Implementing New Veto Analysis Methods in the PyCBC Search for Compact Binary Coalescences

LIGO Caltech SURF Program 2020, Mentor: Derek Davis

Brina Martinez[1]

[1]University of Texas Rio Grande Valley, Brownsville, TX 78520, USA

E-mail: brina.martinez@ligo.org

Abstract. The PyCBC search pipeline has been used since the first detection made by LIGO and continues to be used today in the search for gravitational waves. PyCBC runs a matched-filtering and chi-squared (χ^2) test to determine significant signal-to-noise ratios and compares triggers to previously modeled templates. With current methods of veto analysis which aim to remove and mitigate the effects of glitches with terrestrial origin in the PyCBC search for gravitational waves the possible removal of a hidden signal, a decrease in significance of signals, and a decrease in volume of searches from both strong and weak signals can be seen. To tackle these issues we will be testing veto methods based on multiple tools which include Gravity Spy, Hveto, and iDQ. We will analyze how simulated signals are recovered and calculate the change in background and sensitivity of the search. At this stage of our work we have focused on understanding how a few parts of PyCBC work, likelihood probabilities, reading in data and interpreting the results of our plots.

1. Introduction/Background

The Laser Interferometer Gravitational-Wave Observatory (LIGO) [1] discovered gravitational waves (GW) with the first detection of a binary black hole (BBH) collision, GW150914 [2]. Since then, there have been three separate observing runs along with detector improvements that have increased the rate of detections to multiple times per week in the third observing run (O3). So far, 14 confident detections have been announced [3][4][5].

The PyCBC pipeline has been used since the first detection made by LIGO [6]. PyCBC identifies GW events that are produced by compact binary coalescences (CBCs) and determines how significant each event is as compared to the noise in the detectors. PyCBC uses matched filtering to compare the data against templates that model what GW detection should look like and re-weighs the relationship with the estimated power spectral density (PSD) of the detectors involved. Matched filtering also calculates the signal-to-noise ratio (SNR) of our templates. PyCBC then uses a χ^2 test to filter our SNR and remove triggers that do not match our templates very well. PyCBC also uses gating, coincidence tests, and measures the false alarm rate (FAR) of recorded events [7]. PyCBC's time-slides method identifies and compares triggers from the two detectors. A detection seen by both LIGO detectors can help us calculate a network SNR in which we can determine our FAR. A significant FAR can help us determine the likelihood of seeing our detection again within the same network SNR, so if our FAR is decreased the less chance our detection is due to terrestrial noise.

With the amount of triggers that are produced by matched filtering, we can sometimes find loud glitches that are short in duration sneak across data quality tests. This results in a decrease in search sensitivity through two mechanisms, dead-time and ringing of the match filter [7][8]. As detections become more frequent, the quality and confidence of these detections needs to increase. This project, if successful, will be automated and implemented in the PyCBC search pipeline along future observation runs by LIGO as a data quality tool to improve the search for GWs.

2. Objectives

The goals for this project in assessing Data Quality (DQ) flags include:

- Producing a veto analysis method to remove as many glitches as possible from the data to increase the significance of signals.
- Making sure to remove as little time (data) as possible to reduce the chance of accidentally removing a gravitational wave signal.
- Be able to apply the new veto methods to PyCBC triggers and calculate an improved change in background and sensitivity of the search.

We will evaluate how the probability and distribution of triggers change with respect to time and how we can focus on the times that are interesting to improve PyCBC. We will also analyze how the PyCBC search responds to different configurations and DQ flags. This includes working with tools which include Gravity Spy [9], Hveto [10], and iDQ [11] to develop data quality flags and vetoes. Once we see an improvement in searches, we will know which direction we should continue to follow.

3. Work Plan/Schedule

From the beginning of this project until now a few changes in the timeline have occurred regarding steps we will take to tackle our goals and when we will take them and that is due to a better understanding of where we are currently with investigations and how they are flowing.

Before official start date: Watch videos and read papers to familiarize myself with the background of PyCBC, Hveto, data quality, and anything pertaining to our main goal.

Weeks 1-2: Become familiar with using PyCBC and importing data and learn to recognize first hand how to analyze plots, and figure out what needs to be improved when necessary. We will work on tutorials regarding the parameters PyCBC works with such as likelihood, FAR, and SNR significance.

Weeks 3-4: We will continue to work on tutorials to understand data and begin practicing the use of DQ flags. We will produce plots needed to analyze results and work on the interim report.

Weeks 5-6:We will begin to develop vetoes, both source-specific and non-binary. We will begin working more directly with the PyCBC pipeline.

Weeks 7-8: We will begin to evaluate our VT and implement iDQ. We will utilize work from previous weeks and decide whether or not techniques were effective and continue to apply methods and run tests.

Weeks 9-10: I will produce a final report that includes the nature of our project and its objectives, the methods we implemented, any figures that are related to our testing and results followed by references and acknowledgements. Produce a final presentation that will be 15 minutes and includes information on the project and why it is important, methods we implemented and how we did them, and finally the results obtained and ideas for future work. Acknowledgements included at the end.

4. Progress/current work

The first few weeks of the SURF program I worked on becoming familiar with PyCBC, statistics, data quality, and other parameters that relate to our goals. This includes learning how to understand the data I produce such as plots and how to apply small details into the overall project.

4.1. Understanding PyCBC time-slides and Data Quality flags

Before my official start date in June I familiarized myself with PyCBC by practicing a tutorial that uses match-filtering, DQ flags, and time slides to remove glitches and reveal hidden signals. Time-slides work by sliding data from one detector against data of the other in our case the Hanford (H1) and Livingston (L1) detectors. We check for coincidences between the two. If a gravitational wave is present in both detectors our time-slides would help in finding detector specific glitches and coincident data that would occur at different times at both detectors, helping us make sure we do not remove our signal. DQ flags are segments of time singled out that contain glitches we do not want. When we identify these glitches we are then able to reduce their impact on the search and make our signal more significant. There are different ways we can set up a DQ flag and choose how we want to single them out but in this investigation we will be looking at our whitened auxiliary data above a threshold and within windows of time segments to identify our peaks. In this investigation I am analyzing a data set that includes simulated random Gaussian noise that is recolored to mimic noise properties we could find in either one of the LIGO detectors. The data includes three different types of injections to analyze:

- Simulated sine-Gaussian bursts, which are similar to common, short duration glitches present in LIGO data.
- Simulated gravitational-wave signals with a limited bandwidth, to represent a 'worst case scenario' glitch.
- Simulated gravitational-wave signal, which we will be trying to identify from our filtering and DQ flags.

In this tutorial we want to not only calculate a significant signal but also calculate a significant FAR.

Figure 1 shows whitened auxiliary data which include our signal along with glitches that have very high significance and loud SNR's. From the time series, we are able to pick a threshold and window size we believe will identify peaks and correlate them with our DQ flags. Once we apply our DQ flags, threshold, and windows to the time series we are able to generate time-slides, re-plot our original data, and compare it to the original curve that does not contain our DQ flags. After applying our DQ flag, we can see our signal becomes louder

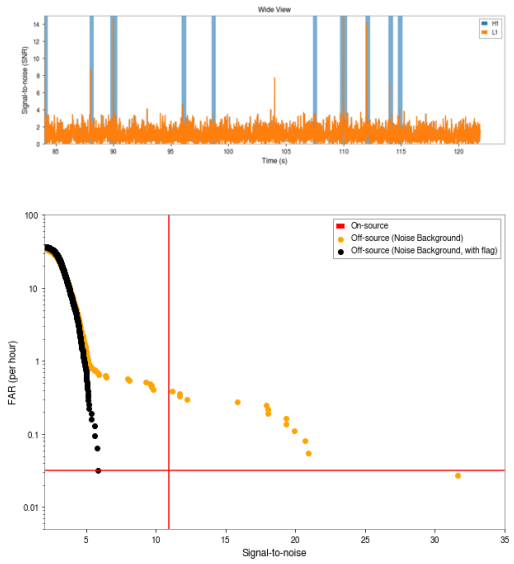

Figure 1: In the plot we are able to see a time series containing data with glitches and a signal which we are not able to tell apart. To help our signal SNR become more significant than the SNRs of our glitches we applied DQ flags and used time-slides that PyCBC uses to find coincidences between the data from our two detectors H1 and L1. We then established a threshold where the whitened auxiliary data is above the value we choose, and we establish windows to highlight those peaks, making sure not to highlight our signal. Our results compare our data before and after using the DQ flags and time slides and we can see a significant increase in significance for our signal compared to the background. In the time series we can see blue bars highlighting the peaks we want to remove without highlighting our signal which is also a significant peak. In the SNR vs FAR plot we can see our original background in orange that was extremely loud compared to our signal and our new background in black that significantly reduced and made our signal the loudest part of our data. Our original FAR was 0.3526 per hour. After applying our time-slides and DQ flags our FAR is now 0.0319 per hour, significantly lower.

than any of the glitches and background.

4.2. Understanding Likelihood

In my first week of SURF I ran an investigation to familiarize myself with how likelihood functions work and how it looks given PyCBC data. Likelihood shows us the probability of

how often an outcome, which we can label x, is expected to occur in a given model, which we can label θ.

$$\mathcal{L}(\theta \mid x) = p_\theta(x) \tag{1}$$

This outcome can occur frequently given our model, giving us a high likelihood or can be very unlikely to occur, giving us a low likelihood. Likelihood is important for determining the odds of our data. When we know the odds we can further use them to determine the ranking statistic and calculate the FAR as an output. For this investigation we wanted to compare the likelihood of astrophysical vs random noise at a given SNR. We begin by plotting our two data sets and making sure they are normalized and carry the same SNR model, we then calculate them against each other as pictured in Figure 2 revealing the SNRs that are more likely to contain astrophysical or random noise. As our SNR increases, we begin to see our astrophysical noise is more likely to occur.

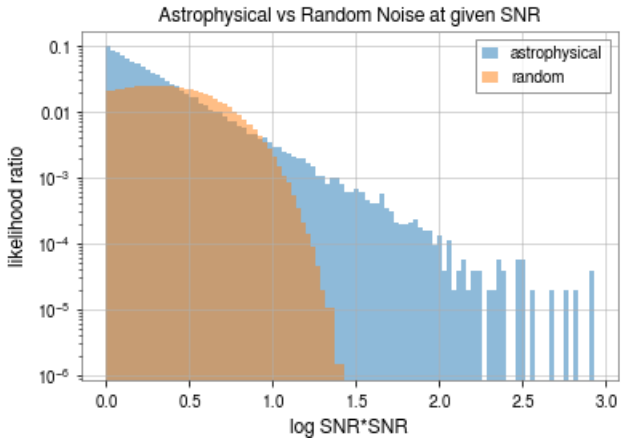

Figure 2: In the plot we are able to see areas where our astrophysical data which is shown in blue or random noise data which is shown in orange has more likelihood of occurring depending on the SNR. We are able to see that above log SNR^2 = 1.2 our likelihood ratio is in favor of being astrophysical. In between log SNR^2 0.5 and 1.0 we see that our random noise has a higher likelihood.

4.3. Understanding how to work with hdf5 files

In my second week of SURF I worked on a learning to read data from Hdf5 files. Hdf5 files are great for holding large amounts of data and store the data in a hierarchical format which uses folders in folders and layers like a directory. In this investigation I looked at data from L1 containing different parameters such as SNR, template duration, end times, and χ^2 degrees of freedom, to name a few. With hdf5 files you can read into what are called different keys or folders until you reach the end of a folder that contains no more groups of sub folders and only contains data sets and we are able to work with the data sets by converting them into a numpy array. For our first plot we wanted to analyze an interesting region of end times vs SNR. We filtered our data sets by zooming into regions that most of our triggers were located and SNR was significant. In Figure 3's right plot we can see there are interesting triggers located below the SNR = 5.25 threshold and different time periods with varying amounts of triggers. A reason we might see this threshold is due to our pipeline: certain consistency tests only focus on triggers with SNR > 5.25, which is why only those below that value remain. To further evaluate the triggers we applied a χ^2 filter which works as a consistency test to

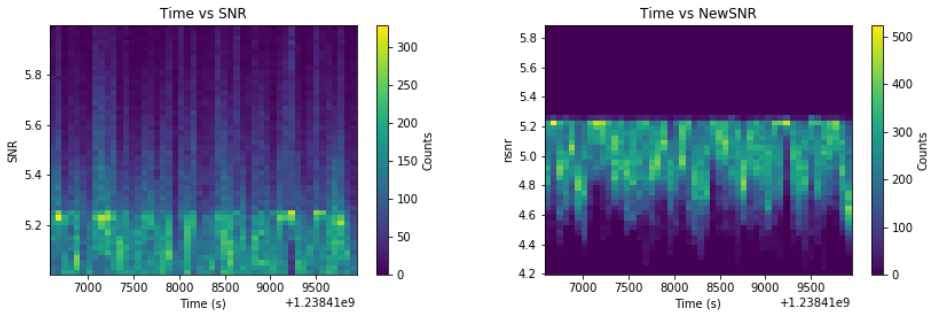

Figure 3: In the plots above we accessed data from an hdf5 file containing triggers from L1. In the left plot the first thing we noticed was the defined threshold line created by our pipeline showing us which SNRs carry more triggers. In the plot we can see that SNRs > 5.25 contain a majority of the triggers we want to look at. We are able to analyze how different variations among these SNRs vs time contain more or fewer triggers and use this to understand our data better. In the right plot we used a χ^2 filter to measure which SNRs fit our model well and down weighed SNRs that did not fit well. We can see this gave us a better view of our triggers and we can evaluate them even further by applying more filters and use our new SNR to rank significance.

measure how well our data fits with our model. This results in a re-weighted SNR that down weighs the triggers that are labeled as a bad fit. The right plot in Figure 3 shows us that after applying the χ^2 filter, we are able to zoom in and view more significant triggers and how they vary across time. We can further evaluate the brighter spots and why more triggers are found in certain time segments than others.

4.4. Learning to submit a workflow

For this project we will be working with many triggers from different observing run periods and I will need to produce files containing large quantities of data which can normally take very long to produce. To help make the process of gathering data go faster I learned how to submit a workflow in PyCBC to the clusters from my laptop which reduced what normally would take a long time to only a day. To practice submitting a workflow we chose to gather data we believed would be interesting to evaluate and eventually filter and analyze with our hdf5 file notebook. For this task we made use of the LIGO data grid and Pegasus (Planning for Execution in Grids) which works to generate an executable workflow.

5. Challenges

One of the challenges I faced when completing the tutorials was learning how to interpret what the plots were showing me and determining which ones were significant enough to look further into. Since I have not worked previously with likelihood, filtering, and PyCBC I struggled to interpret the data and sometimes even manipulate the data to create values to plot. Learning to take a step back and look at the overall picture of what I was working with is something I could further improve on. Another challenge I faced was doing unnecessary work, sometimes values for my plots did not match and I spent quite a while trying to figure out the problem with the data myself before realizing I was working with it wrong on my end, whether it was from missing a filter or forgetting to normalize data.

Potential challenges I believe I will encounter as I move along are understanding which

methods are more useful than others. This includes quickly realizing which results need to change and finding useful solutions quickly.

6. Acknowledgments

Computing support for this project was provided by the LDAS computing cluster at the California Institute of Technology. LIGO was constructed by the California Institute of Technology and Massachusetts Institute of Technology with funding from the National Science Foundation, and operates under cooperative agreement PHY-0757058. This work carries LIGO Document number T2000349-v2.

References

[1] J. Aasi et al. Advanced LIGO. *Class. Quant. Grav.*, 32:074001, 2015.
[2] B.P. Abbott et al. Observation of Gravitational Waves from a Binary Black Hole Merger. *Phys. Rev. Lett.*, 116(6):061102, 2016.
[3] B.P. Abbott et al. 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(3):031040, 2019.
[4] B.P. Abbott et al. GW190425: Observation of a Compact Binary Coalescence with Total Mass $\sim 3.4 M_{\odot}$. *Astrophys. J. Lett.*, 892:L3, 2020.
[5] R. Abbott et al. GW190412: Observation of a Binary-Black-Hole Coalescence with Asymmetric Masses. 4 2020. arXiv:2004.08342.
[6] B.P. Abbott et al. GW150914: First results from the search for binary black hole coalescence with Advanced LIGO. *Phys. Rev. D*, 93(12):122003, 2016.
[7] Samantha A. Usman et al. The PyCBC search for gravitational waves from compact binary coalescence. *Class. Quant. Grav.*, 33(21):215004, 2016.
[8] B P Abbott et al. Effects of data quality vetoes on a search for compact binary coalescences in Advanced LIGO's first observing run. *Class. Quant. Grav.*, 35(6):065010, 2018.
[9] Michael Zevin et al. Gravity Spy: Integrating Advanced LIGO Detector Characterization, Machine Learning, and Citizen Science. *Class. Quant. Grav.*, 34(6):064003, 2017.
[10] Joshua R. Smith, Thomas Abbott, Eiichi Hirose, Nicolas Leroy, Duncan Macleod, Jessica McIver, Peter Saulson, and Peter Shawhan. A Hierarchical method for vetoing noise transients in gravitational-wave detectors. *Class. Quant. Grav.*, 28:235005, 2011.
[11] Reed Essick, Patrick Godwin, Chad Hanna, Lindy Blackburn, and Erik Katsavounidis. iDQ: Statistical Inference of Non-Gaussian Noise with Auxiliary Degrees of Freedom in Gravitational-Wave Detectors. 5 2020. arXiv:2005.12761.