Interim Report 1: Using adaptive filtering to track noise lines or signals with varying frequency and amplitude

Rebecca White
University of Southern California

Ling Sun
California Institute of Technology

(Dated: July 10, 2020)

Advanced Laser Interferometer Gravitational-Wave Observatory (LIGO) and Virgo have observed gravitational wave (GW) signals from compact binary coalescences. Continuous waves (CW) and long-transient waves have not yet been detected in the LIGO-Virgo observing runs despite us knowing of their existence from theory. One of the issues with the LIGO data that plays a role and makes it challenging to detect these narrow-band waves is the instrumental spectral lines which obscure any astrophysical signals at the frequencies where they occur. This project explores an adaptive filter, named "iWave", as an alternative method of tracking these instrumental lines so they can be better factored out of the data as well as exploring the sensitivity of directly tracking weak CW or long-transient signals using iWave.

I. INTRODUCTION

On September 14th 2015, Advanced Laser Interferometer Gravitational-Wave Observatory (LIGO) detected its first gravitational wave (GW) event, GW150914 [3]. This event was detected in the first observing run (O1) of Advanced LIGO (from September 12th, 2015 to January 19th, 2016). After O1 concluded, the second observing run (O2) began on November 30th, 2016 and ended on August 25th, 2017 [5]. During the O2 run the binary neutron star (BNS) merger event, GW170817, was detected [4]. The third observing run (O3) had three confirmed mergers – GW190412 [6], GW190425 [7], and GW190814 [8]. Throughout the entire O1–O3 runs, there have been 14 confirmed GW events, most of them binary black hole (BBH) mergers.

The raw data from the LIGO detectors contain noise and glitches, so one of the most important aspects of LIGO data analysis is detector characterization and noise modeling. Without being able to monitor the detector or track the noise accurately, the significance of an event could be incorrectly estimated. A significant part of LIGO noise is instrumental lines, which can obscure astrophysical signals at the frequencies where they occur. They cannot be factored out easily and some are not very well understood. Some may even have wandering frequencies and/or varying amplitudes, which makes them even more difficult to track [2].

Adaptive filtering is a dynamic approach for characterizing the features in input data, including noise lines or signals with wandering frequencies and amplitudes. This method is very helpful when the input signal changes over a period of time. It could prove help in analyzing the detector interferometric data, tracking the varying instrumental lines over the run. A phase locked loop (PLL) is a control system that uses an oscillator to produce an output signal at a frequency and phase synchronized with the input signal. iWave is a hybrid method of a traditional PLL and an adaptive filter that is used for line tracking. This allows us to combine the wandering frequency/amplitude aspects of adaptive filters with the oscillation frequencies of PLLs to help us better track these spectral lines over a period of time.

There are other types of GWs in addition to those from compact binary coalescences (CBCs). Continuous waves (CWs) are produced by a single spinning neutron star. There are also long-transient GW signals that can be produced by a post-merger remnant from a BNS merger. Burst GWs are not yet very well understood since they are difficult to model. Once detected, however, they will be able to reveal a significant amount of astrophysical information. Stochastic background GWs from the early evolution of the universe are another type of weak signals remaining to be detected [1]. Existing techniques, e.g., the hidden Markov model tracking, can also track wandering signals and has been used for tracking both instrumental lines as well as CW and long-transient GWs [9, 12, 13]). Such methods mainly work in the frequency domain while iWave could prove to be an alternative method for analyzing the data in time domain.

In this project, we aim at using iWave to successfully track instrumental lines, investigate if iWave can subtract the noise lines and hence clean the data, and if iWave can identify weak synthetic GW signals in the data.

II. PROPOSAL OBJECTIVES

In the initial proposal, three main stages are laid out in the Approach section. The first stage is to test if iWave works with the publicly available GW data by: running iWave with the data containing some confirmed instrumental lines, comparing the cleaned and uncleaned spectrograms of the data, and analyzing the output of iWave. The second stage is to use iWave to identify some not-well-understood lines in the data and study certain lines appearing over different time periods by: studying these unknown noise artifacts and quantifying how well iWave can clean the data as well as comparing our results to other studies. The third stage is to test if iWave is sensitive to weak synthetic CW or long-transient GW signals by: checking the root-mean-square (rms) error between the injected and recovered signals and quantifying the strain sensitivity.

III. PROGRESS

Throughout the first 3 weeks, progress has been made on all the objectives mentioned in Section II as well as finishing the Weeks 1–2 benchmark that was under the Schedule section in the original proposal which consists of: setting up an environment on the cluster, reading literature in more detail, and running sample tests with iWave.

A. First Stage

I ran tests in the iWave directory and then rewrote the static and multi-line tracking tests to get a better understanding of both. Figure 1 is the plot output of the rewritten static test. The red sine wave is the input signal. The blue wave is the q_{out}, which is the output of iWave phase shifted by $\pi/2$ to be the quadrature phase. The green line represents $A_{\text{out}} = \sqrt{d_{\text{out}}^2 + q_{\text{out}}^2}$ where d_{out} is the direct output of iWave and A_{out} is the amplitude of the output which exponentially increases until it reaches the input signal's amplitude. The black curve is the input signal subtracted by d_{out}, i.e., the error signal.

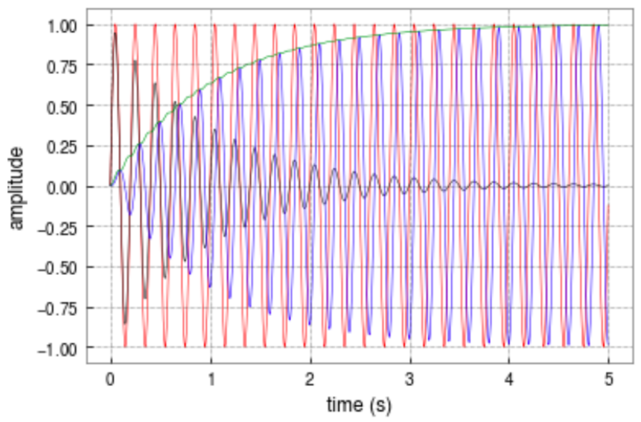

FIG. 1. iWave Static Test. The red sine wave is the input signal with a frequency of 5 and amplitude of 1. The blue wave is the q_{out} signal. The green line is the amplitude of the output. The black wave is the error signal, which becomes less than 0.01 at about 4.62 seconds.

As for the multi-line tracker, I added in a third frequency to see how well iWave would track three lines. I experimented with the two input parameters to see the effect on iWave's ability to track: (1) the initial guess of the signal frequency f_{guess}, and (2) the characteristic timescale of the signal τ. As a starting point, Figure 2 sees a clear lock on because the f_{guess} values which iWave starts its search at are 35 Hz, 32 Hz, and 30 Hz which are the same as the frequencies injected that iWave should be tracking. However, when increasing or decreasing the f_{guess} value for all the lines, iWave seems to begin tracking the closest line even if it is a duplicate as shown in Figure 3 and Figure 4. All these results prove that iWave can lock onto multiple signals. However, the changing f_{guess} results prove that we can put multiple f_{guess} values into iWave and have it lock onto the same line. If two spectral lines are close in frequency, iWave might not be able to track both unless the f_{guess} values are far enough apart so iWave may track both. After this, I tried experimenting with iWave using 30 Hz, 50 Hz, and 75 Hz frequencies for lines. I tried f_{guess} values that were 10–25 Hz away from the real frequencies and still saw a lock on from iWave. In Figure 5, the f_{guess} values are 10 Hz, 60 Hz, and 100 Hz while the lines are at the 30 Hz, 50 Hz, and 75 Hz. What this shows is that, without noise, iWave is able to lock onto a signal even when the f_{guess} given to it is 10–25 Hz away from the actual frequency.

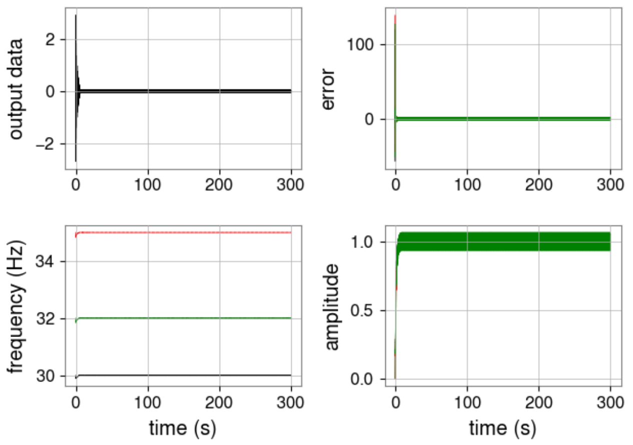

FIG. 2. iWave Multi-Line Test with Default f_{guess} and τ Values. In this, iWave f_{guess} values are set to the exact frequencies of each line (30 Hz, 32 Hz, and 35 Hz) and $\tau = 1$. The top left graph is iWave's output. The top right graph is iWave's error signal when it comes to tracking the 3 lines. The bottom left graph is the frequency throughout the output time series. The bottom right graph is the amplitude of the output data. As seen by the frequency and error graphs, iWave locks onto all three signals.

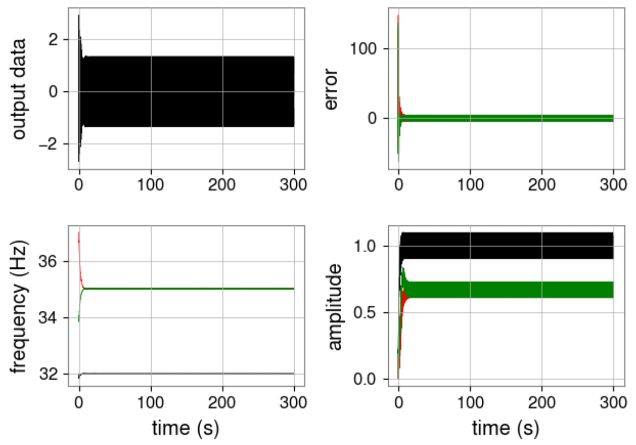

FIG. 3. iWave Multi-Line Test with f_{guess} Values Increased by 2 Hz and τ=1. The f_{guess} values are 32 Hz, 34 Hz, and 37 Hz, which is 2 Hz more than the actual frequencies of 30 Hz, 32 Hz, and 35 Hz. As seen in the bottom left frequency estimate plot, iWave only locks onto the 32 Hz and 35 Hz signals since those are the closest lines to the f_{guess} values despite this meaning that the 35 Hz line is tracked twice.

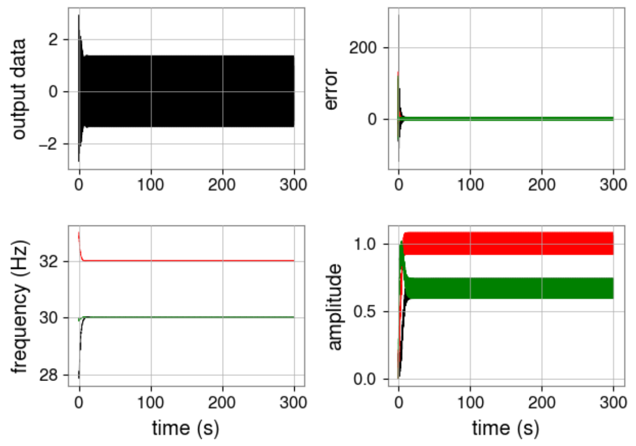

FIG. 4. iWave Multi-Line Test with f_{guess} Values Decreased by 2 Hz and τ=1. The f_{guess} values are 28 Hz, 30 Hz, and 33 Hz which is 2 Hz less than the actual frequencies of 30 Hz, 32 Hz, and 35 Hz. As seen in the bottom left frequency estimate plot, iWave only locks onto the 30 Hz and 32 Hz signals since those are the closest lines to the f_{guess} values despite this meaning that the 30 Hz line is tracked twice.

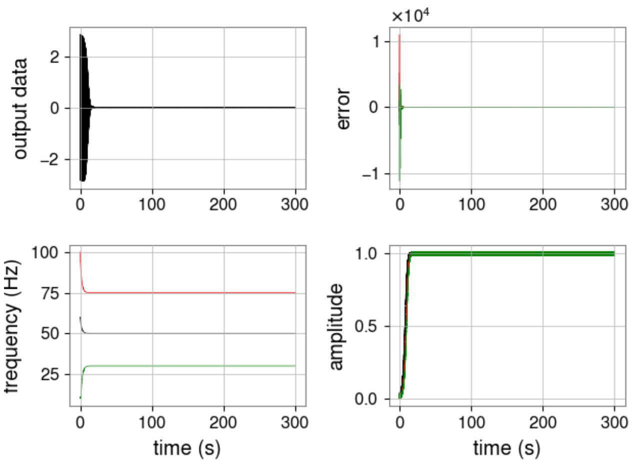

FIG. 5. iWave Multi-Line Test Using Inputs of 30 Hz, 50 Hz, and 75 Hz. This is iWave tracking 3 lines at 30 Hz, 50 Hz, and 75 Hz with f_{guess} values at 10 Hz, 60 Hz, and 100 Hz respectfully. As seen by the frequency estimates, iWave locks onto all three lines despite the Hertz difference between the actual frequencies and f_{guess} values.

B. Second Stage

One thing that is currently being worked on is seeing how well iWave can clean the data. Timesh Mistry of the iWave group developed a matlab script that uses the matlab version of iWave to track four lines in interferometric data. Figure 6 shows the Livingston raw data pre-filtering on the left as well as the raw data from 1000 Hz to 1020 Hz since that is the area we are focusing on. Figure 7 contains plots displaying the parameters of the pre-iWave filter on the left side and the data after using those filters. The filtered data allows iWave to track the lines without as much of an error signal as well as limits iWave to focusing on the 1000 Hz to 1020 Hz frequency band (which can be clearly seen in Figure 8 where the filtered data is plotted in red on top of the raw data). Lastly, the iWave output can be clearly seen in Figure 9. The frequency estimates for the lines do not change much, however the amplitude estimates and error signals do see much variance throughout the time series [11].

Currently, I am working on replicating these results using a python jupyter notebook and the python version of iWave to see if there are any differences between the two iWave versions.

C. Third Stage

I ran some tests with an injected long-transient GW signal in Gaussian noise to see what iWave would do. Changing the τ and f_{guess} values effected iWave's ability to track the signal in the noise. The default τ and f_{guess}

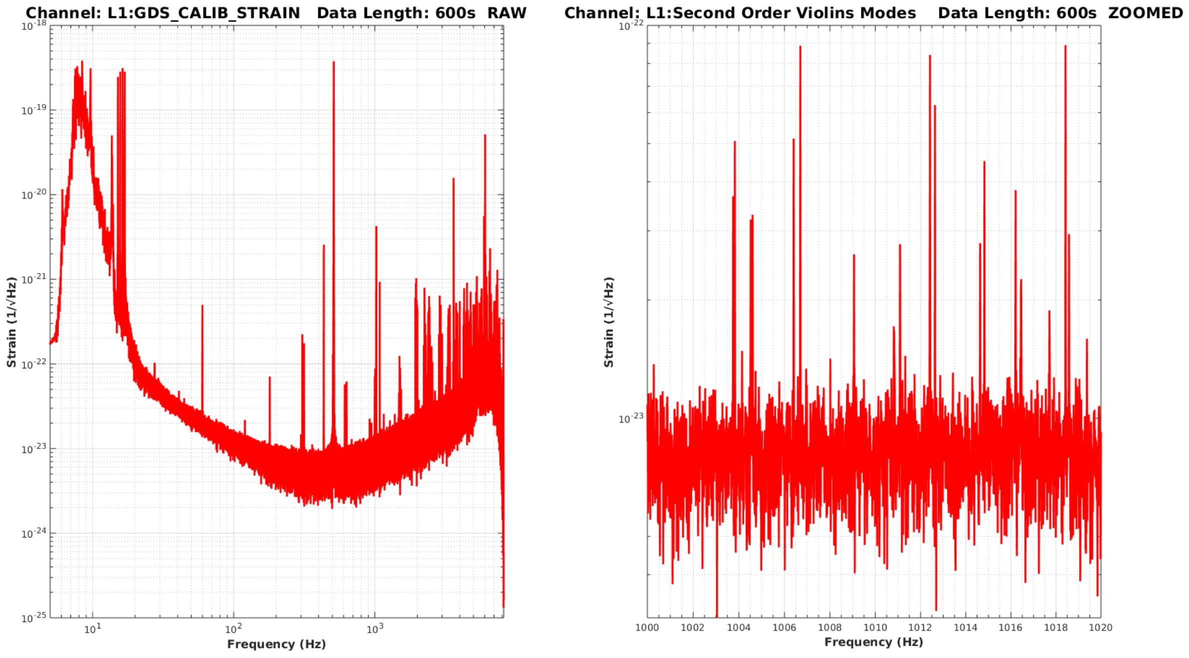

FIG. 6. Livingston Raw Data. This 1st graph is of the raw data spanning the whole frequency series. However the 2nd graph is only of the 1000 Hz to 1020 Hz frequency band. This is because the 1000 Hz to 1020 Hz frequency band is what we are focusing on for iWave to search for lines [11].

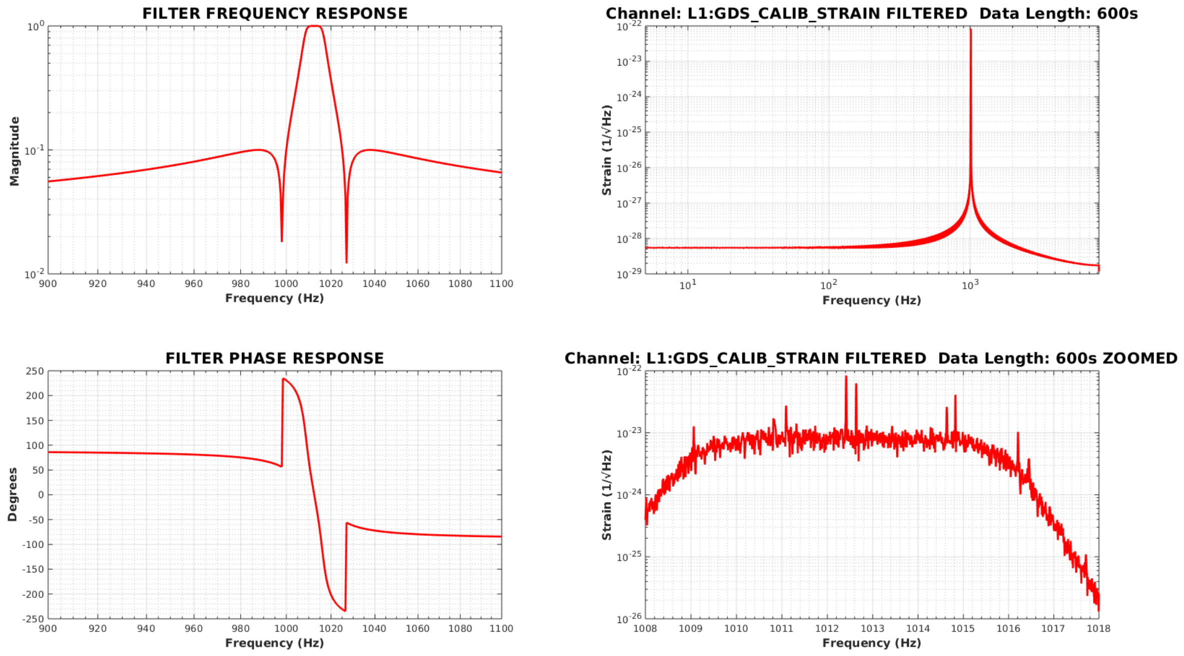

FIG. 7. Filter Bode Plot and Filtered Data. The left two plots are the bode plots for the filter. This shows the frequency and phase responses of the filters placed on the raw data. Changing parts of the filter will lead to different filtered data results and might lead to less accurate iWave results. The right two plots are the filtered data. The top is the filtered data over the whole frequency series while the bottom is the filtered data over the 1000 Hz to 1020 Hz frequency band [11].

values are 1 and 1000 Hz respectfully, and the output iWave produces can be seen in Figure 10. Calculating

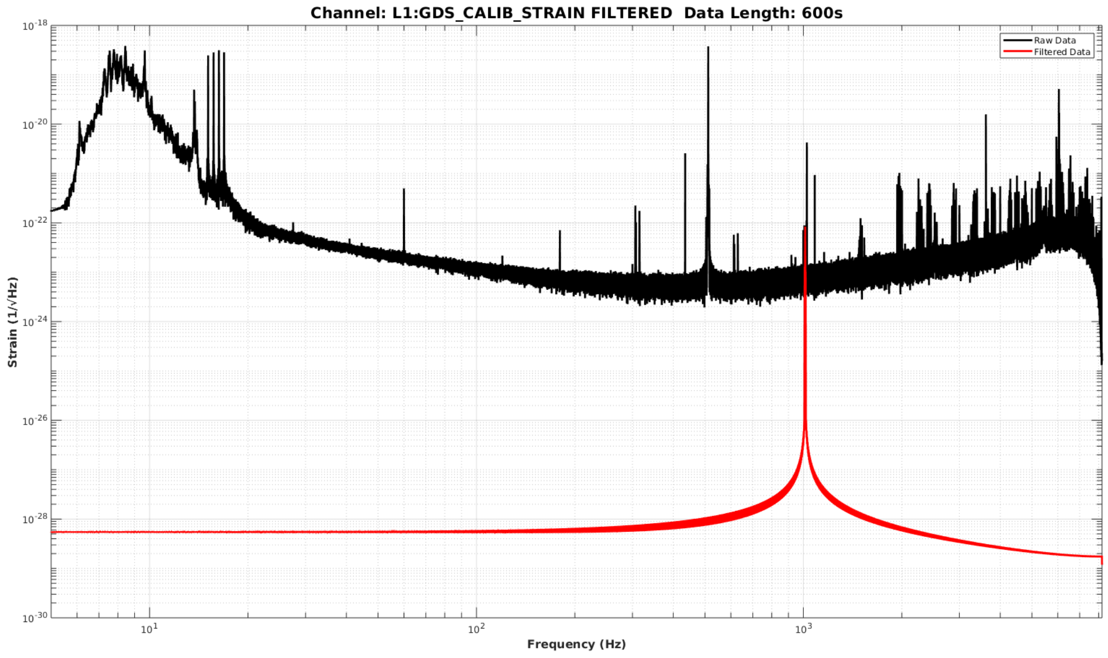

FIG. 8. Comparing the Filtered Data to the Raw Data. This clearly shows how the filtered data takes out all the unnecessary frequencies for our analysis. All the frequencies we do not need are filtered out and, using this filtered data, we can just focus on the 1000 Hz to 1020 Hz frequency band that we want [11].

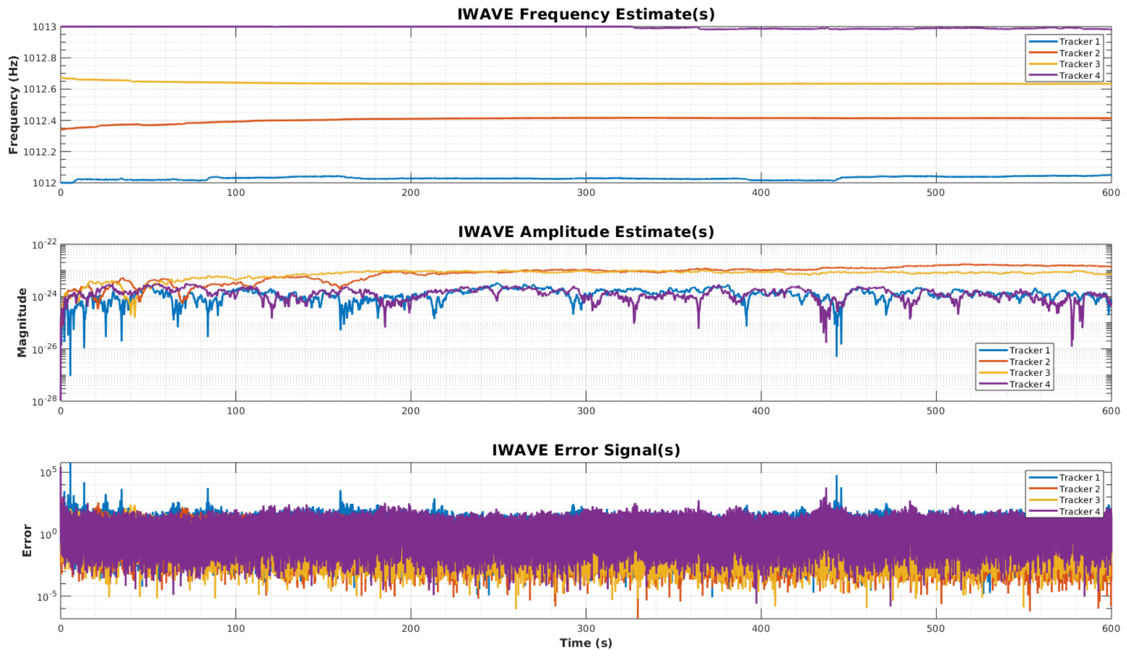

FIG. 9. iWave Output. Throughout the time series, the iWave trackers stay relatively constant in terms of the frequency estimates as seen in the top graph. The amplitude estimates for the trackers (seen in the middle graph) seem to get more constant as the time domain increases. The error signal throughout the entire time series seems larger than ideal, however no rms has been calculated for this test [11].

the rms for this signal once iWave locks on around the 200 second mark, we get 2097.6663629315462, which is

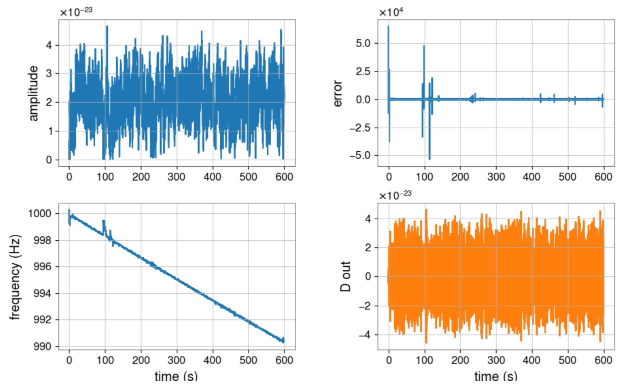

FIG. 10. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1$ and $f_{\text{guess}} = 1000$ Hz. This is the baseline for all the τ and f_{guess} changes for exploring more about iWave's abilities when it comes to this injected signal in this Gaussian noise. The top left graph is the amplitude of the iWave output. The top right graph is the error signal over the time series. The bottom left graph is the frequency estimate of iWave. The bottom right graph is the output of iWave. As seen by the error and frequency graphs, iWave is able to generally lock onto the signal between the 100 and 200 second marks. The calculated rms from 200 to 600 seconds is 2097.6663629315462.

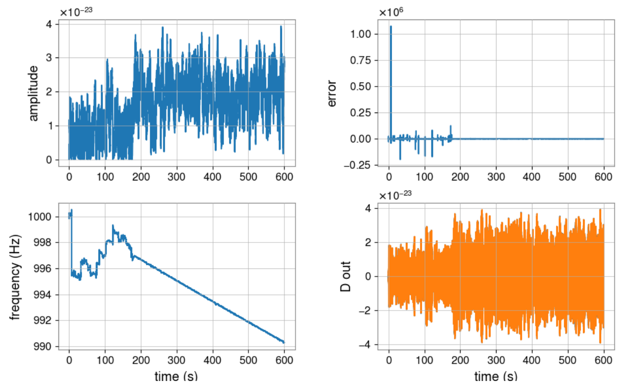

FIG. 11. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1.5$ and $f_{\text{guess}} = 1000$ Hz. As seen by the error and frequency graphs, iWave tracks the wandering line in the test frame GW data and can lock on between the 100 and 200 second marks. The calculated rms from 200 to 600 seconds is 1816.1134661860237, which is less than the rms when $\tau = 1$.

relatively high for what the error plot suggests.

Changing the τ value by the same order of magnitude did not make much of a difference in iWave being able to lock on as seen in Figure 11 where the f_{guess} value is the same 1000 Hz but τ is 1.5. Using these values, the calculated rms is surprisingly less than when tau was 1.

Increasing the f_{guess} value by 5Hz to be 1005 Hz lead to the outputs seen in Figure 12 (where τ is 1) and Figure 13 (where τ is 1.5). Neither of these parameters allowed iWave to be able to lock onto the signal.

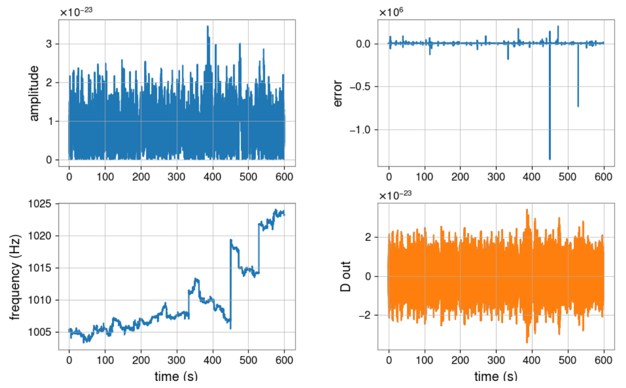

FIG. 12. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1$ and $f_{\text{guess}} = 1005$ Hz. As seen by the error and frequency graphs, iWave is unable to lock onto the signal when the f_{guess} is 1005 Hz and τ is 1. This is because the f_{guess} value is too far away from the signal for iWave to see and lock onto it.

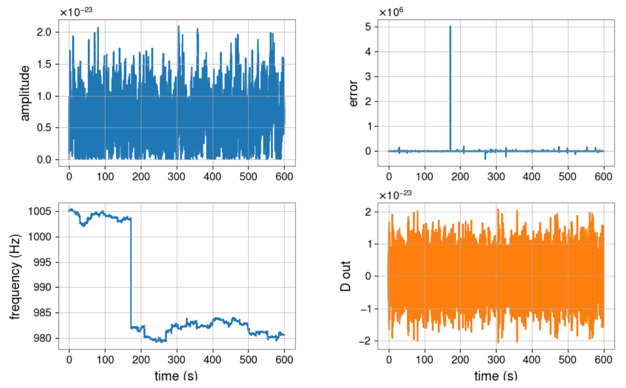

FIG. 13. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1.5$ and $f_{\text{guess}} = 1005$ Hz. As seen by the error and frequency graphs, iWave is unable to lock onto the signal when the f_{guess} is 1005 Hz and τ is 1.5. This is because the f_{guess} value is too far away from the signal for iWave to see and lock onto it.

Changing the f_{guess} value to be 5 Hz smaller than the default 1000 Hz allows iWave to able to lock onto the signal as seen in Figure 14 (where τ is 1 and f_{guess} is 995 Hz) and Figure 15 (where τ is 1.5 and f_{guess} is 995 Hz). However, this may be because the line iWave is tracking crosses through the 995 Hz f_{guess} frequency value chosen.

In order to make sure the reason they locked on was not because the line iWave is tracking crosses through the 995 Hz f_{guess} frequency value chosen, Figure 16 and Figure 17 use an f_{guess} value of 990. In these examples, the τ value matters since when $\tau = 1.5$ (while more accurate when the f_{guess} is 1000 Hz or 995 Hz) iWave is unable to lock onto the signal when the f_{guess} is 990Hz. This proves that we will need to try multiple combinations of τ and

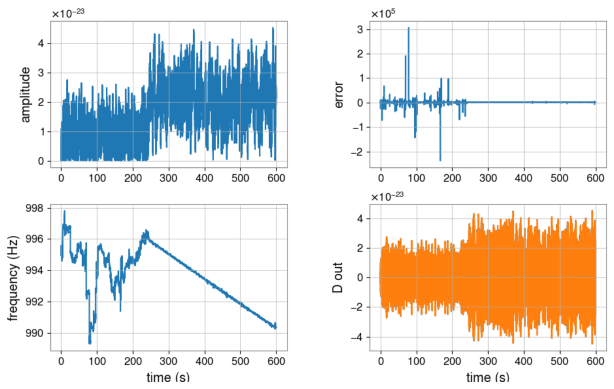

FIG. 14. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1$ and $f_{\text{guess}} = 995$ Hz. As seen by the error and frequency graphs, iWave is able to lock onto the signal between 200 and 300 seconds when the f_{guess} is 995 Hz and τ is 1. The calculated rms from 300 to 600 seconds is 1882.9930918236535.

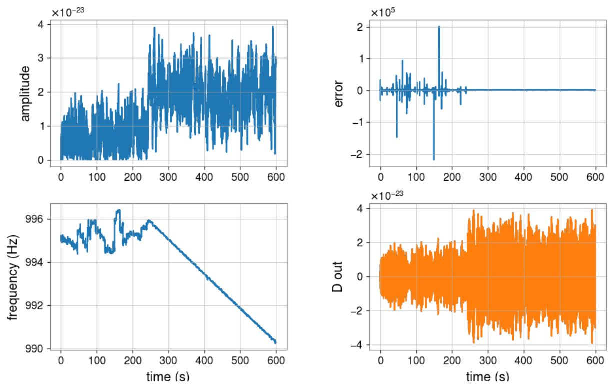

FIG. 15. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1.5$ and $f_{\text{guess}} = 995$ Hz. As seen by the error and frequency graphs, iWave is able to lock onto the signal between 200 and 300 seconds when the f_{guess} is 995 Hz and τ is 1.5. The calculated rms from 300 to 600 seconds is 1630.6827184273648.

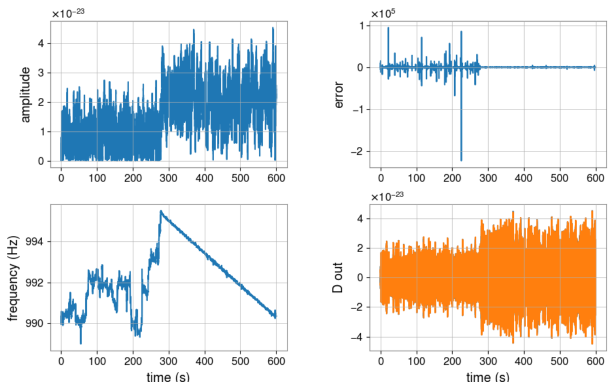

FIG. 16. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1$ and $f_{\text{guess}} = 990$ Hz. As seen by the error and frequency graphs, iWave is able to lock onto the signal between 200 and 300 seconds when the f_{guess} is 990 Hz and τ is 1. The calculated rms from 300 to 600 seconds is 1882.9930569463222.

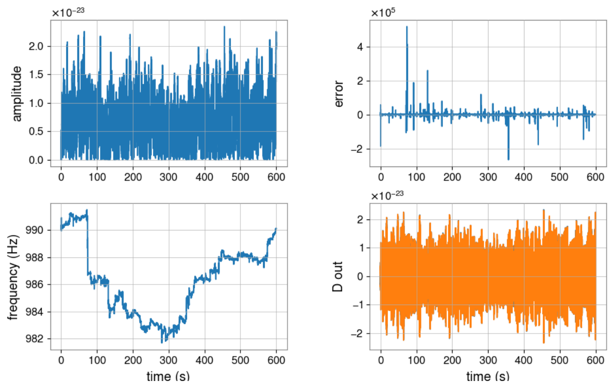

FIG. 17. iWave Outputs for Injected signal in Gaussian Noise when $\tau = 1.5$ and $f_{\text{guess}} = 990$ Hz. As seen by the error and frequency graphs, iWave is unable to lock onto the signal when the f_{guess} is 990 Hz and τ is 1.5. This shows that while an f_{guess} value can work for a certain τ value, having iWave lock onto a signal is about finding the right balance between the f_{guess} and τ.

f_{guess} values to make sure we are using the best value setup for the data.

Another part where some progress has been made is on plotting the GW data on a spectrogram so we can better view iWave's impact on the spectral lines. While iWave has not been used to clean Figure 18 and the spectrogram is of the publicly available GW data for that event, having this initial spectrogram to compare results to is helpful.

I have also checked the rms error on injected and recovered signals in terms of the static/multi-line tracking tests that were testing iWave's ability to track lines without any background noise. The rms error is something that needs to be further studied.

IV. CHALLENGES

Most of the challenges encountered in this project so far stem from inexperience using gwpy. Therefore, a large amount of time was devoted to the gwosc/gwpy/PyCBC tutorials in the GW Open Data Workshop [10]. I also made a jupyter notebook where I experimented with different gwpy.TimeSeries commands to see which ones will be useful and get a better grasp over it. The spectrogam and bandpass filter commands seem particularly useful in terms of seeing iWave's impact as well as limiting the frequencies iWave is looking at to improve accuracy.

Another setback that I have faced while running these scripts is the Kernel's constant restarting and inability to finish all of the jupyter notebook. One solution that

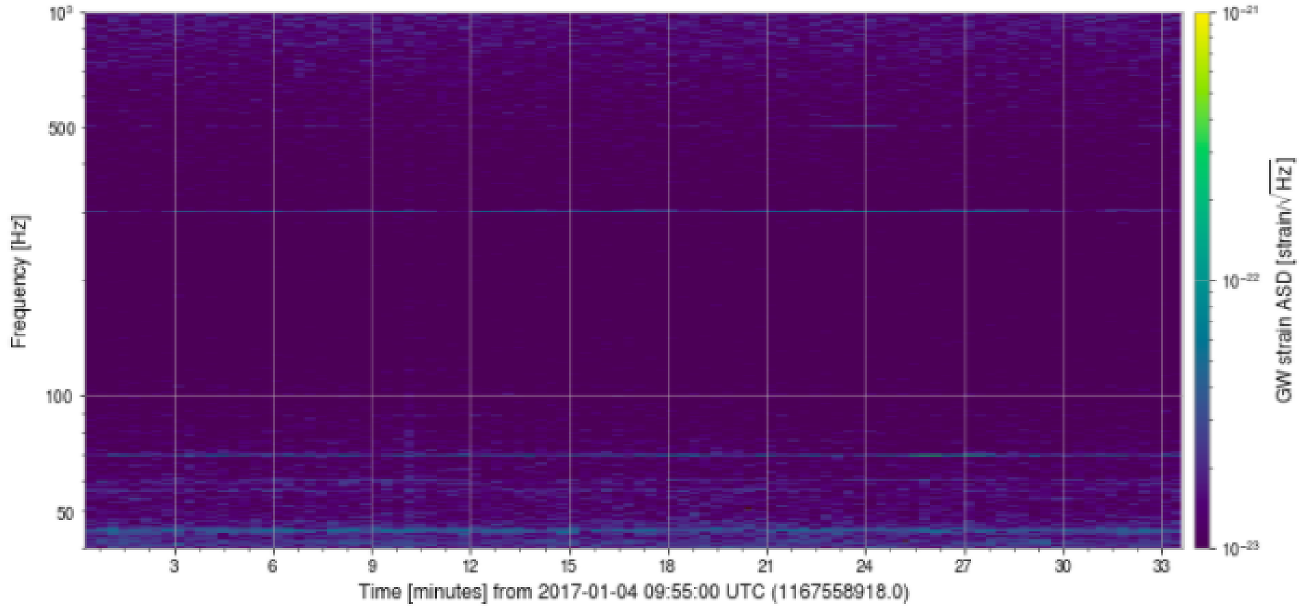

FIG. 18. Spectrogram of GW170104. In this spectrogram of GW event GW170104, there is a clear line around 300 Hz. This line does not show up in the publicly available list of spectral lines and other noise sources. Characterizing these unknown/lesser known lines will help us better understand these spectral lines.

I have not looked into fully is using `jupyter2.ligo.caltech.edu` to run the python notebooks. One way that I have currently been getting around this setback is putting my code into .py files and running them from the command line. This solution is better than waiting for the notebooks to load because of the large amounts of computation that might be necessary for any given script.

Lastly, one challenge that we had to deal with near the beginning and is something we need to look more into is the differences between the python and matlab versions of iWave. In the beginning, the new version of iWave (iwave 3.1) did not contain a file that was needed for its python installation. This has since been patched and we were able to get around this setback by copying the file from Ling's version from when she first installed iWave. However, currently Mistry is working with iWave on matlab which has slightly different commands than the python version. Matlab iWave is more developed so it will be interesting to monitor any changes between our results and his.

V. NEXT STEPS

I would like to finish the python version of Mistry's matlab script and quantify the rms from iWave's output as soon as possible before the iWave meeting on Friday July 10th. If there are any discrepancies with iWave's performance, they can be brought up and discussed at the meeting. This script, once finished, can also be a launching point in terms of tracking not well-known lines and multiple lines at the same time to clean up the data to satisfy the Stage 2 main objective.

As seen in Figure 18, I made a spectrogram of GW170104. I would like to use the GW170104 data to see if iWave can clean the line seen around 300 Hz. Once this is complete, I would like to develop spectrograms for other GW events and see how well iWave can filter out some of the lines on those spectrograms.

Lastly, I would like to explore producing signal data and seeing how well iWave can detect injected signals within Gaussian noise.

[1] Sources and types of gravitational waves.

[2] B P Abbott et al. A guide to LIGO–virgo detector noise and extraction of transient gravitational-wave signals. *Classical and Quantum Gravity*, 37(5):055002, feb 2020.

[3] B P Abbott and others. Observation of gravitational waves from a binary black hole merger. *Phys. Rev. Lett.*, 116:061102, Feb 2016.

[4] B P Abbott and others. Gw170817: Observation of gravitational waves from a binary neutron star inspiral. *Phys. Rev. Lett.*, 119:161101, Oct 2017.

[5] B P Abbott and others. Gwtc-1: A gravitational-wave transient catalog of compact binary mergers observed by ligo and virgo during the first and second observing runs.

Phys. Rev. X, 9:031040, Sep 2019.

[6] B P Abbott and others. Gw190412: Observation of a binary-black-hole coalescence with asymmetric masses, 2020.

[7] B P Abbott and others. GW190425: Observation of a compact binary coalescence with total mass ~ 3.4 m $\odot$. *The Astrophysical Journal*, 892(1):L3, mar 2020.

[8] B P Abbott and others. GW190814: Gravitational waves from the coalescence of a 23 solar mass black hole with a 2.6 solar mass compact object. *The Astrophysical Journal*, 896(2):L44, jun 2020.

[9] Joe Bayley, Chris Messenger, and Graham Woan. Generalized application of the viterbi algorithm to searches for continuous gravitational-wave signals. *Phys. Rev. D*, 100:023006, Jul 2019.

[10] jkanner. Gw open data workshop 3: Hands-on exercises, 2020.

[11] Timesh Mistry, Jun 2020.

[12] Ling Sun and Andrew Melatos. Application of hidden markov model tracking to the search for long-duration transient gravitational waves from the remnant of the binary neutron star merger gw170817. *Phys. Rev. D*, 99:123003, Jun 2019.

[13] S. Suvorova, L. Sun, A. Melatos, W. Moran, and R. J. Evans. Hidden markov model tracking of continuous gravitational waves from a neutron star with wandering spin. *Phys. Rev. D*, 93:123009, Jun 2016.