Fscan Code Development For Advanced Ligo Detector Characterization

Cody Arceneaux
University of Mississippi

Follow this and additional works at: https://egrove.olemiss.edu/etd

Part of the Physics Commons

Recommended Citation
https://egrove.olemiss.edu/etd/756

This Thesis is brought to you for free and open access by the Graduate School at eGrove. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of eGrove. For more information, please contact egrove@olemiss.edu.
FSCAN CODE DEVELOPMENT FOR ADVANCED LIGO DETECTOR
CHARACTERIZATION

A Thesis
presented in partial fulfillment of requirements
for the degree of Master of Science
in the Department of Physics and Astronomy
The University of Mississippi

by

CODY CHRISTOPHER ARCENEAUX

August 2015
ABSTRACT

The Advanced Laser Interferometer Gravitational-wave Observatory (LIGO) is an experiment designed to provide direct detection of gravitational waves. Its searches for periodic gravitational waves are highly susceptible to long-duration noise that appears as spectral lines. FScan is a tool for finding spectral lines and is used to identify the lines that appear in the gravitational-wave data; it is also used to characterize LIGO’s detectors. This thesis details work in rewriting the plotting portion of FScan, making improvements to the FScan driver script, and writing code to assist in the tracking of wandering spectral lines. In particular, this thesis presents the line tracking code, LineTrack.py, and serves as its documentation.
ACKNOWLEDGEMENTS

The author gratefully acknowledges the support of the United States National Science Foundation (NSF) for the construction and operation of the LIGO Laboratory. The author gratefully acknowledges the support of the NSF for provision of computational resources. The original work in this thesis was partially supported by the National Science Foundation through awards PHY-1067985 and PHY-1404139.

The author would also like to thank the individuals who have helped and supported him through this project and the writing of his thesis. The author gratefully acknowledges his advisor, Marco Cavaglià, who has aided him throughout his graduate career. The author gratefully acknowledges Gregory Mendell for his invaluable help in running FScan and providing ideas of what could be done to improve it, and the author gratefully acknowledges Duncan Macleod for helping him resolve some particularly troubling coding issues. The author gratefully acknowledges Shivaraj Kandhasamy and Daniele Trifirò for providing useful advice as he worked to complete this project. The author gratefully acknowledges the family and friends who have helped him, in particular Joey, Zachary, Melissa, Giovanni, Laura, and Shanmuka.
# TABLE OF CONTENTS

## ABSTRACT

## ACKNOWLEDGEMENTS

## LIST OF FIGURES

## 1 INTRODUCTION

1.1 Gravitational Waves ................................................................. 1
1.2 Laser Interferometer Gravitational-wave Observatory (LIGO) .............. 4
1.3 Data Analysis ................................................................. 8

## 2 DETECTOR CHARACTERIZATION

2.1 Detector Characterization .......................................................... 11
2.2 FScan ................................................................. 12

## 3 CODE DEVELOPMENT

3.1 Python Plotting Code ............................................................. 20
3.2 Driver Script Improvements .................................................... 21
3.3 Wandering Line Tracking ....................................................... 21

## 4 CONCLUSION

## BIBLIOGRAPHY

## VITA
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure Number</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Polarizations of Gravitational Waves</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>LIGO Detector</td>
<td>5</td>
</tr>
<tr>
<td>1.3</td>
<td>Detector Sensitivity Curve</td>
<td>7</td>
</tr>
<tr>
<td>1.4</td>
<td>Chirp Waveform</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>Plots Created with Python Plotting Code</td>
<td>18</td>
</tr>
<tr>
<td>2.2</td>
<td>FScan Results Page</td>
<td>19</td>
</tr>
<tr>
<td>3.1</td>
<td>Comparison of FScan Old and New Output Page Formats</td>
<td>22</td>
</tr>
<tr>
<td>3.2</td>
<td>Wandering Lines</td>
<td>23</td>
</tr>
<tr>
<td>3.3</td>
<td>LineTrack.py Diagram</td>
<td>28</td>
</tr>
<tr>
<td>4.1</td>
<td>Wandering Lines in a Spectrogram from GPS Time 1102334418</td>
<td>30</td>
</tr>
<tr>
<td>4.2</td>
<td>Wandering Lines in a Spectrogram from GPS Time 1102420817</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Wandering Lines in a Spectrogram from GPS Time 1102507217</td>
<td>32</td>
</tr>
</tbody>
</table>
CHAPTER 1

INTRODUCTION

1.1 Gravitational Waves

Einstein’s theory of general relativity describes gravity as an effect of spacetime curvature. The latter is related to the energy and momentum of the (non-gravitational) fields present in spacetime by

\[ G_{\mu\nu} = \frac{8\pi G}{c^4} T_{\mu\nu}, \]  

(1.1)

where \( G_{\mu\nu} \) is the Einstein tensor (which defines curvature), \( G \) is Newton’s gravitational constant, \( c \) is the speed of light in vacuum, and \( T_{\mu\nu} \) is the stress energy-momentum tensor of matter and radiation fields[1]. Often, calculations are done in units so that \( c = 1 \).

A spacetime interval is defined by a 4x4 tensor known as the metric, \( g_{\mu\nu} \). The metric of empty space is the Minkowski metric, \( \eta_{\mu\nu} \), which describes a flat spacetime.

When energy is present, the spacetime metric is determined by solving Eq. (1.1). It can be shown that Einstein’s equations can be cast in a wave-like form for small metric perturbations around the Minkowski metric. The solutions of this wave equation represent gravitational waves, i.e., weak, propagating distortions of spacetime. Gravitational waves can be generated by any accelerated mass distribution with non-zero quadrupole momentum[2]. This is an exciting prediction of Einstein’s theory of general relativity[1][2].

Quantitatively, the derivation of a wave equation for spacetime starts with a metric that is just a small perturbation \( (h_{\mu\nu} \ll 1) \) to the Minkowski metric.
\[ g_{\mu\nu} = \eta_{\mu\nu} + h_{\mu\nu}. \]  
\[ (1.2) \]

When \( T_{\mu\nu} = 0 \) and using a particular gauge known as transverse-traceless (TT)[1], Eq. (1.1) reduces, at first order in \( h \), to

\[ \Box h^{TT}_{\mu\nu} = 0, \]  
\[ (1.3) \]

where \( \Box \) is known as the D’Alembertian and is defined as

\[ \Box = \left( \frac{\partial^2}{\partial t^2} + \nabla^2 \right). \]  
\[ (1.4) \]

The plane-wave solution of Eq. (1.3) is

\[ h^{TT}_{\mu\nu} = C_{\mu\nu} e^{ik_\sigma x^\sigma}, \]  
\[ (1.5) \]

where \( C_{\mu\nu} \) is a constant, symmetric tensor and \( k_\sigma \) is the wave vector which satisfies \( k_\sigma k^\sigma = 0[1] \).

It can be shown that \( C^{\mu\nu} \) only has four non-zero components (and only two independent components due to symmetry). These two independent metric elements correspond to the two polarizations of the wave, \( h_+ \) and \( h_\times[1] \).

\[ C^{\mu\nu} = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & h_+ & h_\times & 0 \\ 0 & h_\times & -h_+ & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \]  
\[ (1.6) \]

To determine the effect of a gravitational wave and its different polarizations, it is helpful to observe a ring of particles. A wave with plus-polarization traveling along a direction perpendicular to the plane of the ring causes the ring to cyclically expand and
contract along the x- and y-axes with a period equal to the gravitational-wave period. A

cross-polarized wave causes an expansion and contraction of the ring along directions at

45° to the motion of the plus-polarization[2]. This is shown in Figure 1.1 for a full cycle.

The amplitude of a gravitational wave is usually described by the strain \( h \), i.e., the

relative change in distance between two test masses[1],

\[
h = \frac{\delta L}{L},
\]

where \( \delta L \) is how much the test masses are displaced from each other and \( L \) is the distance

between the test masses before the wave passes through the system. In the case of a
typical binary system of compact objects (binary black holes, binary neutron stars, or
black hole-neutron star systems) inspiraling into each other, the gravitational-wave strain

is expected to be on the order of \( h \sim 10^{-21} \) when the wave reaches Earth[2]. For test

particles separated on the order of \( 10^3 \) meters, the displacement between them is on the

order of \( 10^{-17} \) meters (for comparison, the Bohr radius is on the order of \( 10^{-11} \) meters). A
direct detection of gravitational waves requires an extremely sensitive detector[2].

Gravitational waves have not been directly observed yet, but binary systems of
neutron stars have been observed to lose energy consistent with gravitational radiation.
For example, the orbital period of the Hulse-Taylor binary system (a system of two neutron stars orbiting each other) has been observed to change in a manner consistent with energy loss due to the emission gravitational waves, and this provides an indirect proof for the existence gravitational waves[4].

1.2 Laser Interferometer Gravitational-wave Observatory (LIGO)

Advanced LIGO is an experiment designed to provide a direct detection of gravitational waves. It has two detectors: one in Livingston, LA, and one in Hanford, WA. Both detectors are Michelson laser interferometers with 4km long arms at 90° to each other and a Fabry-Perot cavity along the length of each arm.

The diagram of a LIGO detector is shown in Figure 1.2. The input laser beam is provided by a 180W Nd:YAG three-stage laser with a wavelength of 1064nm. The pre-stabilized laser (PSL) is made up of the laser and stabilization systems. The PSL is in a separate cleanroom from the rest of the interferometer and is situated before the laser enters vacuum[6]. A pre-mode cleaner (PMC) removes higher order modes by only allowing $TEM_{00}$ (the lowest order transverse electric mode) to pass through. The removal of higher order modes improves the overall quality of the beam by lowering beam pointing fluctuations. In addition, by behaving as a low pass filter, the PMC stabilizes the beam’s power at radio frequencies. After the PMC, the power is further stabilized in the 10 Hz to 10 kHz range (i.e., lower that the radio frequencies stabilized in the PMC) with an acousto-optic modulator[7].

The beam then travels to the input mode cleaner (IMC) which is comprised of three suspended mirrors (MC1, MC2, MC3)[6]. The IMC (like the PMC) limits the transmitted modes and reduces pointing instability[8].

From the IMC, the beam goes to the power recycling mirrors (PRM, PR2, PR3)[6]. The beam’s effective power is increased due to a resonant cavity formed by the three power recycling mirrors which return laser light coming from the interferometer back to the beam splitter (BS)[9].
Figure 1.2. LIGO Detector. This is a schematic diagram of the Advanced LIGO detectors[5]. From the PSL, the beam enters Horizontal Access Module (HAM) 1 and goes through the IMC and power recycling cavity in HAM 2 and 3. After going through the BS, the laser light enters the Fabry-Perot cavities and is reflected between ITMX/Y and ETMX/Y. After exiting the Fabry-Perot cavities, the beam goes through the signal recycling cavity in HAM 4 and 5, and then it travels to the OMC in HAM 6.
The beam enters the interferometer arms through the BS. The latter sends half of the laser down the x-arm and half down the y-arm. In each arm, a Fabry-Perot cavity makes the beam resonate between the input test mass (ITMX or ITMY, depending on which arm) and the end test mass (ETMX or ETMY)[6]. Light resonance in the cavity effectively increases the length of the interferometer by a factor of $\sim 100^{10}$.

The laser light that exits the Fabry-Perot cavities is then recombined from the x- and y-arms at the BS and heads toward the signal recycling cavity which, like the power recycling cavity, consists of three mirrors (SRM, SR2, SR3)[6]. The signal recycling cavity sends the signal back into the interferometer. Also, by adjusting the length of the signal recycling cavity, the interferometer may be operated in a narrowband configuration with increased sensitivity at certain frequencies[9].

From the signal recycling cavity, the laser light travels to the output mode cleaner (OMC). The OMC filters out any non-TEM$_{00}$ modes to ensure that the output photodetectors only receive light that carries the gravitational-wave signal[6].

Due to their high sensitivity, the detectors have numerous elements designed to reduce noise. For example, test masses are suspended with complex four-stage fused silica suspensions in order to reduce seismic and thermal noise; the vacuum chambers have both passive and active seismic isolation[6]. Because different noise sources limit the interferometer sensitivity over various frequency bands, the LIGO detectors are not equally sensitive over all frequencies. The Advanced LIGO sensitivity curve is shown in Figure 1.3.

Before science data can be taken, the detector needs to be in a locked state and be in science mode. For the interferometer to be locked, the optical cavities inside the detector have to achieve resonance. Intervals of noisy data must be flagged or vetoed. Additionally, operators in a detector’s control room flag times when the detector is not operating properly or when there is a large source of noise[6].
Figure 1.3. Detector Sensitivity Curve. This is an Advanced LIGO sensitivity curve based on noise and limiting noise sources. These include quantum noise (photon radiation pressure and statistical fluctuations), noise from mechanical and thermal dissipation on the optics’ coatings, and gravity gradients (seismic waves affecting density in the ground so that gravitational forces change) [6]
1.3 Data Analysis

When analyzing LIGO data for gravitational waves, gravitational-wave signals are searched for from four different categories of sources. These sources are compact binary coalescence (CBC) systems, sources that produce short-duration gravitational-wave signals (burst sources), a stochastic gravitational-wave background of astrophysical or cosmological origin, and non-axisymmetric rotating neutron stars (continuous or periodic sources).[11]

Gravitational waves from a CBC system are considered some of the most favorable for detection. CBC systems are pairs of neutron stars, pairs of black holes, or black hole-neutron star pairs. As the orbit of these two compact objects decays and they inspiral into each other, they emit gravitational waves. In the simplest case of two non-rotating objects, the gravitational-wave signal emitted by a CBC system is characterized by a waveform with increasing frequency and amplitude in time until the merger, also known as a “chirp”. Figure 1.4 shows an example of a “chirp”. Since these inspiral waveforms are well known and have been extensively modeled, the search method that is usually adopted is to compare the gravitational-wave data against a bank of templates.[12]

Burst sources emit short-duration gravitational waves, and these sources include
core collapse supernovae and gamma ray bursts. Unlike CBC searches, burst sources are un-modelled and are not searched for by comparing templates to the data. One search method is to coherently combine phase and amplitude of the output from multiple detectors; a signal in the data adds up while summing over multiple detectors, and another method is to analyze data from different detectors and search for coincidences between them[14].

The stochastic gravitational-wave background could arise from a superposition of remnants of the Big Bang and unresolved astrophysical sources, and searches for the stochastic gravitational-wave background look for long-duration signals. These searches use cross-correlation of the gravitational-wave data between two detectors, and a filter function designed to enhance frequencies where the stochastic background is expected to be strong.[15].

Gravitational waves from continuous sources tend to be relatively weak when compared to gravitational waves from CBC and burst sources. The gravitational-wave strain from rotating non-axisymmetric neutron stars goes as

\[ h \sim \frac{\epsilon \Omega I_{NS}}{r}, \]  

(1.8)

where \( \epsilon \) is the fractional asymmetry with respect to the rotational axis, \( \Omega \) is the rotational period, \( I_{NS} \) is the neutron’s star moment of inertia, and \( r \) is the distance to the neutron star. Although the values of these terms are uncertain, estimates for \( h \) for gravitational waves emitted by known rotating non-axisymmetric neutron stars are several orders of magnitude smaller than those emitted by CBC sources. Even though they are relatively weak, signals from continuous sources can be integrated over long time periods to increase the likelihood of detection[2]. Continuous wave searches use several programs including PowerFlux[16] and Einstein@Home[17]. Both are all-sky searches (which are computationally expensive). They each create thirty minute long short Fourier transforms.
(SFTs) from the calibrated gravitational-wave strain data, but how they process it afterwards differs[17].

The PowerFlux search uses the SFTs to get power in a particular portion of the sky for 501 adjacent frequency bins. The 501 power values in these frequency bins are then used to get a signal-to-noise ratio (SNR)[16].

The Einstein@Home search gets around its high computational cost by using distributed computing. Volunteers are able to sign-up and allow their computers be used to perform calculations when they are idle. The architecture for this is the Berkeley Open Infrastructure for Open Computing (BOINC)[18]. The Einstein@Home search calculates a detection statistic called the $\mathcal{F}$-statistic for each frequency, first-derivative of the frequency, right ascension, and declination in the thirty minute long SFT. Each $\mathcal{F}$-statistic is given a count of one if it is above a previously determined threshold, and the search uses a method known as the Hough search to get weighted counts for each frequency and position in the sky[17]. The final Hough number count is

$$n_c = \sum_{i=0}^{N_{seg}-1} w_i n_i$$

(1.9)

where $N_{seg}$ is the total number of thirty minute time segments, $n_i$ is the number of counts, and $w_i$ is a weight based off the detector noise for the $i^{th}$ segment. The top 10,000 candidates are then further analyzed in post-processing[17].
CHAPTER 2

DETECTOR CHARACTERIZATION

2.1 Detector Characterization

Due to the required sensitivity of the detector, reducing the effect of noise is imperative. Instrumental and environmental noise still has an effect despite the detector being designed to be isolated from noise sources. It is then necessary to identify when noise contaminates the gravitational-wave data.

The non-Gaussian and non-stationary nature of the detector noise may produce short-lived noise transients which significantly impact the searches for gravitational-wave transient signals, such as those from CBC or burst sources[11]. To improve the sensitivity of transient searches, data quality flags denote particularly noisy times that require extra analysis. In addition to the gravitational-wave channel, data is recorded from tens of thousands of auxiliary channels with environmental and instrumental information[19]. Data quality flags are created from instrumental and environmental auxiliary channels shown to have correlation with the gravitational-wave channel[11]. These flags can be created online by automated monitors and offline during data analysis. Additionally, operators in the control room can flag times when there are high amounts of noise[20].

Data quality flags can be used to veto times in order to keep noisy data from interfering with searches. These vetoes are categorized with a numerical scale of decreasing severity. For example, in the latest science run of initial LIGO, S6, four veto categories were defined. Category 1 vetoes defined times when the detector was not functioning.
properly; these vetoes meant the detector should not have been in science mode. Category 2 vetoes were created from channels that were known to correlate strongly with the gravitational-wave data, and category 3 vetoes were created from channels that had a less strong correlation with the gravitational-wave data[11]. Category 4 defined injected data for transient searches. Category 1 and 2 vetoes automatically excluded the matched segments from analysis[20].

Just as short-lived noise transients affect CBC and burst searches, long-duration noise affects stochastic background and continuous gravitational-wave searches[11]. Identifying spectral lines is necessary step of continuous wave and stochastic search pipelines. Since these searches analyze the data in the frequency domain over long periods of time, non-astrophysical spectral lines could impair continuous gravitational-wave searches[11].

Some lines are well-known and are either an artifact of the design of the detector or added purposefully[21]. For example, in the United States, the power grid frequency is 60 Hz, thus a 60 Hz line and its harmonics are visible in the LIGO data; resonant frequencies from the mirror suspensions show up as “violin modes” in the gravitational-wave data. Finally, there are lines that are injected into the data for calibration purposes[21].

When continuous wave searches identify a spectral feature, environmental channels are examined to track down its origins. For example, during commissioning of the Advanced LIGO PSL, unwanted spectral features were shown to arise from the PSL periscope[22]. Typical sources of noise affecting continuous-wave searches arise from CPUs, ethernet switches, and other detector electronics[21].

2.2 FScan

FScan is a tool for finding spectral lines[21]. It was originally designed to aid continuous wave searches, and it has also been used for detector characterization[21]. FScan can search for lines in auxiliary channels in addition to the gravitational-wave channel to characterize the detector’s subsystems.
FScan is often run over a day, a week, or a month of data. It is commonly used in
detector characterization investigations to compare data from environmental and
instrumental auxiliary channels to the gravitational-wave data so that the origin of
non-astrophysical noise can be determined[21].

2.2.1 FScan Code

FScan is comprised of a number of different programs working in tandem. A
resource file (.rsc) containing the initialization information for the FScan run is used as
input for a small script written in Tool Command Language (Tcl) called
generateMultiFscanHTML.tcl; an example of the resource file is below. The resource file
usually has some variation of the name FscanGenerator.rsc. The Tcl script converts the
information in the resource file to a format which is suitable for the other FScan
constituent programs. The output of the Tcl script is then fed into a driver script written
in Python, fscanDriver.py. The driver script runs the programs to find the relevant
data, make short-time Fourier transforms (SFTs) of the data, perform a spectrum average
with the SFTs, and plot the results.

```
1 # FscanGenerator resource configuration file.
2 # Set tcl list with list of channels, frame types, etc....
3 #Each item in the list contains: channel_name channel_type
4     IFO input_type output_sft_dir _chan comparison_snr
5     comparison_delta_dir knee_freq sft_time_baseline
6     start_freq band sub_band extra_time_for_data_find freq_res
7     alternative_dir_name
8 set ::masterList {
9 {L1:PSL-FSS_FAST_MON_OUT_DQ ADC_REAL4 L1 L1_R default none 0
10  0 0 1800 0 4000 100 32 1 default}\
11 }
12 
13 # If fixedComparison is 1 then used then ignore the
14     comparison delta dir in the masterList but compare using
15     fixed values:
16 set fixedComparison 0;
17 set fixedComparisonChanDir "";
18 set fixedComparisonString "";
```
The first part of the resource file in lines 4-5 is the “masterList” which lists the auxiliary channels that are to be analyzed and the analysis parameters. Each line in the masterList includes the channel name, the channel type, the interferometer (IFO, where \(H1 = \text{Hanford}\) and \(L1 = \text{Livingston}\), the type of data (including unmodified raw data, second trend data, or minute trend data\(^{[23]}\)), and the location for the produced SFTS.

The start frequency, frequency band, and frequency sub-band set what frequencies are analyzed and how the data is split up. For example, line 5 shows that the start frequency is set to 0 Hz, the end frequency is set to 4000 Hz, and the frequency sub-band is set to 100 Hz, the data is divided into forty parts ranging from 0 to 100 Hz, 100 to 200 Hz, and so on up to 3900 to 4000 Hz. If the knee frequency is set, a high pass filter at that frequency is applied to the data before generating SFTs. The SFT time baseline defines the length of each SFT (which is 1800 s in the example in line 5). The extra time for data
find (which is 32 s in the example) gives the defined time to the program that searches for the data extra time to add or subtract to the starting and ending GPS times so that it might more easily find the data. The frequency resolution (set to 1 Hz in the example) sets the resolution of the spectrogram data.

After the masterList, there is small section from lines (9 - 12) to determine if the results of the FScan run should be compared to a channel from previous run. The channel the current run is compared to in the comparison run is set in line 5 of the resource file; each channel in the current FScan run needs to have a channel entered in its entry in the masterList to compare to in the comparison run. This is done if fixedComparison is set to 1. The location of the comparison run is entered in fixedComparisonChanDir. The fixedComparisonString lists the IFO, the GPS start time, and the GPS end time in the form “ifo_StartTime_EndTime” for the comparison run. An example for results at the Livingston detector with start and end times of 1105228816 and 1105315216 is “L1_1105228816_1105315216”. The fixedComparisonSNR sets what normalized average power is used for the comparison. Any frequencies with normalized average power greater than fixedComparisonSNR in either FScan run are written to a file linked to the results pages; this file also contains coincident frequencies between the two FScan runs that have normalized average power greater than fixedComparisonSNR.

The resource file needs the full path to the driver script fscanDriver.py as shown in line 14. If the older Matlab version of the code that creates the plots is used, the location of Matlab needs to be provided as shown in line 15. The results are written in a subdirectory of the output directory which is specified in line 17. The starting and ending GPS times for this run are set in lines 19-20. The value in line 22 determines what time is used for the name of the subdirectory containing this run’s results; if a 1 is entered, the ending GPS time for the run is used for the subdirectory’s name, and if a 0 is entered, the time that the command to start FScan is entered will be used for the subdirectory’s name. If FScan is set to search for science segments in line 27, the driver script runs
**ligolw_segment_query** to get a list of GPS times that the interferometer was in science mode according to the data monitoring tool (DMT)[20]. This list contains starting and ending GPS times, and any times outside these segments are excised from the analysis.

A text file of time segments with starting and ending GPS times may be used as an additional input to **generateMultiFScanHTML.tcl**; it works similarly to the option in the resource file. Only times listed by the segment file are analyzed.

The information from the resource file and optional segment file is passed to the Python driver script. The driver script submits individual instances of FScan’s constituent programs as jobs to the data grid as part of the parallelization of FScan through Condor[24], a distributed computing system. The submissions through Condor include the program to run, its arguments, the environment, the output location, and the log location.

The data are split up based on the channels and frequency bands, and jobs are submitted by the driver script through Condor to search for the data using a program called **ligo_data_find** which provides a list of locations of the files that contain the channel data. A piece of C code called **lalapps_MakeSFTs** creates the SFTs from the requested data files. Another program written in C, **lalapps_spec_avg**, calculates a spectrum average from the SFTs, and these results are used to make the FScan plots. The driver script also creates the HTML results pages for each run with links to the files and plots the various parts of FScan create.

The plotting part of FScan, **plotSpecAvgOutput**, creates the spectrograms and the normalized average power plots. The data for the spectrogram is formatted as a matrix where the columns of the matrix are the time steps and the rows are the frequency bins. A cutoff value defined as

\[
\text{cutoff} = \text{median} + 5 \left( \frac{\text{median}}{\sqrt{\text{bins}}} \right),
\]

(2.1)

where **median** is the median value of the spectrogram data and **bins** is the number of frequency bins, is applied to the spectrogram data to avoid over-saturation of the plots.
The spectrogram frequency resolution is set in the resource file, and its time resolution is the SFT time baseline; both of these values are set in line 5 of the resource file. The spectrogram units are the channel’s units (volts, counts, etc.) per $\sqrt{\text{Hz}}$. If the optional segment file is used, the columns that correspond to times steps outside of the segments provided are all zero; these values are grayed out in the plot. The normalized average power plots are the power averaged over time for each frequency bin. Their frequency resolution is the inverse of the SFT time baseline in line 5. The average power values are then normalized against the median power of each frequency sub-band. A cutoff value of four is applied to the normalized average power data before plotting to avoid over-saturation. Additionally, the data used for the normalized average power plot is written to a file as a list sorted by the normalized average power with its corresponding frequency[25]. An example of the spectrogram and normalized average power plot is shown in Figure 2.1.

2.2.2 Running FScan

FScan is designed to run on the LIGO cluster. LIGO has computational resources at Caltech, the Livingston site, the Hanford site, and several other sites[26]. The frame files are available on the cluster in addition to software required to run FScan[26].

The FScan suite can be found in the LSC (LIGO Scientific Collaboration) Algorithm Library Suite (LALSuite)[27] package under lalsuite/lalapps/src/pulsar/fscan. After building and sourcing the code in LALSuite on an account on the LIGO cluster, the FScan environment must be set. If the bash (tcsh) shell is used, it is necessary to add the specific locations of several parts of FScan to .bashrc (.cshrc). The variable LSC_DATAFIND_PATH defines the location of ligo_data_find. MAKESFTS_PATH defines the location of MakeSFTDAG, which submits jobs to Condor for lalapps_MakeSFTs to create the SFTs. SPECAVG_PATH is the path to lalapps_spec_avg. PLOTSPECAVGOUTPUT_PATH defines the location of run_plotSpecAvgOutput.sh; this is a shell script that serves to run plotSpecAvgOutput with the appropriate inputs. For example, the typical FScan
Figure 2.1. Plots Created with Python Plotting Code. This is an example of a spectrogram created with the Python plotting code including the added units label. This is a spectrogram of the channel L1-OAF_CAL_DARM_DQ over a day. The part that is grayed out corresponds to times outside a segment file provided to FScan. The colors from blue to red indicate greater spectral amplitude density. It’s corresponding normalized average power plot is also included. Both of these plots were made with data that came from the Livingston interferometer during the sixth engineering run of Advanced LIGO (ER6) between the GPS times 1102334417 and 1102420817.

The output directory where FScan results are written is specified in line 18 of the resource file. The results directory cannot be empty for the first run, so it is suggested to create a “tmp” dummy file in the results directory with the command

$ touch tmp.

FScan can be run with the command line

$ ./generateMultiFscanHTML.tcl FscanGenerator.rsc [SEGMENT FILE] -R.

Once all the jobs submitted to Condor are completed, the user needs to run another
**Figure 2.2. FScan Results Page.** This is an example of the FScan results page from the sixth science run of initial LIGO (S6) for Livingston. To view the results, the date needs to be selected on the calendar followed by the channel name in the list of analyzed channels (in this case, only L1_LSC-DARM_ERR is available)[28].

TCL script named `generateMultiFscanHTML.tcl` to update the HTML files for the results pages with the command

```
$ ./generateMultiFscanHTML.tcl [RESULTS DIRECTORY]
```

The results pages use a nested structure of dates and channels to present the plots and text files. An example of the results page is shown in Figure 2.2.
CHAPTER 3

CODE DEVELOPMENT

The original work of this thesis consisted in rewriting the plotting portion of FScan, making improvements to the driver script, and writing code to assist in the tracking of wandering spectral lines.

3.1 Python Plotting Code

The plotting portion of the code, `plotSpecAvgOutput.py`, was re-written in Python using matplotlib, a Python plotting library[29]. The motivation for this work is easier integration of the FScan code with other LSC detector characterization tools due to a general trend towards the usage of the Python language across the LSC.

Another aspect of the work was to add a label with units to the spectrogram plots as shown in Figure 2.1 along with a normalized average power plot produced with the Python plotting code. Labels were also added to the text files written with the normalized average power data, and a column with the SNR was added; an example of this is shown in Table 3.1.

The python plotting code requires a version 1.3.1 of matplotlib or higher. If that is not available by default (as is the case on the cluster), one needs to install and source a local copy.
Table 3.1. Sorted Normalized Average Power Data

<table>
<thead>
<tr>
<th>Freq.</th>
<th>Normalized Avg Power</th>
<th>SNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>221.181667</td>
<td>375.5570</td>
<td>2151.6662</td>
</tr>
<tr>
<td>221.181111</td>
<td>342.9940</td>
<td>1964.6060</td>
</tr>
<tr>
<td>294.908333</td>
<td>266.7660</td>
<td>1526.7094</td>
</tr>
<tr>
<td>294.908889</td>
<td>228.4820</td>
<td>1306.7846</td>
</tr>
<tr>
<td>256.000000</td>
<td>125.5940</td>
<td>715.7380</td>
</tr>
<tr>
<td>221.182222</td>
<td>40.9843</td>
<td>229.6923</td>
</tr>
</tbody>
</table>

This is the sorted normalized average power data from the Python plotting code with the new SNR column. This table contains data that came from Livingston during ER6 between the GPS times of 1102334417 and 1102420817.

3.2 Driver Script Improvements

Improvements were made to the driver script, `fscanDriver.py`, to make the new Python plotting code better interface with the larger framework of FScan. An option for presenting the plots in different formats was added to the HTML pages. An example of the results page with both the old and new formats is shown in Figure 3.1.

In addition, the code was modified so that when the FScan run is compared to a previous run (see lines 9-12 of the resource file on page 13), the code is able to search for the most recently completed FScan run (FScan is considered to have successfully completed if the spectrograms and normalized average power plots were produced). To use this option, lines 10-11 of the resource file need to be set to

```
10  set fixedComparisonChanDir "~/home/pulsar/public_html/fscan/L1_PSL/L1_PSL/last/L1_PSL-FSS_TPD_DC_OUT_DQ/";
11  set fixedComparisonString "last";
```

Here, `fixedComparisonChanDir` is the full path to the FScan output directory, subdirectory, and channel for comparison. Setting `fixedComparisonString` to “last” causes the driver script to search for the most recently completed FScan run.

3.3 Wandering Line Tracking

FScan has the ability to find spectral lines that show up in the normalized average power data. The code is able to find any frequency with a normalized average power
Figure 3.1. Comparison of FScan Old and New Output Page Formats. This figure shows a comparison of the old and new FScan output page formats due to the FScan driver script additions[30]. Both formats remain as an option on the web page. The old format shows the spectrograms with increasing frequency sub-bands as one scrolls down. The new format option for the FScan output page allows the spectrograms to be viewed with decreasing frequency sub-bands to mimic a continuous frequency range. These pages show data from the Livingston interferometer during ER6.
greater than the value of fixedComparisonSNR in line 11 of the resource file. However, if a line wanders in frequency, it does not have a high normalized average power in a single frequency bin. An example of a wandering line in a spectrogram is shown in Figure 3.2. Its corresponding normalized average power lacks a line at the frequency. Wandering spectral lines may be missed by the methods FScan has to find lines, but they still require identification and characterization. Thus, LineTrack.py was written to search for them.

LineTrack.py looks at the data used to generate FScan spectrograms. The spectrogram data is the result of lalapps_spec_avg and can be found in the output directory of any FScan run; the filename of the spectrogram data has a format of spec_, followed by the starting frequency of the frequency sub-band contained in the file, the ending frequency of the frequency sub-band contained in the file, the interferometer name code (H1 or L1), the starting GPS time, and the ending GPS time of the FScan run. For
example, the file `spec_600.00_1200.00_L1_1111536016_1111622416` covers the 600.00 Hz to 1200.00 Hz frequency sub-band at the L1 (Livingston) interferometer between GPS times 1111536016 and 1111622416. The contents of the file are the spectral amplitude density of the data from the frame files in units of channel’s units per $\sqrt{\text{Hz}}$. `LineTrack.py` looks over all available frequency sub-bands for that time and channel’s output directory.

The wandering line tracking code can be run from the command line on any cluster with FScan output. For example, the command

```
$ ./LineTrack.py linetrack_L1OAF-CAL_DARM.ini
```

runs the LineTrack code with the initialization file `linetrack_L1OAF-CAL_DARM.ini`. The code can also be wrapped in a shell script to be run as part of a cronjob for automation. An example of the initialization file is shown below.

```
1 [when_where]
2 ; Start Time
3 tStart=1113026943
4 ; End Time
5 tEnd=1113034435
6 ; Full/path/to/directory/with/LineTrack/results
7 line_directory=/home/pulsar/public_html/fscan/L1_PSL/Tracking/Results/
8 ; Full/path/to/directory/where/Fscan/results/are/stored/
9 directory=/home/pulsar/public_html/fscan/L1_LSC/L1_LSC/
10 ; Channel to be looked at
11 channel=L1_OAF-CAL_DARM_DQ
12 ; IFO
13 ifo=L1
14 [Tracking]
15 ; How much of a found line needs to be above the threshold to be counted as a line (on a scale from 0.0 to 1.0)
16 tolerance=0.9
17 ; How much a line can wander over time to count it as the same line (in frequency steps)
18 wander=5
19 ; Number of standard deviations above the mean to calculate the threshold value for searching for lines
20 stdnum=3.0
```
LineTrack.py first locates which FScan run corresponds to the starting and ending GPS times provided in the initialization file in lines 3-5. It then finds the directory for the channel determined by lines 9-11 and creates a list of files with spectrogram data. The code then loops over this list, passing each file into a function named findLinesFromFile. This function reformats the data like the plotting code so that it is in the form of a matrix where the time steps form the columns of the matrix and the frequency steps are the rows.

After processing the data, LineTrack.py identifies frequency bins with values of spectral amplitude density higher than the threshold defined in lines 19-20 of the initialization file; the value in the example initialization file has the threshold set for three standard deviations above the mean. The search starts at the lowest frequency bin and the first time step. When the program finds a spectrogram tile with a value greater than the threshold, the program assumes there is a line at the tile’s frequency; the program starts to keep track of the duration of the potential line at this tile’s frequency. It then checks if the tile with the same frequency for the next time step passes the threshold. If the value is below the threshold, this step is repeated n times (defined in line 22) until a spectrogram tile above the threshold is found and the program continues checking spectrogram tiles at that frequency, or, if it does not find a spectrogram tile above the threshold at that frequency, the program checks up and down several frequency steps as defined in line 17 for the next time step to determine if the line started wandering. If it finds a spectrogram tile with a value greater than the threshold, it continues along the time steps for that frequency bin.

When LineTrack.py assumes there is a line at a particular frequency bin, it records the values of the maximum frequency, the minimum frequency, duration, and maximum spectral amplitude density as variables in the code. For the assumed line at a
frequency to be recorded, its duration needs to meet the minimum set by line 16 of the initialization file (which is a fraction of the total number of time steps in the spectrogram data). If the frequency is saved as a line, the starting frequency, maximum frequency, minimum frequency, and maximum spectral amplitude density are also recorded as part of a Python list, and the final list is used for the output. This entire line tracking process is diagrammed in Figure 3.3.

The final list of lines is written to an XML output file. Each entry in the XML table contains the name of the interferometer, channel, start time, and duration of the FScan run. The information recorded for each of the identified lines includes the maximum spectral amplitude density of the line, the starting frequency of the line, and its maximum and minimum frequency. An example of the XML file output is shown below; the output has been trimmed to show only three entries.

```xml
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE LIGO_LW SYSTEM "http://ldas-sw.ligo.caltech.edu/doc/ligolwAPI/html/ligolw_dtd.txt">
<LIGO_LW>
  <Table Name="process:table">
    <Column Type="lstring" Name="process:comment"/>
    <Column Type="ilwd:char" Name="process:process_id"/>
    <Column Type="lstring" Name="process:ifos"/>
    <Column Type="lstring" Name="process:username"/>
    <Column Type="lstring" Name="process:program"/>
    <Stream Delimiter="", Type="Local" Name="process:table">"converted from LineTrack.py output","process:process_id:0","L1","pulsar","FscanLineTrack"
        </Stream>
  </Table>
  <Table Name="sngl_burst:table">
    <Column Type="lstring" Name="sngl_burst:ifo"/>
  </Table>
</LIGO_LW>
```

26
"/
<Column Type="int_4s" Name="sngl_burst:start_time"/>
<Column Type="int_4s" Name="sngl_burst:stop_time"/>
<Column Type="lstring" Name="sngl_burst:search"/>
<Column Type="ilwd:char" Name="sngl_burst:event_id"/>
<Column Type="ilwd:char" Name="sngl_burst:process_id"/>
<Column Type="real_4" Name="sngl_burst:central_freq"/>
<Column Type="real_4" Name="sngl_burst:snr"/>
<Column Type="real_4" Name="sngl_burst:flow"/>
<Column Type="real_4" Name="sngl_burst:fhigh"/>
<Column Type="lstring" Name="sngl_burst:channel"/>
</Stream Delimiter=""," Type="Local" Name="sngl_burst:table">
  "L1",1102377616,1102464016,"FscanLineTrack","sngl_burst:event_id:0", "process:process_id:0",1.2,0.00028177952,0,1.2,"L1_PSL-PMC_MIXER_OUT_DQ",
  "L1",1102377616,1102464016,"FscanLineTrack","sngl_burst:event_id:1",
  "process:process_id:0",56.1,0.00028177952,55.7,56.1,"L1_PSL-PMC_MIXER_OUT_DQ",
  "L1",1102377616,1102464016,"FscanLineTrack","sngl_burst:event_id:2",
  "process:process_id:0",59,0.00028177952,59,59,"L1_PSL-PMC_MIXER_OUT_DQ",
...
  </Stream>
</Table>
</LIGO_LW>
Figure 3.3. LineTrack.py Diagram. This diagram shows the logical flow of LineTrack.py as it searches for lines in the spectrogram data. The program searches for a spectrogram tile above the threshold, and if it finds one, it continues along the time steps for that frequency bin. If a tile in that frequency bin drops below the threshold, it checks forward several time steps, and if it does not find a tile above the threshold, it searches adjacent frequency bins to determine if the line has wandered.
CHAPTER 4

CONCLUSION

The main goal of this thesis was to improve the FScan spectral line search pipeline in order to aid in Advanced LIGO detector characterization. Non-astrophysical noise that appears as spectral lines has a large impact on continuous wave and stochastic background searches, and its characterization is a necessity. This goal was accomplished by rewriting and improving the plotting code for FScan (Section 3.1), making improvements to the FScan driver script (Section 3.2), and creating a program for tracking wandering lines (Section 3.3). The new plotting code has been tested and found to produce output which matches with the original matlab code. The driver script has been improved to provide additional formats for presenting the FScan output as illustrated in Figure 3.1.

The wandering line tracking code, LineTrack.py, is able to successfully locate wandering lines that would otherwise be missed by the other line tracking methods of FScan. Examples of the line tracking code come from the data for a magnetometer at Livingston during ER6. Two wandering lines between $\sim$180-190 Hz appear on the spectrograms, but nothing significant appears near those frequencies for the corresponding normalized average power plots. LineTrack.py finds from the spectrogram data shown in Figure 4.1 two wandering lines: one at 183.8 Hz that wanders between 180.4-183.8 Hz and another at 190.6 Hz that wanders between 188.4-190.6 Hz. LineTrack.py finds from the spectrogram data shown in Figure 4.2 two wandering lines: one at 183.7 Hz that wanders between 180.7-183.7 Hz and another at 190.6 Hz that wanders between 188.5 and 190.6
Figure 4.1. Wandering Lines in a Spectrogram from GPS Time 1102334418. This plot shows a spectrogram with two wandering lines between \(\sim 180-190\) Hz. LineTrack.py finds two wandering lines: one at 183.8 Hz that wanders between 180.4-183.8 Hz and another at 190.6 Hz that wanders between 188.4-190.6 Hz. This plot was made with data from the channel L1-EY_MAG_VEA_FLOOR_X_DQ (a magnetometer at the end of the y-arm of the interferometer) that came from Livingston during ER6 between the GPS times of 1102334418 and 1102420817[30].

Hz. LineTrack.py finds from the spectrogram data shown in Figure 4.3 two wandering lines: one at 183.4 Hz that wanders between 181.0-183.4 Hz and another at 190.2 Hz that wanders between 188.5-190.4 Hz.

The examples used in this thesis came from data taken from the sixth science run of initial LIGO, S6, and the sixth engineering run of Advanced LIGO, ER6, leading up to its first observation run. The example results’ page in Figure 2.2 came from S6. The examples in Figures 2.1, 3.1, 3.2, 4.1, 4.2, and 4.3 come from ER6.

Further work to improve FScan may include creating a better method of measuring the significance of a line than the SNR or normalized average power; a possible new measurement of significance is the false alarm rate found in [31]. LineTrack.py currently exists as a stand-alone application to use with FScan, and it would be useful to integrate it
Figure 4.2. Wandering Lines in a Spectrogram from GPS Time 1102420817. This plot shows a spectrogram with two wandering lines between ∼180-190 Hz. LineTrack.py finds two wandering lines: one at 183.7 Hz that wanders between 180.7-183.7 Hz and another at 190.6 Hz that wanders between 188.5-190.6 Hz. This plot was made with data from the channel L1-EY_MAG_VEA_FLOOR_X_DQ that came from Livingston during ER6 between the GPS times of 1102420817 and 1102507217[30].

fully with the FScan infrastructure. Other possible future developments could focus on rewriting other portions of the FScan pipeline in Python, for example, the code to make the SFTs or perform the spectral averages.
Figure 4.3. Wandering Lines in a Spectrogram from GPS Time 1102507217. This plot shows a spectrogram with two wandering lines between \( \sim 180-190 \) Hz. LineTrack.py finds two wandering lines: one at 183.4 Hz that wanders between 181.0-183.4 Hz and another at 190.2 Hz that wanders between 188.5-190.4 Hz. This plot was made with data from the channel L1-EY_MAG_VEAFLOOR_X_DQ that came from Livingston during ER6 between the GPS times of 1102507217 and 1102593617[30].
BIBLIOGRAPHY


Cody Arceneaux was born in 1988 in Slidell, LA. He participated in a research experience for undergraduates at the LIGO site in Hanford, WA, in the summer of 2009. He received a Bachelor of Science degree in Physics from Louisiana State University in 2010.