiate the Likelihood
likelihood = My_Likelihood(interferometers=[H1], waveform_generator=waveform, priors=prior)


#### launch sampler
result = bilby.core.sampler.run_sampler(
    likelihood,
    prior,
    sampler="dynesty",
    npoints=500,
```

```
    walks=5,
    nact=3,
    injection_parameters=injection_parameters,
    outdir="Inject_recover",
    label="Echo_recover",
    dlogz=0.01
)

#### This will automatically show the signal-to-noise Bayes factor
#### Plot the corner plot
result.plot_corner()


#### Plot the recovered signal
idxs = likelihood.interferometers[0].strain_data.frequency_mask  # This is a boolean mask of the frequencies which we'll use in the analysis
plt.figure(figsize=(10, 6))
plt.loglog(likelihood.interferometers[0].frequency_array[idxs],
        likelihood.interferometers[0].amplitude_spectral_density_array[idxs], label='H1 ASD', alpha=0.5, color='blue')
plt.loglog(waveform.frequency_array[idxs], np.sqrt(waveform.frequency_array[idxs])*
         np.abs(waveform.frequency_domain_strain()['plus'][idxs]), label='Recoverd template', color='red')
plt.fill_between(waveform.frequency_array[idxs], np.sqrt(waveform.frequency_array[idxs])*
                 np.abs(waveform.frequency_domain_source_model(waveform.frequency_array, 0.45-0.05, 100.61-1.46, 62.71-0.88,
                              0.37-0.02, 0.3737, -0.08896, 0.2161, 0, 0)['plus'])[idxs],
                 np.sqrt(waveform.frequency_array[idxs])*
                 np.abs(waveform.frequency_domain_source_model(waveform.frequency_array, 0.45+0.06, 100.61+1.50, 62.71+0.85,
                              0.37+0.02, 0.3737, -0.08896, 0.2161, 0, 0)['plus'])[idxs],
                 alpha=0.6, color='green')

plt.xlabel('Frequency [Hz]')
plt.legend(framealpha=0.6)
plt.ylabel("Strain Amplitude")
plt.title("Best-Fit template")
plt.grid(True, which="both", ls=":")
plt.tight_layout()
#plt.savefig("./Freq_domain_recover_vs_psd.png", dpi=300)
plt.show()


#### Calculate the SNR
sig = (1/(2*np.pi))*np.sqrt(waveform.frequency_array[idxs])*np.abs(waveform.frequency_domain_strain()['plus'][idxs])
snr = np.sqrt(4/H1.duration * np.sum((likelihood.interferometers[0].frequency_domain_strain[idxs]*np.conj(sig))/
                                      likelihood.interferometers[0].power_spectral_density_array[idxs]).real)

print("The SNR:", snr)
```

Listing 2. Injection Study python snippet

Appendix C: GW150914 study

```

import numpy as np
import matplotlib.pyplot as plt
from astropy import constants as const
import scipy.special as special
import bilby
import H1 as get_interf


#### Define time of event in GPS
time_of_event=1126259462.4
#### Get the data and PSD
H1 = get_interf.get_H1(time_of_event=1126259462.4, post_trigger_duration=2, duration=3, psd_duration_multi=32)


#### Our model
pc = (const.pc).value
Gpc = pow(10,9)*pc
M_sun = (const.M_sun).value
G = (const.G).value
c = (const.c).value
#### Defining the model
def echo(frequency_array, R, d, M, D, wr, wi, alpha, t_0, phi):


    w = 2 * np.pi * frequency_array


    Zp = np.sqrt(np.pi/2) * (M/D) * (M_sun/Gpc) * np.float64(G/(c**2)) * (np.exp(1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3)))))*t_0)*(1+R)*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3)))))))*(((wr/(M*(M_sun*(G/(c**3)))))*np.sin(((wr/(M*(M_sun*(G/(c**3)))))*t_0)+phi)+((1j*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))*np.cos((wr/(M*(M_sun*(G/(c**3)))))*t_0)+phi))))/
                                                  ((((w-(1j*wi)/(M*(M_sun*(G/(c**3)))))*(w-(1j*wi)/(M*(M_sun*(G/(c**3))))))-((wr/(M*(M_sun*(G/(c**3)))))**2))*((np.pi*special.gamma(1-((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))+(np.exp(2j*d*(M*(M_sun*(G/(c**3))))*w)*R*np.cosh((np.pi*wr)/alpha)*special.gamma((1/2)-(1j*(w+(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3))))))
```

```python
    ))*special.gamma((1/2)-(1j*(w-(wr/(M*(M_sun*(G/(c**3))))))/(alpha/(M*(M_sun*(G/(c**3)))))))*special.gamma(1+((1j*w)/(alpha/(M*(M_sun*(G/(c**3))))))))))

    cross = np.zeros(len(frequency_array))
    return {"plus": Zp, "cross": cross}


#### Define the Likelihood according to what bilby likes
class My_Likelihood(bilby.Likelihood):

    def __init__(self, interferometers, waveform_generator, priors=None):

        super(My_Likelihood, self).__init__(dict())
        self.interferometers = interferometers[0]
        self.waveform_generator = waveform_generator
        self.priors = priors

    def priors(self):
        return self.priors

    def log_likelihood(self):

        waveform = self.waveform_generator.frequency_domain_strain(self.parameters)
        residual = self.interferometers.frequency_domain_strain - \
                        self.interferometers.get_detector_response(waveform, self.parameters)
        psd = self.interferometers.power_spectral_density_array
        duration = self.waveform_generator.duration

        log_l = -2.0 / duration * np.sum((np.conj(residual)*residual) / psd)

        return log_l.real

    def noise_log_likelihood(self):

        noise = self.interferometers.frequency_domain_strain
        psd = self.interferometers.power_spectral_density_array
        duration = self.waveform_generator.duration

        log_l = -2.0 / duration * np.sum(np.abs(noise)**2 / psd)

        return log_l.real


#### Define the sampling frequency and the data duration
sampling_frequency = H1.sampling_frequency
duration = H1.duration


#### Call the waveform_generator to create our waveform model.
waveform = bilby.gw.waveform_generator.WaveformGenerator(
    duration=duration,
    sampling_frequency=sampling_frequency,
    frequency_domain_source_model=echo
)


prior = bilby.core.prior.PriorDict()
prior['R'] = bilby.core.prior.Uniform(name='Reflection', minimum=0.0,maximum=1.0)
prior['d'] = bilby.core.prior.Uniform(name='Compactness', minimum=0.0, maximum=70.0)
prior['M'] = bilby.core.prior.Uniform(name="Mass", minimum=10, maximum=80, unit="$M_{sun}$")
prior['D'] = bilby.core.prior.Uniform(name="Luminosity Distance", minimum=0.1, maximum=1.0, unit="$Gpc$")
prior['wr'] =  0.3737
prior['wi'] =  -0.08896
prior['alpha'] =  0.2161
prior['t_0'] =  -0.001
prior['phi'] =  0.0
## Specifying the parameters of antenna pattern
prior['ra'] = 2.19432
prior['dec'] = -1.2232
prior['psi'] = 0.532268
prior['geocent_time'] = time_of_event


#### Instantiate the Likelihood
likelihood = My_Likelihood(interferometers=[H1], waveform_generator=waveform, priors=prior)


#### launch sampler
result2 = bilby.core.sampler.run_sampler(
    likelihood,
    prior,
    sampler="dynesty",
    npoints=500,
    walks=5,
    nact=3,
    outdir="GW150914_search",
    label="ECHO_search",
    dlogz=0.01
)


#### This will automatically show the signal-to-noise Bayes factor
#### Plot the corner plot
result.plot_corner()


#### Plot the recovered signal
idxs = H1.strain_data.frequency_mask  # This is a boolean mask of the frequencies which we'll use in the analysis
plt.figure(figsize=(10, 6))
plt.loglog(H1.frequency_array[idxs],
           H1.amplitude_spectral_density_array[idxs], label='H1 ASD', alpha=0.5, color='blue')
```

```python
plt.loglog(waveform.frequency_array[idxs], np.sqrt(waveform.frequency_array[idxs])*
        np.abs(waveform.frequency_domain_strain()['plus'][idxs]), label='Best-Fit', color='tab:red')
plt.fill_between(waveform.frequency_array[idxs], np.sqrt(waveform.frequency_array[idxs])*
                 np.abs(waveform.frequency_domain_source_model(waveform.frequency_array, 0.92-0.04, 35.88-1.78, 15.83-0.52,
                              0.78-0.15, 0.3737, -0.08896, 0.2161, -0.001, 0)['plus'])[idxs],
                 np.sqrt(waveform.frequency_array[idxs])*
                 np.abs(waveform.frequency_domain_source_model(waveform.frequency_array, 0.92+0.03, 35.88+1.23, 15.83+0.81,
                              0.78+0.14, 0.3737, -0.08896, 0.2161, -0.001, 0)['plus'])[idxs],
                 alpha=0.6, color='tab:green')

plt.xlabel('Frequency [Hz]')
plt.legend(framealpha=0.6)
plt.ylabel("Strain Amplitude")
plt.title("Best-Fit template")
plt.grid(True, which="both", ls=":")
plt.tight_layout()
#plt.savefig("./Freq_domain_bestfit_vs_psd.png", dpi=300)
plt.show()


#### Calculate the SNR
sig = (1/(2*np.pi))*np.sqrt(waveform.frequency_array[idxs])*np.abs(waveform.frequency_domain_strain()['plus'][idxs])
snr = np.sqrt(4/H1.duration * np.sum((likelihood.interferometers[0].frequency_domain_strain[idxs]*np.conj(sig))/
                                      likelihood.interferometers[0].power_spectral_density_array[idxs]).real)

print("The SNR:", snr)
```

Listing 3. GW150914 python snippet

Appendix D: Obtaining the data and PSD

```python

import numpy as np
import matplotlib.pyplot as plt
import bilby
from gwpy.timeseries import TimeSeries


def get_H1(time_of_event, post_trigger_duration, duration, psd_duration_multi):

    H1 = bilby.gw.detector.get_empty_interferometer("H1")

    #### Definite times in relation to the trigger time (time_of_event), duration and post_trigger_duration
    analysis_start = time_of_event + post_trigger_duration - duration
    print("Analysis start time:", analysis_start - time_of_event)
    print("Data segment:", analysis_start - time_of_event, analysis_start + duration - time_of_event)
    #### Use gwpy to fetch the open data
    H1_analysis_data = TimeSeries.fetch_open_data(
    "H1", analysis_start, analysis_start + duration, sample_rate=4096, cache=True)


    #### Initializing the interferometer with strain data
    H1.set_strain_data_from_gwpy_timeseries(H1_analysis_data)


    #### Downloading the Power Spectral Data
    psd_duration = duration * psd_duration_multi #32
    psd_start_time = analysis_start - psd_duration
    print("PSD start time:", psd_start_time - time_of_event)
    print("PSD segment:", psd_start_time - time_of_event, psd_start_time + psd_duration - time_of_event)
    H1_psd_data = TimeSeries.fetch_open_data(
    "H1", psd_start_time, psd_start_time + psd_duration, sample_rate=4096, cache=True)


    #### Specifying PSD by proper windowing using psd_alpha used in gwpy
    psd_alpha = 2 * H1.strain_data.roll_off / duration
    print("PSD alpha:", psd_alpha)
    H1_psd = H1_psd_data.psd(fftlength=duration, overlap=0, window=("tukey", psd_alpha), method="median")


    #### Now Initializing the interferometer with PSD
    H1.power_spectral_density = bilby.gw.detector.PowerSpectralDensity(
    frequency_array=H1_psd.frequencies.value, psd_array=H1_psd.value)


    #### Neglcting the high frequency part at it's a downsampling effect as we are
```

```
    using 4096 Hz
    H1.maximum_frequency = 1024
    print("Neglecting the high frequency part at", H1.maximum_frequency, "Hz as it's a downsampling effect as we are using 4096 Hz data.")


    return H1
```

Listing 4. A python snippet describing how the data and PSD was aquired and used in the injection and GW150914 search. This was stored as H1.py which was imported in both the above mentioned runs.