

# BayesWave Optimization Report

Tyson Littenberg,\* Neil Cornish, Jonah Kanner, and Francesco Pannarale

---

\* [tyson.littenberg@northwestern.edu](mailto:tyson.littenberg@northwestern.edu)

## I. OVERVIEW

**BayesWave** is a Bayesian parameter estimation and model selection code designed to follow-up candidate events from Burst search pipelines. The outputs of the algorithm are relative evidence estimates for each of three models for the data: (i) Gaussian noise, (ii) Gaussian and transient noise “glitches”, or (iii) Gaussian noise and a gravitational wave signal coherent across the detector network. The evidence is a large multidimensional integral that must be numerically calculated with a Markov Chain Monte Carlo algorithm. Because of the dimension of the models,  $\mathcal{O}(10^7)$  points are needed for the numerical integration. Each sample in the MCMC is relatively cheap to compute – the **BayesWave** model is inconsequential compared to, say post-Newtonian templates used in CBC searches – but there is no getting around the large number of trials needed in the Monte Carlo integration. Before computing evidence for each model, **BayesWave** uses a combination of the glitch model and the **BayesLine** algorithm to estimate the noise power spectral density used in the Gaussian noise model. The two algorithms are described in great detail in Refs [1, 2].

For GW signals **BayesWave** provides sky maps, reconstructed waveforms, and posterior distribution functions for waveform characteristics such as central frequency, bandwidth, duration, signal energy, etc. The algorithm also provides reconstructed waveforms and parameter estimates for glitches which can be used in detector characterization studies **The goal is to enable high-confidence detections by rejecting glitches, and to infer properties of the gravitational wave signals.**

**BayesWave** uses a linear combination of sine-Gaussian wavelets as the model for glitches and signals. A Reversible Jump Markov Chain Monte Carlo (RJMCMC) algorithm marginalizes over the number of basis functions used in the linear combination. To compute the evidence we use thermodynamic integration which requires several RJMCMCs to be run simultaneously, each sampling from a different likelihood function. In studies using archived data, we have seen that the hierarchical pipeline of **coherent WaveBurst** combined with follow-up analysis by **BayesWave** gives improved separation of signals and glitches, compared with **coherent WaveBurst** alone. See <https://wiki.ligo.org/Bursts/BayesWave/BwbImbh> for details.

**BayesWave** investigates **coherent WaveBurst** triggers using only the GPS time of an event, either zero-lag or time slides. For each GPS time **BayesWave** analyses 4 seconds of data with a sampling frequency of 4096 Hz. The (log) Bayes factors (evidence ratios) between the signal and glitch model and signal to noise model are used to determine the significance of a candidate event. We analyze **coherent WaveBurst** triggers from time slides to estimate the background for the signal to glitch Bayes factors.

## II. COMPUTATIONAL COST ESTIMATE

### A. Computing Cost “First Principles” Estimate

In order to compute the evidence for each model, **BayesWave** performs an RJMCMC with  $2 \times 10^6$  steps for each of 20 chains. At each step, the main cost in the algorithm is computing the glitch/signal waveform and the likelihood that the model fits the data, which involves and integral in the Fourier domain over the bandwidth of waveform. Fourier transforms of sine-Gaussians are computed analytically so there are no calls to FFTs. Typical bandwidths are  $\sim 4 \times 10^3$  Fourier samples (i.e. frequency bins). A careful walkthrough of the code shows that each likelihood evaluation requires 60 FLOPs per bin. We can then estimate the total cost of running 1 trigger as:

$$2 \times 10^6 \text{ bins/chain/model} \times 20 \text{ chains} \times 60 \text{ FLOPs/bin} \times 4 \times 10^3 \text{ bins} \times 4 \text{ models} = 4 \times 10^{13} \text{ FLOPs}$$

This simple estimate is within a factor of 4 of the speeds we see when benchmarking the code. It does not include once-per-iteration computations that may have significant cost, including proposal distributions for extrinsic parameters, the LIGO response functions for the GW model (involving several trigonometric functions), additional calls to the waveform function to compute co-variance matrices, and matrix inversions to determine eigenvalues and eigenvectors of said matrices.

### B. Benchmarking

**BayesWave** has been tested on dedicated benchmarking cores at CIT. For our test suite we have used the 32 most significant **coherent WaveBurst** background events from the L1H1 network during S6D using our current default settings. Median run-times are  $\sim 24$  hours per trigger on a single core. Figure 1 shows a histograms of CPU time in hours on the dedicated cores for the 32 background events. Additional background events and injections will be

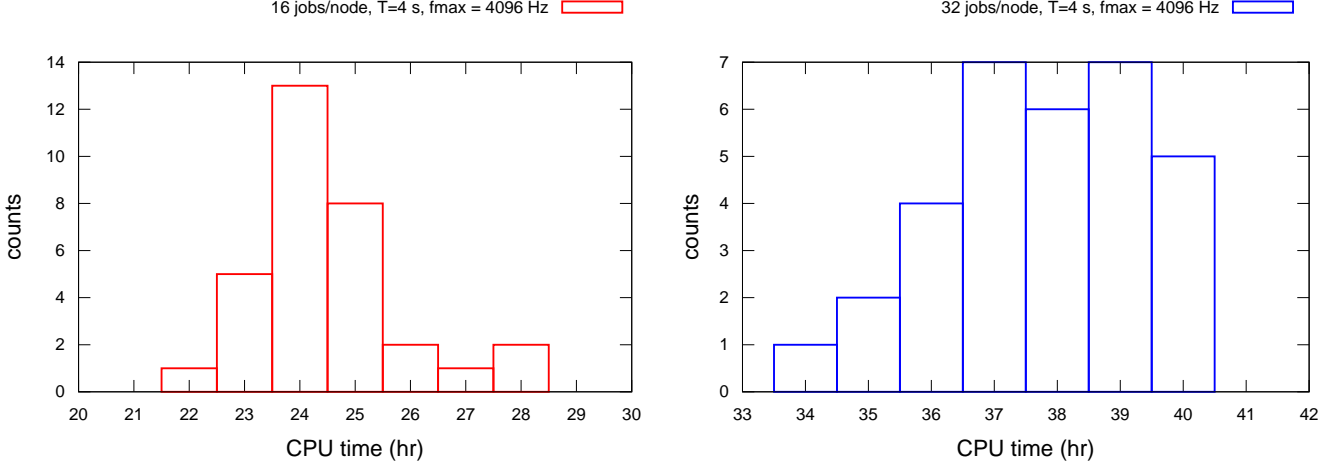


FIG. 1. Histogram of CPU time for each **BayesWave** run. Benchmarking was performed on a dedicated node on CIT analyzing the 32 most significant **coherent WaveBurst** background events from S6D.

included as our benchmarking efforts continue. The left-hand plot (red histogram) shows the results when running 16 jobs at a time (one job per core). The right hand plot is running 36 jobs simultaneously (2 jobs per core). Why the increase in wall time is less than a factor of 2 is something we need to investigate.

### C. Estimated Search Cost

**BayesWave** is designed to act as a “follow-up” pipeline to interesting GPS times identified by other pipelines. By design, the run-time for a single **BayesWave** trigger is relatively stable from event to event. This is because **BayesWave** uses a fixed number of steps in the Markov-chain. For the parameter space discussed in this document, we have found that chain “convergence” can typically be achieved with chains using two million iterations, and 20 chains. This leads to a run time per trigger,  $\tau$ , of 20-24 hours, on a single core. Continued bench-marking and optimization of  $\tau$  is currently in progress, and is discussed in this document.

Because  $\tau$  is stable from event-to-event, the computational cost of a full search can be estimated using only  $\tau$  and the number of triggers to be analyzed,  $N$ . The total cost of a search in CPU-hours is simply  $\tau \times N$ . For each aspect of the search, we estimate the O1 computing cost. Extension to O2 and O3 is possible by scaling the expected livetime, and by multiplying by 3/2 for 3 detectors instead of 2.

#### 1. Online search and time-slides

In low-latency, we wish to identify significant candidate events, as well as produce parameter estimation results (waveform reconstruction, sky localization, etc.). Identifying event candidates can be achieved using a “3- $\sigma$ ” level background, which corresponds to  $\sim 300$  time-slides. Because we wish to follow-up events which may be sent to astronomers for EM follow-up, we set our threshold for follow-up at a **coherent WaveBurst** false alarm rate of 2 triggers per week.

For 8 weeks of livetime in O1, this leads to 16 triggers per lag, or  $N_{\text{online}} = 4800$  triggers for all lags in O1.

$$N_{\text{online}} = 2 \text{ triggers per week} \times 8 \text{ weeks} \times 300 \text{ time-slides} = 4800 \text{ triggers}$$

#### 2. Simulation

For simulation, we find that 300 injections per waveform type is sufficient to characterize the pipeline efficiency. The “standard” Burst MDC includes 33 waveform types. So, a full simulation run with **BayesWave** would require  $300 \times 33 = 10,000$  injections. We expect that such a “full” simulation set would be analyzed once for each observing run. So,  $N_{\text{simulation}} = 10,000$  triggers for each observing run.

$$N_{\text{simulation}} = 300 \text{ injections} \times 33 \text{ waveforms} = 10,000 \text{ triggers}$$

### 3. Development

For testing and de-bugging, we have found it useful to run jobs in batches of 250 triggers, on average once every 2 weeks. For 1 year, this is  $250 \text{ triggers} \times 26 \text{ runs} = 6500 \text{ triggers per year}$ . So, each year of development corresponds to  $N_{\text{development}} = 6500 \text{ triggers}$ .

$$N_{\text{development}} = 250 \text{ triggers} \times 26 \text{ runs} = 6500 \text{ triggers}$$

### 4. Detection Follow-up

**BayesWave** will run additional lags to accumulate high statistics for any possible detection candidate. To trigger this follow-up, an event must have a cWB FAR of less than 1 per 2 years. The additional follow-up targets a “4- $\sigma$ ” detection confidence level, and so requires 100,000 lags.

$$N_{\text{follow-up}} = 100,000 \text{ lags} \times 0.2 \text{ years livetime} \times 0.5 \text{ trigs/year} = 10,000 \text{ triggers}$$

### 5. Totals

We can compute the total search cost by taking the total number of triggers that need to be processed, and multiplying by the median run time of 24 hours per trigger.

Run	$N_{\text{Total}}$	SUs
O1	32,000 Triggers	770,000 SU
O2	66,000 Triggers	1,600,000 SU
O3	90,000 Triggers	2,200,000 SU

## III. PARAMETERS THAT EFFECT COMPUTING COST

1. The **coherent WaveBurst** FAR of the online search sets the production costs. Raising the **coherent WaveBurst** threshold would reduce cost, at the expense of potentially missing weak signals. This could be optimized using studies. This is likely a factor of  $\sim 2$  uncertainty in the production cost.
2. Simulations are used only to test the pipeline and set upper-limits - they are not used in detection statements. For this reason, it is possible to vary the number of simulations. We have attempted to quantify the minimum number needed to explore parameter space. However, there are possible trade-offs in the number of simulations we perform. This leads to a factor of  $\sim 2$  uncertainty in the simulation cost.
3. The **coherent WaveBurst** FAR used for detection follow-up has similar trade-offs as in production. Reducing this threshold would reduce computing costs, potentially at the expense of missing marginally detectable signals.

For each **BayesWave** analysis the run time is determined by the number of chains used for the thermodynamic integration, the number of RJMCMC steps for each chain, and the time-frequency volume of the data being analyzed. The default settings for these key quantities have been determined empirically through analysis of background events and injections. There are many command line options that control proposal distributions that drive the RJMCMC exploration of the parameter space, and additional “knobs” to tune the prior distributions that define our signal and glitch models.

Options relating to the proposal distributions impact the computational efficiency of the search through the correlation lengths of the Markov Chains, thereby setting the demand for the number of samples in the chain. Options controlling the prior distributions impact the overall effectiveness of the algorithm in terms of the fidelity of the waveform reconstructions and the ability to distinguish signals from glitches. The prior on the number of wavelets can have a significant impact on the computational cost.

There is a complicated interplay between the settings used to define the priors and the settings in the proposal distributions that result in the shortest correlation lengths. We have begun a systematic exploration of this high dimensional space of possible settings.

## IV. COMPLETED OPTIMIZATION WORK

### A. Methodological

The best way to increase the efficiency of a Markov Chain Monte Carlo algorithm is to improve the frequency with which proposed trial locations in parameter space are accepted by the chain. Time spent developing good proposals is time saved when running the MCMC. We use a variety of custom time-frequency proposals for placing the wavelets. One is based on a pilot run with maximization over phase, time and amplitude that locates regions of excess power. This is then used to produce a proposal distribution that preferentially proposes adding wavelets in regions where there is excess power. We also use a time-frequency proposal that preferentially proposes to add new wavelets in the vicinity of existing wavelets in the model.

For the GW model we take advantage of the known degeneracy in sky location parameters along ring of constant time delay between detectors in a 2-interferometer network. New sky location parameters are preferential chosen to lie along the ring on the celestial sphere.

### B. Algorithmic

We have implemented several algorithmic “tricks” to minimize the computational cost of each sample in the MCMC.

1. **Recursion relations for sine-Gaussian phase:** The Fourier domain wavelet model can be separated into a frequency dependent amplitude  $\mathcal{A}(f)$  and a frequency dependent phase term  $e^{i\Phi(f)}$  where  $\Phi(f) \sim 2\pi t_0(f - f_0)$ . A brute-force calculation of the phase would require calling trigonometric functions at each Fourier bin. Instead we take advantage of the recursion relationship for the  $j^{\text{th}}$  Fourier bin:

$$e^{-i2\pi\Delta f j} = e^{-i2\pi\Delta f (j-1)} + e^{-i2\pi\Delta f (j-1)} (e^{-i2\pi\Delta f} - 1)$$

where each Fourier sample is separated by  $\Delta f = 1/T$  and  $T$  is the total observation time. Using this relation we only need to use trigonometric functions to determine the phase at the lowest frequency in the bandwidth of interest and the phase shift term  $e^{-i2\pi\Delta f}$ . We similarly take advantage of this relationship when time-shifting the GW model to compute the response of each detector.

2. **Limited bandwidth for sine-Gaussian waveforms:** The wavelet basis functions we use for the signal and glitch model have compact time frequency support, with bandwidth determined by  $2\pi f/Q$ . When we propose new wavelet parameters we only need to compute changes to our model over the bandwidth of the signal
3. **Computing likelihood only over frequency bins where model has changed:** We can similarly take advantage of the small bandwidth of the wavelets by limiting how many samples are included in the likelihood evaluation at each point in the MCMC. Typically the MCMC algorithm uses the ratio of the likelihood at the proposed parameters to the current parameters to determine if the new position will be adopted. The likelihood ratio is 1 everywhere except in frequency bins where the waveforms have changed, i.e. over a bandwidth much smaller than the full band of the data. We can directly compute the likelihood ratio by only considering how the frequencies encompassed by the bandwidth of the wavelet with a proposed update change.
4. **Adaptive temperature spacing for parallel tempering:** We have recently adopted the adaptive temperature scheme in [3]. Doing so has enabled us to use 20% fewer chains while achieving better convergence for the MCMC using our fixed  $2 \times 10^6$  samples.

### C. Code

We have started a systematic optimization process of the `BayesWave` code. Our first step was to add the `-O3` flag when compiling the code with `gcc`. By looking at runtime values in Condor logfiles, we observed that this increased speeds by a factor of 3. As a second step, we set up a test run and profiled it. The test run we used consists of a signal injection from an xml table and 5000 MCMC iterations. We profiled the code performance by using the `perf` tool and recording cpu-clock events on the CIT pcdev1 headnode for our short test run. According to the profiling results, the top 4 functions in terms of runtime were: `ThermodynamicIntegration` ( $\sim 25\%$ ), `EvaluateMarkovianLogLikelihood` ( $\sim 24\%$ ), `__ieee754_exp` ( $\sim 13\%$ ), and `__ieee754_log` ( $\sim 6\%$ ). Two `BayesWave` functions were, therefore, taking up 50% of the runtime. By looking at the profiling results of the functions individually, we were able to spot a major

bottleneck within `EvaluateMarkovianLogLikelihood`, caused by two `if` conditions within two nested loops. By rearranging this section of the code, we obtained a consistent speed-up: the runtime was reduced by 17%. Secondly, we went through the code routines and systematically used strict inequalities, introduced more auxiliary variables to avoid duplicate operations, and favoured multiplication over division operations, where possible. This exercise pushed the speed-up to  $\sim 22\%$ . At this point we ran a production run (i.e., the same setup of the test run, but with 1500000 iterations) on Condor and found that the changes introduced had reduced the runtime by about  $\sim 32\%$ .

## V. ON-GOING OPTIMIZATION WORK

### A. Methodological

We have identified a number of straight-forward optimizations that should be simple to implement. Substantial testing will be needed before these optimizations can be deemed “ready” for use in a search. Our goal is to have these implemented and tested in time for O2, but we have not accounted for any gains in performance for our computing request that rely on code that does not yet exist.

1. **Hierarchical model comparisons:** `BayesWave` serves two main functions, parameter estimation and model selection. Parameter estimation requires far fewer chains in the temperature ladder than is needed in the thermodynamic integration of the evidence used in model selection. The difference is a factor of  $\sim 5$  speed up. We are investigating ways to get crude estimates of the evidence with the few chains needed for parameter estimation in a hierarchical scheme where the more expensive full evidence calculations are only performed when there is not a clear winner between the noise, glitch and signal models. This will significantly reduce the total cost of the background analysis – our experience from running on glitches from S6 has shown that  $\sim 80\%$  of background events are easily identifiable as glitches by `BayesWave` and only  $\sim 20\%$  would require the full evidence integral follow up. There is potential for great savings if this trend persists for the modern implementation of `coherent WaveBurst` and for Advanced LIGO background glitches.
2. **Different TFV for different triggers:** The CPU demand of a `BayesWave` run is approximately proportional to the amount of data being used in the analysis. `BayesWave` uses trigger times from `coherent WaveBurst` to select which data to analyze. `BayesWave` uses a 4 second window centered on the GPS time from the search and a sampling rate of 4096 Hz. The sampling rate is chosen so that `BayesWave`’s bandwidth is similar to the standard `coherent WaveBurst` all sky search. Because we are simultaneously estimating the noise power spectrum we choose a segment length of 4 seconds to ensure that we have enough data for good spectral estimation down to 30 Hz. The exact choice of segment length and bandwidth could be adapted depending on the maximum likelihood central frequency of the trigger. Low frequency triggers, say around 200 Hz, could use a lower sampling rate (i.e. 1024 Hz) which would improve run time by a factor of  $\sim 4$  while high frequency triggers at  $\sim 1$  kHz could use shorter segments of data and only consider frequencies above  $\sim 256$  Hz so that we still have reliable spectral estimation. Exactly where to divide the follow-up analysis will need to be optimized, and care will be required to ensure that we still achieve good spectral estimation and that the likelihood support for our glitch/signal model does not go to the edge of the restricted prior range on time and frequency.

### B. Algorithmic

### C. Code

We have recently started experimenting with architecture-dependent compilation flags and, so far, were able to speed-up our test run further ( $\sim 25\%$  with respect to our initial, un-optimized test run performed with a -O3 compiled executable). This speed up should correspond to a  $\sim 35\%$  speed-up in a production run, with respect to the original code. We plan to experiment further in this direction and to continue with the profiling work that we started recently. Additionally, we will consider the possibility of parallelizing the most computationally expensive parts of our code.

## VI. FUTURE OPTIMIZATION

### A. Methodological

The key is to get a large number of *independent* samples of the posterior distribution for the model parameters. MCMC sampling techniques produce correlated samples with some correlation length.

One way to cut the computational cost is to reduce the correlation length, so that the same number of independent samples can be harvested from a smaller number of evaluations of the prior and likelihood. Note that the correlation length is not the same for all parameters. Correlation lengths for sky location parameters (for making sky maps) are of order 1000 samples while the correlation length for the likelihood (for evidence calculation) is of order 100 samples. It can be tricky to compute correlation lengths in a RJMCMC since the dimension of the parameter space changes. The situation is especially difficult for the wavelet parameters, as the wavelets blink in and out of existence from one sample to the next and there is no reliable way to identify which parameters belong to a physical state. We can measure the autocorrelation of the model dimensions indicator variable, which tells us how well the chains are executing trans-dimensional moves ( $\sim 200 - 300$  for our standard test-sure background event). We can also do tests where we fix the model dimension and turn off the wavelet swapping transitions (so-called birth-death moves), so that we can compute correlation lengths for the wavelet parameters.

Through years of effort we have already significantly reduced the correlation lengths using specially designed proposal distributions for the extrinsic parameters in the signal model and for the time-frequency location of wavelets. Further improvements may be possible by continuing the development of custom proposals.

1. **Joint wavelet Fisher matrix proposals** Currently the wavelet parameter updates are based on a Fisher matrix approximation to the likelihood that assumes no correlation between the parameters of a single wavelet, nor any correlations between the parameters of different wavelets. Each wavelet has its own Fisher matrix estimate, and the parameters of wavelets are updated independently in separate steps of the MCMC.

We know that both of these assumptions are incorrect. Correlation lengths can be reduced by using the full Fisher matrix which accounts for parameter correlations, and updating parameters for all of the wavelets in the model at each MCMC step. The trade-off here comes from computing this larger matrix and numerically inverting it to find eigenvectors for a jump proposal. Treating the wavelets independently means these computations can be done analytically. Code development is required to implement and tune the more complicated proposal and thorough benchmarking will be needed to test whether the reduction in the correlation length is generically offset by the additional cost of the more complicated jump proposal.

2. **Extrinsic proposals for 3-detector network** There are known degeneracies for extrinsic parameters that can be taken advantage of in jump proposals. We have already implemented a proposal that preferentially draws sky-location samples along the ring of constant time delay between a pair of interferometers. When a third detector is added to the network (e.g. Virgo) the ring degeneracy is broken but a near degeneracy exists by reflecting the current sky-location across the plane of the detectors. The sky location and relative polarization angle for the two modes in the posterior is known analytically and can be used in a jump proposal. This is a low priority development task as we focus on our readiness for O1.

### B. Algorithmic

1. **Analytic marginalization of time and/or phase parameters:** The output of a Markov chain is a list of samples from the posterior distribution function of the model which are then marginalized over nuisance parameters to arrive at the quantities of interest (e.g. sky maps, credible intervals on parameters, etc.). Furthermore, the evidence is the posterior marginalized over all model parameters. Marginalization over time and phase parameters can be done analytically [4]. Marginalizing over time and phase parameters complicates the likelihood evaluations (introducing FFTs and Bessel functions) but reduces the complexity of the model and the dynamic range of the likelihood. For CBC PE codes the trade off in sampling efficiency has been worth the added cost of likelihood evaluations. It is worth experimenting with in **BayesWave** but, because our signal/glitch model is so inexpensive compared to the CBC waveforms, it may prove to be more computationally efficient to continue doing that marginalization numerically. Implementing the marginalized likelihoods is a major overhaul of the code. Because it is a large investment and not a guaranteed return

## C. Code

## VII. — INTERNAL NOTES —

All searches with non-trivial computing costs should be sure to address the following items, quoted directly from the NSF charge for the March 2015 review:

- [Provide performance numbers in absolute times to identify performance rate limiting steps.]
- [Justify the choices of compilers and libraries.]
- [Identify and characterize computationally expensive kernels, reporting on measured performance, using descriptive measures such as: What percentage of peak Flops is achieved? What is the percentage of the bandwidth utilized? Is the kernel vectorized? What is the quality of the vectorization? What is the potential for performance optimization of these kernels?]
- [Report on your evaluation of the parallelization of the code and evaluation of multi-threading to improve performance.] Markov Chain Monte Carlo algorithms are, by construction, serial stochastic samplers. We use parallel tempering (running multiple MCMCs) which could be sent to separate cores via MPI as is done in the CBC PE codes [4]. Parallel chains need to be in frequent communication, so even though multi-threading decreases wall time CPU time may increase due to the cost of passing information between cores. Different models (glitch, signal, Gaussian noise) could be run separately, but they all share memory from the PSD-estimation step at the beginning of each run. It is beyond our expertise, and therefore not obvious to us, if there are other benefits to parallelizing the code. Nor is it clear to us which is preferred – using a single for  $\sim 1$  day per job, or  $\sim 16$  cores for  $\sim 1/16$  days.
- [Report on your evaluation of hardware accelerators and inexpensive single precision GPUs [and MiC? accelerators].]
- [Specify future production and development computing requirements in SUs rather than cores, along with schedule for the total number of SUs needed over time, while ensuring adequate computing at peak load.]

- 
- [1] N. J. Cornish and T. B. Littenberg, arXiv:1410.3835 (2014), arXiv:1410.3835 [gr-qc].  
[2] T. B. Littenberg and N. J. Cornish, arXiv:1410.3852 (2014), arXiv:1410.3852 [gr-qc].  
[3] W. Vousden, W. M. Farr, and I. Mandel, ArXiv e-prints (2015), arXiv:1501.05823 [astro-ph.IM].  
[4] J. Veitch, V. Raymond, B. Farr, W. Farr, P. Graff, *et al.*, Phys.Rev. **D91**, 042003 (2015), arXiv:1409.7215 [gr-qc].