Draft version September 6, 2022
Final Report for Caltech LIGO SURF 2022

GW strain calibration of LIGO detectors with high precision and low latency

Naomi Shechter,[1]
Mentors: Ethan Payne,[2] and Dr. Alan Weinstein[2]

[1] *Department of Physics and Astrophysics, DePaul University*
[2] *Caltech LIGO*

ABSTRACT

The detection of gravitational waves (GW) has opened a new era of astrophysical observation, allowing scientists to view and analyze previously unseen phenomena. This process hinges upon measuring the strain $h = \Delta L/L$ of space over $L = 4km$ long baselines, which change by a differential arm (DARM) length on the order of $\Delta L = 10^{-20}$ meters when a GW passes. From this information-rich time series, a wealth of astrophysical information may be deduced. In order to produce a reliable estimate of the strain, the Laser Interferometer Gravitational Wave Observatory (LIGO) detectors must be precisely calibrated. Furthermore, the calibration pipeline must produce an associated calibration uncertainty estimate with which to characterize the strain. While uncertainty estimates can currently be produced with low latency, it takes months to investigate the sources of error and verify the quality of calibration. Producing a high precision, low latency uncertainty estimate and a diagnostic monitoring software is therefore crucial for LIGO's fourth observing run (O4), scheduled to begin in March 2023. It is this task that forms the basis of this research project. The uncertainty estimation and monitoring software are to be produced using pyDARM, a python package which implements the DARM control loop model and is currently under development. The software must reliably output uncertainty estimates and a suite of diagnostic plots on the timescale of an hour. It will actively be of use in O4.

I. PRINCIPLES OF OPERATION AND CALIBRATION FOR THE LASER INTERFEROMETER GRAVITATIONAL WAVE OBSERVATORY

1.1 - Operation of Laser Interferometer Gravitational Wave Observatory

The Laser Interferometer Gravitational Wave Observatory (LIGO) at Hanford and Livingston is among the most sensitive detectors ever built. This makes it all the more remarkable that LIGO functions similarly to the archetypal Michelson Interferometer. Due to this high sensitivity, the detector must be well calibrated to produce meaningful, unbiased results. However, calibration process cannot be adequately described without first explaining how LIGO operates. Laser light enters the detector and encounters the beam splitter, which sends it into two perpendicular, 4 km long arms (Fig.1). Each arm includes a Fabry-Perot cavity. The first mirror in the cavity is 99% reflective, so only 1% of the incident light enters the cavity. It builds in power significantly due to the nearly 100% reflective test mass at the other end of the arm. However, rather than produce a pattern of constructive interference after the split laser beams of the interferometer recombine, the light reflected by the arm cavities largely cancels the newly incident light emitted by the laser. In other words, LIGO is an active null instrument that relies on destructive interference to maintain a baseline output from which deviations may be measured. When a gravitational wave (GW) passes, a transverse strain $\Delta L/L$ is induced in arms, which causes a minuscule differential arm (DARM) length change on the order of 10^{-20} meters. This causes a small phase shift in the light recombining at the beamsplitter, and thus a small amount of light exits the beamsplitter in the direction of the readout port.

1.2 - DARM Control Pipeline and Calibration Pipeline

A feedback control loop constantly attempts to maintain a null position of the interferometer. The actual change of the differential length in the arms, ΔL_{free}, is measured in the presence of the feedback control loop which actively suppresses it through actuation. The residual DARM length ΔL_{res} is passed to the sensing function, which produces the error signal d_{err}. This is the input for the DARM control pipeline's digital filters and actuation function, which

help maintain the nominal positions of the test masses and resonance in the arm cavities (Fig.2). The d_{err} signal is also fed into the calibration pipeline, which uses a detailed reconstruction of the $1/C$ and A models to produce the gravitational wave strain. It does this by comparing the model to the measured response over different time epochs [3]. A new epoch begins and new models are made whenever significant changes are made to the detector. From the resultant response function and the outputs of various statistical algorithms, one may characterize the complete detector response along with the calibrated strain time series $h(t)$. The strain is comprised of GW signals, displacement noise and sensing noise. To more fully understand the quality of $h(t)$, we also produce the strain uncertainty estimate, which is dependent solely on GW information.

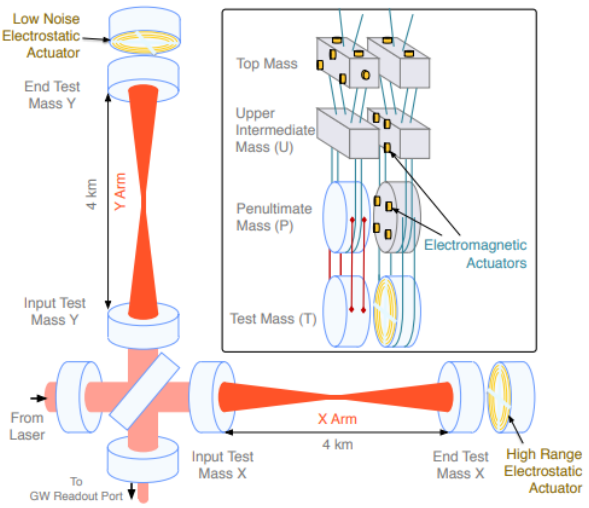

Figure 1. A simplified figure of the LIGO detector [1]. Laser light enters from the left and encounters the beam-splitter. The test masses (inset) are constantly actuated to keep the detector on resonance and nearly complete destructive interference at the GW Readout Port (bottom left). The differential arm length is detected from the readout port as a change in intensity.

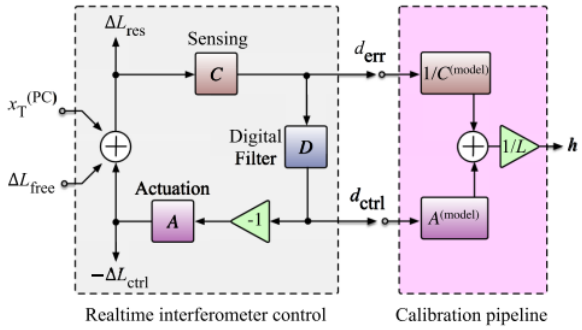

Figure 2. The LIGO control loop and the LIGO calibration pipeline [1]. The sensing block C measures the residual change ΔL_{res} in length as the error signal d_{err} which the digital filter block D converts into d_{ctrl}. The control signal is then fed to actuation block A to actuate the test masses by the control length ΔL_{ctrl}. This ensures that the mirrors remain in their nominal positions. The calibration pipeline passes these same signals through filters $1/C^{model}$ and A^{model} which are fit by MCMC and GPR methods that compare the measurements to a model. The output of the calibration pipeline is the reconstructed strain h(t) and the uncertainty estimate of the response function (see Equations 1-3 below).

$$h = \frac{\Delta L_{free}}{L} = \frac{1}{L} * \frac{1}{C^{model}} d_{err} + A^{model} d_{ctrl} = \frac{1}{L} * R^{model} d_{err} \tag{1}$$

where,

$$R^{model} = \frac{1 + A^{model} C^{model} D}{C^{model}} \tag{2}$$

and the uncertainty in the response function (and thus the certainty with which we can reconstruct the strain) is thus equivalent to,

$$\frac{h^{true}}{h^{model}} = \frac{R^{true}}{R^{model}} \tag{3}$$

1.3 - The Importance of Precision Low Latency Calibration and Uncertainty Estimates

Calibration is a crucial process to ensure that the aforementioned pipeline provides accurate data, allowing all downstream data analysis to be completed with the utmost confidence. Given how sensitive LIGO must be to measure GW strain, it must also be as accurate as possible. Currently, a code called pyDARM is being utilized to model the DARM control loop and strain uncertainty estimation in a more accessible manner than has been done in O3 [2]. Testing and supplementing pyDARM in order to develop a reliable low latency calibration uncertainty estimate for O4 forms the basis of my research.

As of the third observing run (O3) which lasted from April 2019 to March 2020, our calibration uncertainty within the sensitive band of 20-2000 Hz was known to $\sim 2\%$ in magnitude and $\sim 2^\circ$ in phase [1]. An uncertainty estimation graph illustrating this is given in Figure 3. There are several reasons why we continue to seek higher precision, although

all are motivated by the prevention of biased astrophysical results [3 - 4 - 5 - 6 - 7 - 8]. Concerning tests of general relativity in the strong regime, only evidence collected with extremely reliable calibration would be capable of shedding doubt on the theory. For future observation runs such as O5, we expect an increase in sensitivity so significant that uncertainty must be lowered to 1% levels in order to reliably reconstruct signals of SNR greater than 100. Additionally, calibration uncertainty directly affects the uncertainty in the estimation of parameters such as binary masses and spins, and sky location and luminosity distance; all important for both GW-based and multi-messenger astronomy[1]. To summarize, many data analysis pipelines and research projects rely on high quality detector calibration. This makes it not only a crucial piece of LIGO's intricate construction, but one which must be well-understood, precise and efficient to produce reliable data.

Understandably, sub-optimal calibration introduces many issues; verifying calibration during O3 was a particularly time-intensive process [3]. The $\sim$ 2% level of uncertainty was only released after months of verifying the initial uncertainty estimate which was earlier estimated to be of order 5%. Furthermore, by the time the calibration model C01 was completed, much of the parameter estimation had already occurred with the less precise C00 model (which actually lacked a reliable uncertainty estimate). In preparation for O4, the calibration group is working to expedite the process to verify the low latency strain uncertainty estimate. My research project will contribute to the realization of this goal.

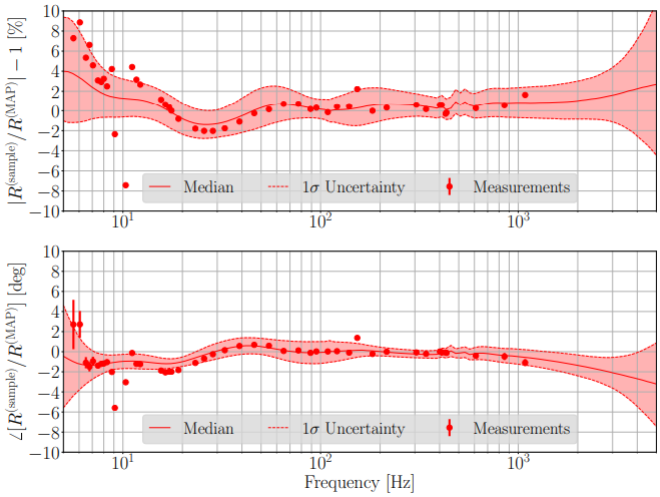

Figure 3. A graph of the combined error and uncertainty estimate from O3a at the Hanford detector[1]. The data points are a variety of residuals between the modeled response function and the samples of the response function $R^m(f)$, compiled from many measurements. The solid line is the median of this relative response. It is the best estimate for the frequency dependent systematic error. The light red envelope outlines the $+1\sigma$ and -1σ boundaries on the uncertainty. While both the uncertainty and the standard deviation are large above 10 Hz, they even out significantly around 20 Hz. In the interval from 20 Hz to 1000 Hz, they barely pass positive or negative 2% in magnitude or $\sim 2^\circ$ in phase.

II. APPROACH AND OUTLINE FOR THE RESEARCH PROCESS

In order to expedite the process of generating calibrated strain uncertainty with high precision and low latency, I wrote a series of Python files which utilized already extant data from O3. I also produced a series of preliminary diagnostic plots which track the time dependent correction factors (TDCFs) [1]. Further work may include a monitoring program which raises a warning if the uncertainty falls outside satisfactory limits, a graph of the average frequency dependent systematic error over time, and other such diagnostic tools. The majority of the Python files ran reliably, without crashing, and with a latency of less than an hour. However, there is still much room to improve how efficiently the TDCFs and time-varying uncertainty are calculated.

Developing a thorough understanding of pyDARM was necessary in order to produce a program which accurately characterizes uncertainty. Therefore, I spent the first week or so of this summer stepping through a Jupyter notebook by Evan Goetz and colleagues, which contains various demonstrations on sensing and actuation function models, as well as Markov Chain Monte Carlo (MCMC) and Gaussian Process Regression (GPR) algorithms [2]. My understanding of

the algorithms has been supplemented with several papers [9 - 10]. In the process of producing plots and encountering errors, I have become more familiar with the various pyDARM classes. I have also done some introductory learning about control systems and null instruments to more fully understand the LIGO control pipeline [11].

After acquiring this background information, I constructed several pyDARM-compatible model files using older files from O3. New models of the sensing, actuation and digital filter transfer functions are produced after significant changes to the detector, and demarcate periods called epochs. Once these were complete, I began calculating the uncertainty via the previously described MCMC and GPR methods. Once these two forms of analysis were completed, a final calibrated strain uncertainty estimate was generated. It is informed by all previous steps. Finally, a graph of various TDCFs was produced. It sampled twice a day for a week surrounding the date of measurement to examine how quantities like the detector bandwidth and actuation stage-specific gains vary over time. While there were a couple goals which did not come to fruition this summer, such as producing a full suite of diagnostic plots or constructing a program which raises a warning should the uncertainty be above a certain threshold, the work which I have completed forms a solid basis for a new tool to be employed during O4.

III. RESEARCH PROCESS

Most of my first week and a half was spent reading papers, exploring the pyDARM code and building the background knowledge necessary to carry out my research. In this section, I will outline my understanding and share plots which are illustrative of the various subjects I have been introduced to thus far.

3.1 - Linear Time Invariant Systems, Control Systems and Null Instruments

Before interacting with pyDARM, I sought out resources about linear time-invariant (LTI) systems, control systems and null instruments [11]. Consider a mass on a spring, to which you provide an impulse by hitting it suddenly. The function for driven dampened simple harmonic oscillators defines the impulse response, and the result would be a decaying sinusoid as the oscillating mass eventually comes to rest. If you model a more complicated input signal as a sum of impulses, the output of an LTI system will be the linear superposition of the impulse response from each individual impulse. In mathematical terms, the output is the convolution of the impulse response with the input. For LTI systems, convolution becomes multiplication after performing a Fourier transform, so it is simpler to work in frequency domain. Therefore, rather than the impulse response determining the output of the system, we use the Fourier transform of the impulse response: the transfer function. In the context of LIGO, rather an an impulse, the excitation signal is a swept sine stepping through a broad range of discrete frequencies, across the sensitive band of 20-2000 Hz.

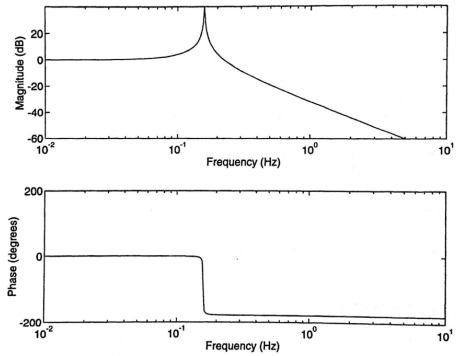

Figure 4. Bode plot of a mechanical oscillator from Fundamentals of Interferometric Gravitational Wave Detectors by Peter R. Saulson. The large peak indicates the resonant frequency, at which the phase passes through 90° [11].

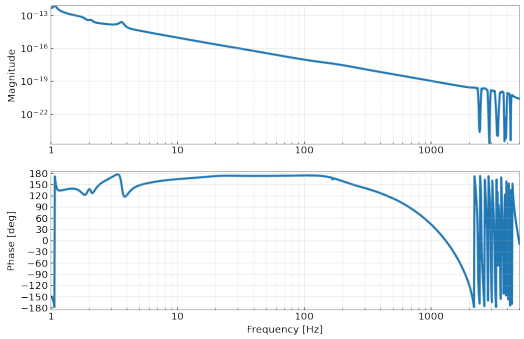

Figure 5. The complete actuation transfer function for the LIGO Hanford calibration model produced on April 16th 2019. The decreasing slope of the upper magnitude graph is similar to the decreasing slope which occurs in Figure 4 at frequencies higher than resonance.

The transfer function of a driven damped simple harmonic oscillator, such as a pendulum, is plotted over a range of frequencies in Figure 4. The large spike in the magnitude of oscillation is resonance, at which the driving frequency equals the natural frequency of the oscillator. As the two frequencies become completely out of phase, the magnitude

becomes negligible. This means that there is a pole located at resonance, or that (should we act this transfer function upon a signal), frequencies around resonance would be amplified. There are also zeroes, at which the signal goes to zero and nearby frequencies are attenuated. In other words, poles and zeroes are a very useful method to quantify transfer functions, and allow us to construct filters. For example, the actuation transfer function behaves in such a way as to reduce noise from the movement of the test masses (Fig.5). The upper intermediate mass and penultimate mass each contribute to the overall transfer function, and much like in Figure 4, they attenuate high frequencies. Effectively, the entire transfer function works as a digital low pass filter.

3.2 - Markov Chain Monte Carlo and Gaussian Process Regression

After I produced several plots of the the transfer functions in pyDARM, I moved on to understanding the way in which we compare and fit the measured data to a parametrized model of the detector response. The first of these which I encountered was the Markov Chain Monte Carlo (MCMC) method [9]. Given measured data and a parametrized model, we produce a prior distribution and a likelihood function of the model values and measured data. Then, MCMC generates a distribution of functions dependent on the model parameters, from which the maximum a posteriori (MAP) values are calculated. These MAP parameters correspond to the model which has the highest posterior probability, and they are used to construct the best fit curve. In the case of this project, I performed MCMC analysis on many individual measurements within each epoch. The file was set up such that a list of models and measurements could be provided as command line arguments, and each measurement object would be initialized with the corresponding model for the epoch it is in. However, as demonstrated on the left in Figure 6, fitting with the MCMC algorithm does not completely account for systematic error and does not adequately fit the data by itself. The residual systematic errors are handled by Gaussian Process Regression (GPR) as seen on the right half of Figure 6.

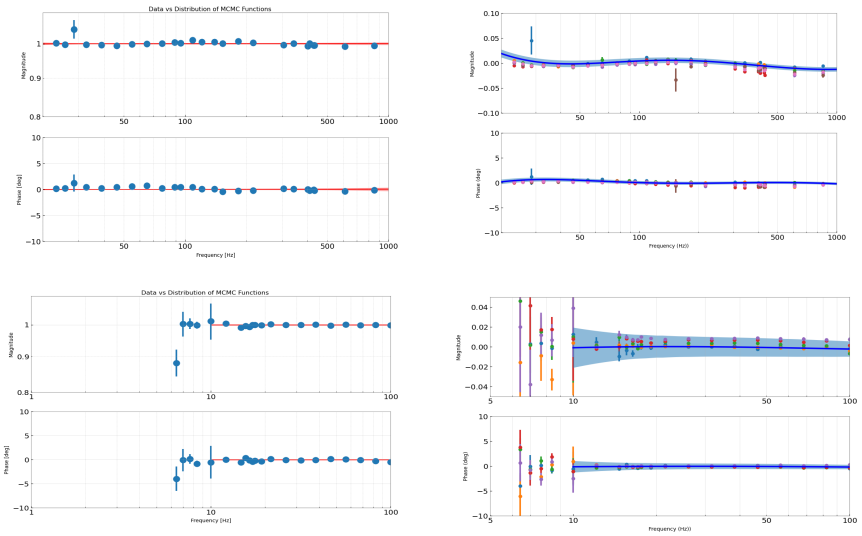

Figure 6. Several graphs of the MCMC (left) and GPR (right) analyses for January 3rd 2020. These are Bode Plots, with magnitude versus frequency on top and phase versus frequency on the bottom. The first row is for the sensing function while the second row is for the actuation function's test mass (TST) pendulum stage. In the MCMC graphs, the red lines are 100 possible functions which characterize the residual difference between the modeled sensing/actuation function and the one produced by measurement data. The actuation function only runs from around 8 Hz to 100 Hz due to the fact that the actuation function is only important for the overall response at low frequencies; at higher frequencies, the sensing function dominates. The variation of the residual data points from the various distributions is not well described by these graphs alone. The GPR more fully characterizes the uncertainty. The solid blue line in these graphs is the average magnitude of the residuals, while the envelope represents the $+1$ and -1σ boundaries. A number of data points are graphed in order to illustrate a few of the many MCMC chain results which are used to generate the envelope. They diverge below 10 Hz for the actuation function due to the presence of seismic noise which is too powerful to be resolved through actuation. Thus, the actuation function is difficult to measure precisely

The GPR code takes the MAP parameter results of all measurements for a given epoch as input. It stacks these measurements, and then produces a probability distribution of many possible functions, each of which describes a model response function $R(f)$ consistent with the measurements. The median of those functions is interpreted as the systematic error, and the spread in those functions is an estimate of the uncertainty in the response [10]. As more measurements are provided, the uncertainty envelope becomes more narrow. Compared to MCMC analysis, GPR more fully captures the uncertainty between the model and measurements. This uncertainty arises both from the failure of the (relatively simple) parameterized model to accurately describe the detector response, and statistical errors in measurement. Additionally, due to the extrapolation and interpolation between points, GPR is the dominant contributor to the final uncertainty estimate. The final uncertainty estimate is on R/R_{model}, or the measured response divided by the parametrized model of the response function, as produced by a combination of the sensing, actuation and digital filter functions.

3.3 - Uncertainty Generation and Preliminary Monitoring Plots

Once the full MCMC chain and the GPR posteriors were saved, I was able to create uncertainty configuration files for each measurement (Fig. 7). These configuration files contained the results of the MCMC and GPR analyses, as well as a list of channels corresponding to the time dependent correction factors (TDCFs) sampled for the final uncertainty. The TDCFs I included were $\kappa_c(t)$, $\kappa_{tst}(t)$, $\kappa_{pum}(t)$ and $f_{cc}(t)$ which are the scalar gain factor of the sensing function and the actuation functions of the lowest two pendulum stages, and the coupled cavity pole frequency respectively. They were sampled every twelve hours for a week surrounding the date of measurement (Fig. 8). The UIM pendulum stage was producing unexpected results in GPR, to be investigated, and therefore could not be included in the final uncertainty generation (Fig. 9). However, the UIM is also the least impactful of the three lowest pendulum stages upon the final uncertainty estimate, with the majority of uncertainty it causes being below 15 Hz. Therefore, the current results should be sufficiently reliable.

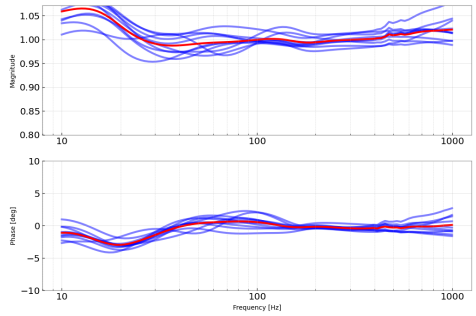

Figure 7. An overall uncertainty estimate graph for January 3rd 2020, in the sensitive region from 10-1000 Hz. The blue lines are several possible distributions which characterize the uncertainty, with limits of $+1\sigma$ and -1σ. They diverge around 1000 Hz because the photon calibrators (Pcal) used to provide reference measurements have a low signal to noise ratio, as actuation is less effective at high frequencies. Therefore, noise dominates. Furthermore, the red line is interpreted as the systematic error. The ideal average systematic error would be a straight line at 1 in magnitude and 0° in phase from 50 Hz onward, but the detector is not yet modeled to such precision. However, this graph is very close to ideal considering our current limitations.

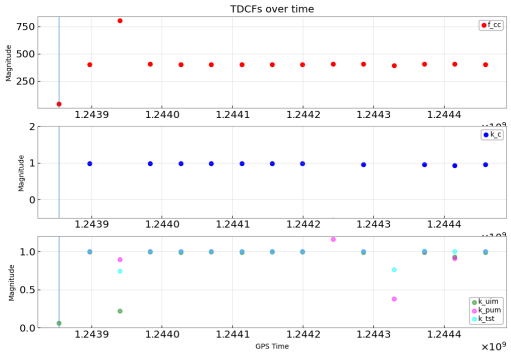

Figure 8. A graph of the TDCFs over time from June 6th 2019. This date is particularly illustrative of both expected variation and unexpected detector behavior. For example, the gain of the actuation stages is expected to vary with time as charge builds up on the electrostatic actuators, reducing their effectiveness. However, the anomaly at the third GPS time in $f_{cc}(t)$ and $\kappa_c(t)$ does not have as simple of an explanation. This is only one of many diagnostic plots which may be produced to provide a closer look at the strain calibration uncertainty.

IV. FUTURE WORK

There is much more work to be done before my summer project can be implemented in O4. While it is both reliable and robust for the majority of the process, the code which generates the uncertainty estimate and monitoring data takes an unreasonable amount of time. There are several ways this may be amended.

The first among these is splitting one large job into many smaller jobs. The approach I took involved three input files. Though the exact contents of these files is discussed further in the Appendix, they may be described as a list of models, a list of measurement files, and a list of GPS times. The uncertainty code would run through each epoch, viewing a collection of GPS times for each measurement. To speed up this process, one may choose to run each epoch separately, or even to run each measurement by itself. This would greatly reduce the amount time the uncertainty and TDCF computation requires. Another option may be to change what is returned when one runs the pyDARM function *compute_response_uncertainty*. This slow-moving function takes a GPS time, with which it computes the TDCFs in order to obtain the nominal response of the detector at that time. However, it does not return the TDCFs. This means that in order to find the TDCFs, the uncertainty code currently calls two functions which each take a decent amount of time. If *compute_response_uncertainty* could be edited to return the TDCF's as well (for example, if a certain keyword is input as True) then the overall time to produce the uncertainty and TDCF data would be reduced. Finally, the code to produce the TDCF data could be separated into its own file, depending on how necessary plots of the overall uncertainty estimate are over time versus solely on the day of the measurement. Any of these options would help make the code more efficient and reduce run time.

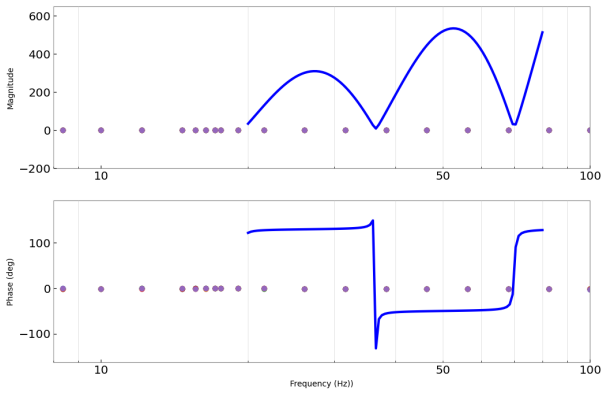

Figure 9. The unexpected graph of the UIM actuation transfer function, discussed in the following paragraph. It has uncharacteristic poles and zeroes at 36 Hz and 70 Hz. Normally, it should be a straight line somewhat matching up to the purple data points, with an envelope similar to that in Figure 6. The data plotted is the same data provided to the *run_gpr* function, and has been confirmed to be correct.

Other than that, two goals remain. The first is to produce a suite of other diagnostic plots, which may include the average systematic error over time, and the error present at a broad selection of representative frequencies over time. If possible, this monitoring program would be equipped with the capability to raise a warning if the uncertainty is beyond a certain threshold, though there was not sufficient time to pursue this over the summer. The second main goal for future work is to uncover why the UIM actuation stage was producing odd results for GPR despite the inputs to the *run_gpr* function being verified as correct. In order to take the full detector response into account when generating the uncertainty, this must be resolved before O4 in March 2023.

V. CONCLUSION

During the LIGO SURF program in the summer of 2022, I worked with the pyDARM package to produce several files which precisely characterize the calibrated strain uncertainty estimate and produce informative plots. It was not only a lesson in detector calibration, but in control systems, coding efficiently, git and many other useful tools and practices. While there is work yet to be done to prepare the code I've written for O4, my project forms a solid foundation to be optimized for the circumstances and requirements of the run. As I enter my senior year of college,

I can confidently say that the LIGO SURF program has helped me better understand my professional goals, and has provided me with a newfound sense of self-efficacy.

VI. ACKNOWLEDGMENT

This report was funded by the National Science Foundation (NSF) and completed as part of the LIGO SURF Program at Caltech, in contribution to the LIGO Scientific Collaboration. Many thanks to my mentors Dr. Alan Weinstein and Ethan Payne.

REFERENCES

[1] Sun, L., Goetz, E., Kissel, J. S., et al. "Characterization of systematic error in Advanced LIGO calibration." Classical and Quantum Gravity, 37, 225008. doi:10.1088/1361-6382/abb14e (2020).

[2] pyDARM Gitlab Repo of the Laser Gravitational Wave Interferometer Calibration Group, https://git.ligo.org/Calibration/pydarm.

[3] Conversations with Mentors concerning background knowledge and the calibration process, during the first week of the LIGO SURF Program.

[4] Payne, E., Talbot, C., Lasky, P. D., et al. "Gravitational-wave astronomy with a physical calibration model." PhRvD, 102, 122004. doi:10.1103/PhysRevD.102.122004 (2020).

[5] Vitale, S., Del Pozzo, W., Li, T. G. F., et al. "Effect of calibration errors on Bayesian parameter estimation for gravitational wave signals from inspiral binary systems in the advanced detectors era." PhRvD, 85, 064034. doi:10.1103/PhysRevD.85.064034 (2012).

[6] Vitale, S., Haster, C.-J., Sun, L., et al. "physiCal: A physical approach to the marginalization of LIGO calibration uncertainties." PhRvD, 103, 063016. doi:10.1103/PhysRevD.103.063016 (2021).

[7] Essick, R. & Holz, D. E. "Calibrating gravitational-wave detectors with GW170817." Classical and Quantum Gravity, 36, 125002. doi:10.1088/1361-6382/ab2142 (2019).

[8] Huang Y., Chen H.-Y., Haster C.-J., Sun L., Vitale S., Kissel J. "Impact of calibration uncertainties on Hubble constant measurements from gravitational-wave sources." arXiv:2204.03614 (2022).

[9] Cahillane C. et al. "Calibration uncertainty for Advanced LIGO's first and second observing runs." Physical Review D 96.10 102001. (2017).

[10] Wang, J. "An intuitive tutorial to Gaussian processes regression." arXiv:2009.10862 (2020).

[11] Saulson, Peter R. "Fundamentals of Interferometric Gravitational Wave Detectors". Singapore: World Scientific (2017).

APPENDIX

A. CODE DOCUMENTATION

Here is the documentation for the several files I used to produce the final calibrated strain uncertainty estimate. Everything should work for both detectors and their respective favoring of the X- or Y- arms, although I only interacted with Hanford data to reduce the amount of .ini model files and .ini uncertainty configuration files which had to be manually set up. Additionally, only the actuation files are labelled with their respective function. For example, you have an *mcmc_tst_wip.py* file to run MCMC analysis on the the TST stage, but the sensing function's MCMC file is called *mcmc_wip.py*, not *mcmc_sensing_wip.py*. Furthermore, all Python files are structured to be as flexible as possible. They may analyze individual measurements and models, a single epochs worth of data, or multiple epochs worth of data.

A.1. ***Directory Structure***

This section of the Appendix will list what is in the various directories my code creates to organize the numerous graphs and data files. My materials can be found under /home/naomi.shechter/projects/pydarm/ on the LIGO CIT computing cluster, or at https://git.ligo.org/naomi.shechter/pydarm. Any file which must be manually created is marked with two asterisks.

1. ** model_files Directory. This is what I called mine, but it can be called whatever you like. However, you will have to go through and replace all instances of model_files in filepaths with your chosen directory name. This was one early summer organizational mistake which I did not have time to go back and fix.
   (a) ** Model configuration .ini files, one per epoch.
   (b) ** A .txt file where each line is a path to one of the above listed configuration files.
   (c) ** querylist.txt (explained in Appendix A.2).
   (d) All MCMC/GPR files, the uncertainty file (*unc_wip.py*) and the plot generation file (*plotmaker.py*).
   (e) ** Directory called 'unc' where all the uncertainty configuration files are stored.
   (f) Directory where all the results are further organized. The MCMC file allows you to specify the name of this directory. Mine is called makeorganize.
2. makeorganize
   (a) One directory (folder) per epoch date.
3. Within each of those epoch folders...
   (a) GPR graphs and results.
   (b) querytime.txt (the files which are listed in querylist.txt, further explained in Appendix A.2.)
   (c) One folder per measurement or uncertainty date.
4. Within each of those measurement/uncertainty date folders...
   (a) MCMC plots and results.
   (b) Uncertainty plots and results.
   (c) monitoring folder.
   (d) samples folder.
5. Monitoring plots are in the 'monitoring' folder.
6. The TDCFs and Uncertainty data for every GPS time is in the 'samples' folder.

A.2. *Inputs and Outputs*

The inputs to MCMC and GPR are two .txt files.

1. The first file contains N .ini file paths (one per line) with N being the number of epochs. Each .ini file is a pyDARM compatible model file. The model files do not need to be named in any specific way.
2. The second file contains N .txt file paths (one per line).
   (a) Furthermore, each .txt file contains several measurement config file paths (one per line).
   (b) The measurement files are listed in pairs (ie: line one is the DARM_OLGTF file, line two is the PCSL2DARMTF file and so on). The basepath is year-month-day followed by the file specifications given in the previous sentence.
3. Outputs:
   (a) MCMC: For all 3 stages and for the sensing fuction, there is a corner plot, graph of residuals with several MCMC chains over top (distro_mcmc), a plot of the residuals alone (hresid) and a plot of the residuals with the non-normalized model underneath (residover). Also outputs and saves the MCMC chain as a .json file and .hdf5 file.
   (b) GPR: A graph of the GPR envelope and residuals for all 3 actuation stages and the sensing function, as well as the .hdf5 file of the GPR data called gprresults or stage_[gprresults].

The code which produces the uncertainty estimate and TDCFs takes the same model .txt file as MCMC and GPR. Additionally, it takes...

1. A .txt file of N .txt file paths (one per line) where each of these .txt files contains all of the uncertainty configuration .ini files for a given epoch (again, one path per line).
   (a) Due to the volume of unc .ini files, I made a separate folder called unc within the model_files directory and listed them all out there.
   (b) The uncertainty configuration files are .ini files, one for each measurement. There's probably an easier way to grab the date of the uncertainty configuration files, but I just ended up naming them in the format H1_unc_year-month-day.ini and then indexing the string to get the date. Therefore, uncertainty configuration files need that format.
2. A.txt file with the start and end GPS times of each epoch (one GPS time per line).
3. Outputs:
   (a) Samples of the response function uncertainty for a week in GPS time after the uncertainty configuration date, taken every 12 hours, named [GPS time].txt.
   (b) TDCFs calculated at those same times, in files named tdcfs_[GPS time].txt.
   (c) querytimes.txt. One for each epoch. This file contains N lines, where N is the number of measurements. Each line has a list of the GPS times at which that measurement was sampled.
   (d) A graph of the combined uncertainty per each unc configuration file, called unc_check.txt.

The monitoring takes the previously mentioned .txt files with model and uncertainty configuration file paths. It also takes...

1. A file with N lines, N being the number of epochs, where each line is a path to that epochs querytimes.txt.
2. Ouputs:
   (a) A plot of the TDCFs over time.

A.3. *MCMC*

These files are called *mcmc_wip.py* or *mcmc_[stage]_wip.py*. With 10 burn-in-steps, the MCMC algorithm wakes 5 minutes to run.

1. Takes model and measurement files described above.
2. Python files exist for sensing and all 3 actuation stages.
3. The steps and burn in steps for the MCMC chain are set lower than ideal for testing. Please increase these to your desired length, or remove and leave as default.

A.4. *GPR*

These files are called *gpr_wip.py* or *gpr_[stage]_wip.py*. This takes around a minute to run.

1. Takes same model and measurement text files as the MCMC.
2. Measurements may not be plotted on the GPR graphs correctly, didn't have time to look into this.

A.5. *Uncertainty*

This file is called *unc_wip.py*. This is the file which needs the most changes to run in a reasonable amount of time. One epoch of 12 measurements takes several hours. See Section IV for a more thorough discussion of ways this problem may be solved.

1. Takes same model, measurements file as MCMC/GPR, and GPS time file described in the inputs section.
2. Samples response uncertainty and TDCFs twice a day for a week surrounding the measurement.
3. No merge has happened to update currently extant issues in uncertainty.py at the time of writing, so preliminary edits had to be made to get things to run. See merge request !132.
4. The GPS times used must be adjusted in order for the code to calculate TDCFs past the date of the final uncertainty configuration file. For example, if you want to sample for three days after the final measurement, you must add more GPS times to the array of times to be sampled. See line 112 in *unc_wip.py* for more information.
5. UIM is not included due to odd behavior in GPR.

A.6. *Monitoring Plots*

This file is called *plotmaker.py*. It takes one or two minutes to run.

1. Reads in from text files since $save\_to\_file$ in *compute_response_uncertainty* does not yet work for hdf5.

A.7. *Example Command Line Inputs*

Here is an example of a sequence of command line instructions, with arguments. This series of commands would give you results for the sensing function.

1. python mcmc_wip.py -mod H1_modeltext.txt -mymeas H1_allep_meas.txt
2. python gpr_wip.py -mod H1_modeltext.txt -mymeas H1_allep_meas.txt
3. python unc_wip.py -mod H1_modeltext.txt -myunc H1_allunc.txt -gps H1_allep_gpstimes.txt
4. python plotmaker.py -mod H1_modeltext.txt -myunc H1_allunc.txt -query querytimes.txt