BayesWave User's Guide

Authors: Neil Cornish, Tyson Littenberg

Developers: Jonah Kanner, Francesco Pannarale

Abstract. This document is a work-in-progress User's Guide for the BayesWave software package. BayesWave uses a linear combination of wavelets to model non-Gaussian GW data. It can be used to reconstruct and remove glitches, or as a Burst parameter estimation code. Under the default settings the code outputs Bayesian evidence for three models under consideration: Gaussian noise only, Gaussian noise plus glitches, or Gaussian noise plus a GW signal. Optional command line arguments can restrict the number of models used (e.g., the glitch only model for noise characterization, or the signal only model for rapid parameter estimation), deploy BayesWave as a glitch cleaning algorithm, or simultaneously fit for glitches and gravitational waves.

1. Disclaimer

This software is VERY MUCH in development and this User's Guide is in a very incomplete and amateur state. Things are changing on a weekly basis as we apply the code to different tasks and add new features to improve performance and run time. Bear with us.

2. Installation

To check out the SVN:

```
svn checkout https://svn.ligo.caltech.edu/svn/bayeswave
```

Anyone with LIGO credentials should be able to checkout the code, but commits are restricted to developers. To find the source code:

```
cd [path to]/bayeswave/trunk/src/
```

Included in the src directory is a Makefile. The first two lines have to be modified to point to your favorite installation of `LAL`, `GSL`, and `FFTW`. If you are building `BayesWaveBurst` on the LDG the makefile should work as-is. Build with:

```
make
```

which will create executables for the main analysis and post-processing called `BayesWaveBurst` and `BayesWavePost`

Back to Table of Contents

3. Running `BayesWaveBurst`

```
BayesWaveBurst --help
```

will display the required and (many) optional command line arguments. In the near future a detailed description of what each option does will appear in a more thorough User's Guide.

```
REQUIRED:
--ifo IFO interferometer (H1,L1,V1)
--IFO-flow minimum frequency, preferably 2^N (Hz)
--IFO-cache full path to cache file
--IFO-channel channel name (IFO:LSC-STRAIN)
--trigtime GPS trigger time
--srate sampling rate (Hz)
--seglen duration of data (s)
--PSDstart GPS start time for PSD estimation
--PSDlength duration of PSD estimation length
--dataseed Required if using simulated noise e.g.  --H1-cache LALAdLIGO

OPTIONAL:
------------------------------------------------------------------------------
--- Run parameters -------------------------------------------------------------
------------------------------------------------------------------------------
--Niter number of iterations (2000000)
--NCmin minimum number of parallel chains (25)
--NCmax minimum number of parallel chains (25)
--Ncycle number of model updates per BurstRJMCMC iteration (100)
--Nburnin number of burn-in iterations (100000)
--chainSeed random number seed for Markov chain (1234)
--runName run name for output files
--0noise no noise realization
--zeroLogL logL = constant test
--restart restart run using psd and residual from file
--gnuplot output files for gnuplot animations
--verbose output hot chains
--window duration of time window for model characterization runs

------------------------------------------------------------------------------
--- Model parameters -----------------------------------------------------------
------------------------------------------------------------------------------
--fullOnly require signal && glitch model
--noClean skip cleaning phase and go right to reduced window
--cleanOnly run bayesline & glitch cleaning phase only
--noiseOnly use noise model only (no signal or glitches)
--signalOnly use signal model only (no glitches)
--glitchOnly use glitch model only (no signal)
--noPSDfit keep PSD parameters fixed
--bayesLine use BayesLine for PSD model
```

```
------------------------------------------------------------------------------
--- Priors & Proposals -------------------------------------------------------
------------------------------------------------------------------------------
--Dmin minimum number of wavelets total (1)
--Dmax maximum number of wavelets per channel (20)
--clusterPrior use metric-based clustering prior
--clusterAlpha distance between wavelets to be considered a cluster (2)
--clusterBeta cluster prior exp(-beta K) (4)
--clusterGamma occam penalty exp(gamma*J) (0)
--noAmplitudePrior don't use SNR-dependent amplitude prior
--clusterProposal don't use clustering & TF density proposal

------------------------------------------------------------------------------
--- Parallel Tempering parameters --------------------------------------------
------------------------------------------------------------------------------
--tempMin minimum temperature chain (1)
--adaptTemperature adjust PT ladder to maintain acc.  rate from 40% to
60%
--tempSpacing set temperature spacing for geometric ladder (1.5)

------------------------------------------------------------------------------
--- LALInference injection options -------------------------------------------
------------------------------------------------------------------------------
--inj injfile.xml Injection XML file to use
--event N Event number from Injection XML file to use

------------------------------------------------------------------------------
--- Burst MDC injection options ----------------------------------------------
------------------------------------------------------------------------------
--MDC-channel IFO1-chan, IFO2-chan, etc
--MDC-cache IFO1-mdcframe, IFO2-mdcframe, etc
--MDC-prefactor Rescale injection amplitude (1.0)

EXAMPLE:
BayesWaveBurst
--ifo H1 --H1-flow 32 --H1-cache LALSimAdLIGO --H1-channel LALSimAdLIGO
--trigtime 900000000.00 --srate 512 --seglen 4
--PSDstart 900000000 --PSDlength 4
--NCmin 2 --NCmax 2
--dataseed 1234
```

Move to a new directory and try running the example command line. You will have to add the src directory to your path variable

```
export PATH=[path to]/bayeswave/trunk/src/:$PATH
```

This will run for ∼20 minutes. Real runs take ∼ 12 − 24 hours, depending on the settings and the models being used. Really long segments (1024 s) of really glitchy data can take a week.

This run will simulate Advanced LIGO data for H1 with a random seed for the noise realization `dataseed` = 1234; a low-frequency cutoff of `IFO-flow` = 32 Hz; a total observation time of `seglen` = 4 s; and a sampling rate of `srate` = 512 Hz.

Because we are using `LALInference` for data handling (which expects in-spiral signals) the `trigtime` argument gives the GPS time which will be analyzed. **The analysis window will be `seglen` seconds long and will end 2 seconds after `trigtime` .** For `seglen` T the start time will be

$$t_0 = \texttt{trigtime} + 2 - \texttt{seglen} \tag{1}$$

By default, the code runs through 4 phases:

(i) *Cleaning*: `BayesWave` assumes all non-Gaussian features are glitches and estimates the PSD.

(ii) *Noise Only*: Samples the posterior and computes the evidence for the Gaussian noise model (the data contains only Gaussian noise)

(iii) *Glitch Only*: Samples the posterior and computes the evidence for the Glitch model (the data contains Gaussian noise and at least one glitch)

(iv) *Signal Only*: Samples the posterior and computes the evidence for the Signal model (the data contains Gaussian noise and a GW signal).

Back to Table of Contents

4. `BayesWaveBurst` Output

When `BayesWave` is finished you will find several files in the run directory and three subdirectories (`snr/`, `chains/`, and `waveforms/`). The directories `snr/` and `waveforms/` will be empty if you use the example command line above.

4.1. Run directory

The *.run files contain settings for each phase of the run

`bayeswave.run` [settings for full run, plus BayesWaveBurst command line]
`clean.run` [cleaning phase]
`noise.run` [Gaussian noise phase]
`glitch.run` [Glitch+Gaussian noise phase]
`signal.run` [Signal+Gaussian noise phase]

The `[model]_evidence.dat` files contain the integrand of the evidence (Z) integral. They are used for diagnosing the run. The file `evidence.dat` contains the computed evidence for each model. The columns are:

`model`, $\ln Z_{\texttt{model}}$, $\mathrm{Var}(\ln Z_{\texttt{model}})$

Results from the example command should be similar to:

```
noise 0.142466 0.00032892
glitch 0.0188961 0.00180285
signal 0 0.0018743
```

Everything is normalized by the signal model so it should always be 0. The Bayes Factor between models $B_{X,Y}$ is the evidence ratio, so

$$B_{\texttt{signal,noise}} = \exp(\ln Z_{\texttt{signal}} - \ln Z_{\texttt{noise}}) \sim 0.9 : 1. \tag{2}$$

The `evidence_stacked.dat` output file has the same information as `evidence.dat` but in a single row:

$\ln Z_{\texttt{noise}}, \ln Z_{\texttt{glitch}}, \ln Z_{\texttt{signal}}, \mathrm{Var}(\ln Z_{\texttt{noise}}), \mathrm{Var}(\ln Z_{\texttt{glitch}}), \mathrm{Var}(\ln Z_{\texttt{signal}})$

so that many runs can be concatenated and plotted together.

4.2. Chains directory

The `chains/` directory contains samples from the Markov chains. There are five types of chain files, some for repeated for different models:

`fourier_domain_data_ifo`I`.dat`
`[clean/noise/glitch/signal/full]_intchain.dat.0`
`[signal/full]_extchain.dat.0`
`[clean/glitch/clean/full]_glitchchain_ifo`I`.dat.0`
`[signal/full]_wavechain.dat.0`

where $I = [0, N_{IFO} - 1]$ and N_{IFO} is the number of IFOs,

The `intchain` files contain roughly the same information as the intrinsic parameter (which is roughly the same as the intrinsic parameters from the chain – these files need to be renamed and reformatted):

cycle, ln likelihood, $N_{\texttt{signal}}, N_{\texttt{glitch}}^{I=0}, ..., N_{\texttt{glitch}}^{I=N_{IFO}-1}, \eta^{I=0}, ..., \eta^{I=N_{IFO}-1}$, ln prior

The `extchain` files contain the extrinsic parameters of the signal model – this is mostly redundant with `wavechain` files but easier to parse:

cycle, ln likelihood, r.a., sin(dec), $\psi, e, \delta A, \delta\phi, \delta t$

The `glitchchain` files contain the parameters of wavelets in the glitch model for `ifo`I:

$N_{\texttt{glitch}}, \vec{\gamma}[0], ..., \vec{\gamma}[N_{\texttt{glitch}} - 1]$

where $\vec{\gamma}[i] \rightarrow \{f[i], t[i], Q[i], A[i], \phi[i]\}$.
Finally, the `wavechain` files which have the parameters for the signal model, including extrinsic and wavelet parameters

$N_{\texttt{signal}}, \text{r.a.}, \sin(\text{dec}), \psi, e, \delta A, \delta\phi, \delta t, \vec{\gamma}[0], ..., \vec{\gamma}[N_{\texttt{signal}} - 1]$

Back to Table of Contents

5. Post processing with `BayesWavePost`

The `BayesWavePost` code reads the output files from `BayesWaveBurst` and recreates the model for each step of the Markov chain. From the recreated model different quantities of interest (e.g. the waveform "moments" are computed. The result is a Markov chain for the different moments from which posterior distributions can be computed.

```
BayesWavePost --help
```

will display the required and (fewer) optional command line arguments.

```
REQUIRED:
--ifo IFO interferometer (H1,L1,V1)
--IFO-flow minimum frequency, preferably 2^N (Hz)
--IFO-cache LALSimAdLIGO: MUST USE SIMULATED DATA
--trigtime GPS trigger time
--srate sampling rate (Hz)
--seglen duration of data (s)
--PSDstart GPS start time for PSD estimation
--PSDlength duration of PSD estimation length
--dataseed 0000:  REQUIRED FOR SIMULATED NOISE
--0noise no noise realization:  SET SIMULATED NOISE TO 0

OPTIONAL:
-------------------------------------------------------------------------------
--- Run parameters ------------------------------------------------------------
-------------------------------------------------------------------------------
--runName run name for output files

-------------------------------------------------------------------------------
--- Model parameters ----------------------------------------------------------
-------------------------------------------------------------------------------
--bayesLine use BayesLine for PSD model

-------------------------------------------------------------------------------
--- LALInference injection options --------------------------------------------
-------------------------------------------------------------------------------
--inj injfile.xml Injection XML file to use
--event N Event number from Injection XML file to use

-------------------------------------------------------------------------------
--- Burst MDC injection options -----------------------------------------------
-------------------------------------------------------------------------------
--MDC-channel IFO1-chan, IFO2-chan, etc
--MDC-cache IFO1-mdcframe, IFO2-mdcframe, etc
--MDC-prefactor Rescale injection amplitude (1.0)

EXAMPLE:
BayesWavePost
--ifo H1 --H1-flow 32 --H1-cache LALSimAdLIGO
--trigtime 900000000.00 --srate 512 --seglen 4
--PSDstart 900000000 --PSDlength 4
--dataseed 0000 --0noise
```

Most of the command line arguments, such as the sampling rate and trigger time, must be identical to the `BayesWaveBurst` run that you used. The three lines highlighted in red are the exception. Regardless of your `BayesWaveBurst` command line arguments `BayesWavePost` **must** be given a simulated noise cache (`LALSimAdLIOG`), a random seed for the simulated noise (`--dataseed`) and, most importantly, the `--Onoise` argument which explicitly sets the simulated noise realization to all zeroes. This is a bit of a hack, but the point is to have the injected waveform saved in memory without any noise so that we can compute overlaps etc. between our recovered signals and the injection. Because the burst MDC injections and `LALSimulation` XML injection tables are handled differently in the data-handling part of the code, we felt this was the easiest way to get the injected waveforms all by themselves.

Back to Table of Contents

6. `BayesWavePost` Output

When `BayesWavePost` is finished you will find a new directory `post/` containing:

`freqsamp.dat`
`timesamp.dat`
`[clean/glitch/injection/signal]_[colored/whitened]_moments.dat.`I
`[clean/glitch/signal]_recovered_[colored/whitened]_waveform.dat.`I
`[clean/glitch/signal]_stats.dat.`I
`[colored/whitened]_data.dat.`I
`injected_[colored/whitened]_waveform.dat.`I

where $I = [0, N_{IFO} - 1]$ and N_{IFO} is the number of IFOs. Many of the files have "colored" and "whitened" versions. The "colored" files correspond to the model as seen by the detectors, the "whitened" files have been normalized by the noise PSD.

The `*_moments.dat.`I files contain different scalar moments computed for each waveform. Each row of the file corresponds to the moments at a different iteration of the Markov chain. The columns correspond to the moments being computed.:

$$\text{SNR}, \text{energy}, h_{\text{rss}}, t_0, \Delta t, f_0, \Delta f, \mathcal{O}^I[h_{\text{inj}}, h_{\text{rec}}], \mathcal{O}^{\text{Net}}[h_{\text{inj}}, h_{\text{rec}}]$$

where Δt is the duration, Δf is the bandwidth, and $\mathcal{O}[a, b]$ is the overlap between a and b. Superscript I, as with the filenames, corresponds to a single detector while the Net superscript is for the network (therefore redundant in the different moments files). All of these quantities are defined at

https://wiki.ligo.org/pub/Bursts/AllSkyPE/moments.pdf

The `*_waveform.dat.`I files contain the time domain waveforms printed in a single row making them unreadable by human. The `injected_*` waveform is a single row containing the injected waveform. The `[clean/glitch/signal]_recovered_*` files contain

a single row for each sample in the Markov chain. These files are parsed further downstream by scripts which generate the output pages.

Back to Table of Contents

7. Tips

7.1. Use Cases

You can find an assortment of example use cases at

https://ldas-jobs.ligo.caltech.edu/~tyson/BayesWave/examples/

including run scripts (`script.sh`) and output pages (`index.html`). The different use-cases can be used as unit tests before you set out on your own study. Available examples and a brief description follow

7.1.1. Quick test run – `test/` This example matches the suggested command line from the `--help` output of `BayesWaveBurst` and `BayesWavePost`. The run will take $\sim$ 20 minutes, running on a single interferometer (`--IFO H1`) using simulated Gaussian advanced LIGO noise (`--H1-cache LALSimAdLIGO --H1-channel H1:LALSimAdLIGO`), a low sampling rate (`--srate 512` in Hz) and the minimum number of chains (`--NCmin 2`) required by the post-processing tools. To compute reliable evidence we typically use 25 chains. Back to Table of Contents

7.1.2. Burst MDC injection into simulated data – `burst_mdc_injection/` Here is an example for using BurstMDC injections. The first line of the script secure copies the frame file to your local directory. The frame file is big so this will take a while. If you are running on CIT I would recommend pointing your `lalapps_path2cache` line directly to the copy in the example directory and commenting out the `gsiscp` line. Without the secure copy the run will take $\sim$ 10 hours. This particular case injects a sine-Gaussian waveform into simulated noise. If you compare the `BayesWaveBurst` command line for this event to the `test/` run you will find an additional set of interferometer arguments (`--IFO L1 --L1-cache LALSimAdLIGO --L1-channel L1:LALSimAdLIGO`), a sufficient number of chains to compute the evidence (`--NCmin 25 --NCmax 25`) and three additional command line arguments all starting with `MDC--` which handle the injection. The `--MDC-cache` file is looking for a LIGO cache file (as made by, e.g., `lalapps_path2cache`) which contains the burst MDC frame file, the `--MDC-channel` arguments specify which channel of the frame file `HL-SineGaussian-968046516-4095.gwf` contains the injection, and the `--MDC-prefactor` is an overall amplitude scaling of the injection to control the signal-to-noise ratio. **Note that the same cache file can contain paths to injection frame files for different interferometers which is not the case for using real**

detector noise with the `--IFO-cache` arguments. Back to Table of Contents

7.1.3. CBC injection in simulated data – `cbc_xml_injection/` Back to Table of Contents

7.1.4. Background glitch rejection – `cwb_background_event/` Getting the time slide right is tricky. This example corresponds to a coherent WaveBurst background event from S6D. The event summary file has GPS times for the glitch in each detector. From this we compute the correct time-slide. In this example, the trigger time corresponds to H1 (GPS_{H1}) so we must shift the L1 data to recreate the background event.

$$\Delta t_{L1} = GPS_{H1} - GPS_{L1} \tag{3}$$

The time slides should all be integer seconds but the difference of the GPS time will not be because there are not enough significant digits in the summary tables. Be careful rounding to the correct nearest integer when the time slide is negative. Heres a snippet of C code that I used to compute the time slides:

```
if(lag>0.0)
{
if( fabs(lag-(int)(lag)) < 0.5 ) lag=(int)(lag);
else lag=(int)(lag)+1.;
}
else
{
if( fabs(lag-(int)(lag)) < 0.5 ) lag=(int)(lag);
else lag=(int)(lag)-1.;
}
```

Back to Table of Contents

7.2. `BayesLine`

Back to Table of Contents

7.3. Cache files

```
ligo_data_find -o [observatory] -t [channel type] -s [start time] -e [end
time] --lal-cache -u file > [name].cache
```

`[observatory]` H,L,V **(not H1,L1,V1)**
`[channel type]` using H1,L1,V1 for `[IFO]`
`[IFO]_RDS_R_L1` S6 LIGO DARM_ERR
`[IFO]_RDS_C03_L2` S5 LIGO $h(t)$
`HrecOnline` VSR1 $h(t)$
`[IFO]_LDAS_C02_L2` S6 LIGO $h(t)$

Example:

```
ligo_data_find -o L -t L1_RDS_C03_L2 -s 875000000 -e 875000100 --lal-cache
-u file > 875000000-L1.cache
```

Back to Table of Contents

7.4. Condor

Back to Table of Contents

7.5. Glitch cleaning

Back to Table of Contents

7.6. Channels

We are set up to run on h(t) and DARM-ERR. If other channels are of interest they are easy to add, we just need to know reasonable ranges for the PSD. Currently the supported channel types are For H1 or L1:

`LDAS-STRAIN`
`LSC-DARM_ERR`
`LALAdLIGO` (advanced LIGO simulated data)
`LALLIGO` (initial LIGO simulated data)

and for V1:

`h_16384Hz` (for Virgo)
`LALAdLIGO` (advanced LIGO simulated data)
`LALLIGO` (initial LIGO simulated data)

If the `--IFO-channel` option is not one of the above the code will run and may work, but the channel name is used to set priors for PSD parameters so it could be bad.

Back to Table of Contents