Technical Note LIGO-T080011-00-Z May 8, 2008

Coherent WaveBurst
wat version 4.7.0

S. Klimenko (UF), I. Yakushin (LLO), A.Mercer (UF), G. Mitselmakher (UF)

Contents

1. Introduction

In this note we describe a coherent pipeline based on the constraint likelihood method [9] for detection of gravitational wave (GW) bursts in interferometric data. The pipeline is designed to work with arbitrary networks of gravitational wave interferometers. In general the resonant bar detectors can be also included in the network, but the current pipeline version does not support this option.

The pipeline consists of two stages: a) coherent trigger production stage, when the burst events are generated for a network of GW detectors and b) post-production stage, when additional selection cuts are applied to distinguish the GW candidates from the background events. This division into two stages is not fundamental. At both stages the pipeline executes coherent algorithms, based both on power of individual detectors and the cross-correlation between the detectors. It is essentially different from a traditional burst search pipeline, when, first, an excess power search is used for generation of triggers and, second, the coherent algorithms are applied [1, 11]. Instead, by using the likelihood approach, we combine in a coherent way the energy of individual detector responses into a single quantity called the network likelihood statistic, which has a meaning of the total signal-to-noise ratio of the GW signal detected in the network. The coherent triggers are generated if the network likelihood exceed some threshold which is a parameter of the search.

The coherent WaveBurst pipeline has essentially the same structure as the incoherent WaveBurst pipeline used in combination with the r-statistic [10] for the burst analysis (see Figure 1). Both pipelines share the same input, data conditioning and some of the output infrastructures, but the event trigger generators are different. The incoherent pipeline is based on the time-frequency coincidence of energy between the detector data streams and the coherent pipeline is based on the network likelihood.

The coherent algorithms are implemented in the Wavelet Analysis Tool (WAT) developed at University of Florida, which in turn is one of the toolboxes in the LIGO Data Monitoring Tool (DMT). All the software is stored in CVS [7]. The WAT package is available also as a standalone library [8].

2. Coherent analysis

Coherent network analysis is addressing a problem of detection and reconstruction of gravitational waves with the networks of GW detectors. In coherent methods, a statistic is built as a coherent sum over detector responses and, in general, it is more optimal (better sensitivity at the same false alarm rate) than the detection statistics of individual detectors. Also coherent methods provide estimators for the GW waveforms and the source coordinates on the sky.

2.1. Constraint Likelihood

The coherent WaveBurst pipeline uses a method for a coherent detection and reconstruction of burst signals based on the use of the likelihood ratio functional [9]. For a general case of Gaussian quasi-stationary noise it can be written in the wavelet (time-frequency) domain as

$$\mathcal{L} = \sum_{k=1}^{K} \sum_{i,j=1}^{N} \left(\frac{w_k^2[i,j]}{\sigma_k^2[i,j]} - \frac{(w_k[i,j] - \xi_k[i,j])^2}{\sigma_k^2[i,j]} \right) , \tag{1}$$

where K is the number of detectors in the network, $w_k[i,j]$ is the sampled detector data (time i and frequency j indexes run over some TF area of size N) and $\xi_k[i,j]$ are the detector responses. Note, we omit the term 1/2 in the conventional definition of the likelihood ratio. The detector noise is characterized by its standard deviation $\sigma_k[i,j]$, which may vary over the TF plane. The detector responses are written in the standard notations

$$\xi_k[i,j] = F_{+k} h_+[i,j] + F_{\times k} h_\times[i,j] , \tag{2}$$

where $F_{+k}(\theta,\phi)$, $F_{\times k}(\theta,\phi)$ are the detector antenna patterns (depend upon source coordinates θ and ϕ) and $h_+[i,j]$, $h_\times[i,j]$ are the two polarizations of the gravitational wave signal in the wave frame. Since the detector responses ξ_k are invariant with respect to the rotation around z-axis in the wave frame, the polarization angle is included in the definition of the h_+ and $h_\times$. The GW waveforms h_+ and $h_\times$ are found by variation of $\mathcal{L}$. The maximum likelihood ratio is obtained by substituting the solutions into the functional $\mathcal{L}$. The waveforms in time domain are reconstructed from the inverse wavelet transformation. Below, for convenience we introduce the data vector $\mathbf{w}[i,j]$ and the antenna pattern vectors $\mathbf{f}_+[i,j]$ and $\mathbf{f}_\times[i,j]$

$$\mathbf{w}[i,j] = \left\{ \frac{w_1[i,j]}{\sigma_1[i,j]}, .., \frac{w_K[i,j]}{\sigma_K[i,j]} \right\} \tag{3}$$

$$\mathbf{f}_{+(\times)}[i,j] = \left\{ \frac{F_{1+(\times)}}{\sigma_1[i,j]}, .., \frac{F_{K+(\times)}}{\sigma_K[i,j]} \right\} \tag{4}$$

Further in the text we omit the time-frequency indexes and replace the sum $\sum_{i,j=1}^{N}$ with $\sum_{\Omega_{TF}}$, where Ω_{TF} is the time-frequency area selected for the analysis.

The likelihood functional (Eq.1) can be written in the form $\mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2$:

$$\mathcal{L}_+ = \sum_{\Omega_{TF}} \left[2(\mathbf{w} \cdot \mathbf{f}_+) h_+ - |\mathbf{f}_+|^2 h_+^2 \right] , \tag{5}$$

$$\mathcal{L}_\times = \sum_{\Omega_{TF}} \left[2(\mathbf{w} \cdot \mathbf{f}_\times) h_\times - |\mathbf{f}_\times|^2 h_\times^2 \right] , \tag{6}$$

where the antenna pattern vectors $\mathbf{f}_+$ and $\mathbf{f}_\times$ are defined in the Dominant Polarization wave Frame (DPF) [9]. In this frame the antenna pattern vectors are orthogonal to each other: $(\mathbf{f}_+ \cdot \mathbf{f}_\times) = \mathbf{0}$. Note, the norms $|\mathbf{f}_+|^2$ and $|\mathbf{f}_\times|^2$ characterize the network sensitivity to the h_+ and $h_\times$ polarizations.

It is convenient to introduce the whiten GW waveforms g_+ and $g_\times$ and re-write the likelihood functionals Eq. 5,6 in the canonical form

$$\mathcal{L}_+ = \sum_{\Omega_{TF}} \left[2(\mathbf{w}\cdot\mathbf{e}_+)g_+ - g_+^2\right], \quad g_+ = |\mathbf{f}_+|h_+ \;, \tag{7}$$

$$\mathcal{L}_\times = \sum_{\Omega_{TF}} \left[2(\mathbf{w}\cdot\mathbf{e}_\times)g_\times - g_\times^2\right], \quad g_\times = |\mathbf{f}_\times|h_\times \;, \tag{8}$$

where $\mathbf{e}_+$ and $\mathbf{e}_\times$ are the unity vectors along the $\mathbf{f}_+$ and $\mathbf{f}_\times$ directions respectively. The estimators of the whiten GW waveforms are obtained by the variation of the canonical likelihood functionals

$$g_+ = (\mathbf{w}\cdot\mathbf{e}_+) \tag{9}$$

$$g_\times = (\mathbf{w}\cdot\mathbf{e}_\times). \tag{10}$$

The whiten detector responses are reconstructed from the waveforms g_+ and $g_\times$ as

$$s_k = \frac{\xi_k}{\sigma_k} = e_{k+}g_+ + e_{k\times}g_\times \;. \tag{11}$$

2.1.1. Likelihood regulators As first shown in [9], there is a specific class of constraints (often called regulators), which arise from the way the network responds to a generic gravitational wave signal. A classical example is a network of aligned detectors where the detector responses ξ_k are identical. Therefore the algorithm can be constrained to search for an unknown function ξ rather than for two gravitational wave polarizations h_+ and $h_\times$, which span much larger parameter space. Note, in this case $|\mathbf{f}_\times|^2 = 0$, the $\mathbf{e}_\times$ vector is not defined and the solution for the $h_\times$ waveform can not be found. The regulators are important not only for aligned detectors, but also for networks of miss-aligned detectors, for example, the LIGO and Virgo network [**?**, **?**]. Depending on the source location the network can be much less sensitive to the second GW component ($|\mathbf{f}_\times|^2 << |\mathbf{f}_+|^2$) and the $h_\times$ waveform may not be reconstructed from the noisy data.

In the coherent WaveBurst analysis we introduce a regulator by changing the norm of the $\mathbf{f}_\times$ vector

$$|\mathbf{f}'_\times|^2 = |\mathbf{f}_\times|^2 + \delta, \tag{12}$$

where δ is some parameter. This is equivalent to adding one more, dummy detector, to the network with the antenna patterns $f_{+,K+1} = 0$, $f_{\times,K+1} = \sqrt{\delta}$ and zero detector output ($x_{K+1} = 0$). In this case, the regulator preserves the orthogonality of the vectors $\mathbf{f}_+$ and $\mathbf{f}'_\times$ and the maximum likelihood statistics are written as

$$L_+ = \sum_{\Omega_{TF}} (\mathbf{w}\cdot\mathbf{e}_+)^2 \tag{13}$$

$$L_\times = \sum_{\Omega_{TF}} (\mathbf{w}\cdot\mathbf{e}'_\times)^2. \tag{14}$$

Depending on the value of the parameter δ different statistics can be generated, for example:

- $\delta = 0$ - standard likelihood,

- $\delta = \infty$ - hard constraint likelihood.

All mentioned above constraints are implemented in the WAT class "network" (see Section 5)

2.1.2. Coherent Energy Equation 13 shows that the likelihood ratio is a quadratic form

$$L_+ = \sum_{\Omega_{TF}} \mathbf{w}^T P_+ \mathbf{w}, \tag{15}$$

$$L_\times = \sum_{\Omega_{TF}} \mathbf{w}^T P_\times \mathbf{w}. \tag{16}$$

The matrices $P_+ = \mathbf{e}_+^T \mathbf{e}_+$ and $P_\times = \mathbf{e}_\times^T \mathbf{e}_\times$ are the orthogonal projections which kernel is a plane defined by the vectors $\mathbf{e}_+$ and $\mathbf{e}_\times$. The complementary space of the projections is the null space. The diagonal terms of the likelihood quadratic form ($i = j$) represent the incoherent energies E_+ and $E_\times$ detected in the network. Respectively the off-diagonal terms ($i \neq j$) represent the coherent energies C_+ and $C_\times$. One could naturally define a total correlated energy $E_c = C_+ + C_\times$ which is a measures of the correlation between the detector responses to a common GW signal. The correlated energy is used in the coherent WaveBurst post-production analysis to distinguish genium GW signals from the environmental and instrumental artifacts.

The definition of correlated energy E_c above is intuitively simple but not exactly correct. The problem is that the statistics C_+ and $C_\times$ (and also E_+ and $E_\times$) are correlated. Assuming that $\mathbf{w}$ is a vector of random variables, the covariance of two quadratic forms Λ_1 and Λ_2 is

$$cov[w^T \Lambda_1 w, w^T \Lambda_2 w] = 2tr[\Lambda_1 W \Lambda_2 W] + 4\omega^T \Lambda_1 W \Lambda_2 \omega, \tag{17}$$

where ω and W are the expected value and the variance-covariance matrix of $\mathbf{w}$ respectively. Assuming that uncorrelated detector noise is normally distributed with the unity variance (W is the unity matrix) and zero mean ($\omega = 0$) one can calculate the covariance of the likelihood quadratic forms:

$$cov[w^T C_+ w, w^T C_\times w] = -\sum_i e_{i+}^2 e_{i\times}^2, \tag{18}$$

$$cov[w^T E_+ w, w^T E_\times w] = \sum_i e_{i+}^2 e_{i\times}^2. \tag{19}$$

The problem is particularly obvious in the case of the two detector network where the coherent energies C_+ and $C_\times$ are perfectly anticorrelated and $C_+ + C_\times = 0$.

To solve this problem we use the fact that for a given kernel the projections P_+ and $P_\times$ are not uniquely defined. Indeed, any rotation

$$\tilde{\mathbf{e}}_+ = \mathbf{e}_+ \cos(\phi) + \mathbf{e}_\times \sin(\phi), \tag{20}$$

$$\tilde{\mathbf{e}}_\times = \mathbf{e}_\times \cos(\phi) - \mathbf{e}_+ \sin(\phi), \tag{21}$$

where ϕ is the rotation angle, defines the projections $P_+ = \tilde{\mathbf{e}}_+^T \tilde{\mathbf{e}}_+$ and $P_\times = \tilde{\mathbf{e}}_\times^T \tilde{\mathbf{e}}_\times$ with the same kernel. Note, the detector responses s_k and the total likelihod statistic $L_m = \tilde{L}_+ + \tilde{L}_\times$ are invariant with respect to the rotation. Given a data vector $\mathbf{w}$, such a rotation angle ϕ can be selected that $(\mathbf{w}, \tilde{\mathbf{e}}_\times) = 0$ and thus $\tilde{L}_\times = 0$. We define the incoherent and coherent energies as the diagonal and the off-diagonal terms of the *principle likelihood component* $\tilde{L}_+$.

2.1.3. Likelihood normalization The L_m is a positive definite quadratic form. In case of Gaussian stationary noise, when no GW signal is present, the expected value of the L_m is

$$E[L_m] = 1 + \frac{|\mathbf{f}|^2_\times}{|\mathbf{f}|^2_\times + \delta}. \tag{22}$$

This Equation shows that first, the L_m is a biased estimator of the total signal SNR detected in the network and second, the likelihood regulator breaks the normalization of the maximum likelihood ratio. The likelihood normalization is particularly important when a search is performed over a range of sky locations, because it affects the reconstruction of the source coordinates.

The likelihood normalization follows from the requirement (constraint) that the whiten detector responses

$$\mathbf{s} = \{s_1, .., s_K\} \tag{23}$$

should be orthogonal to the reconstructed detector noise

$$\sum_{\Omega_{TF}} \left[(\mathbf{w} \cdot \mathbf{s}) - |\mathbf{s}|^2\right] = 0. \tag{24}$$

To take the constraint into account, the canonical functional for the principle likelihood component should be modified as follows

$$\mathcal{L}_+ = \sum_{\Omega_{TF}} \left[2(\mathbf{w} \cdot \tilde{\mathbf{e}}_+)\tilde{g}_+ - \tilde{g}_+^2\right] + \lambda \sum_{\Omega_{TF}} \left[(\mathbf{w} \cdot \tilde{\mathbf{e}}_+)\tilde{g}_+ - \tilde{g}_+^2 \sum_{k=1}^{k=K} \tilde{e}_{+k}^2\right], \tag{25}$$

where λ is the Lagrange multiplier. When $\delta = 0$, the constraint is trivially satisfied for any λ. It corresponds to the local likelihood normalization when constraint is satisfied for every term in the sum Eq. **??**

$$(\mathbf{w} \cdot \tilde{\mathbf{e}}_+)\tilde{g}_+ - \tilde{g}_+^2 = 0\ . \tag{26}$$

In the presence of the regulator we can either force the local normalization

$$(\mathbf{w} \cdot \tilde{\mathbf{e}}_+)\tilde{g}_+ - \tilde{g}_+^2 \sum_{k=1}^{k=K} \tilde{e}_{+k}^2 = 0\ , \tag{27}$$

or solve the variation problem for the likelihood functional Eq.25 and obtain the solutions with the *global likelihood normalization.* Both likelihood normalization algorithms are implemented in the cWB pipeline. Currently the local normalization is used and the corresponding solutions for the GW waveforms in the principle component frame are

$$\tilde{g}_+ = \left(\sum_{k=1}^{k=K} \tilde{e}_{+k}^2\right)^{-1} (\mathbf{w} \cdot \tilde{\mathbf{e}}_+), \quad \tilde{g}_\times = 0. \tag{28}$$

3. Data analysis algorithms

In this section we describe the algorithms used in the coherent WaveBurst pipeline. The outline of the pipeline is shown in Figure 1. Many of the algorithms are used in the incoherent WaveBurst pipeline and they have already been described in [3, 4, 5],. But for completeness we briefly describe below all the algorithms used in the coherent pipeline.

3.1. Wavelet transformation

The discrete wavelet transformations (DWT) are applied to discrete data and produce discrete wavelet series w_{ij}, where j is the scale index (dilation) and i is a time index (translation). Applied to time series, the DWT maps data from time domain to wavelet domain. All DWTs implemented in WAT have *critical sampling* when the output data vector has the same size as the input data vector. It is different from Q-transformation [11] which can be viewed as the oversampled wavelet.

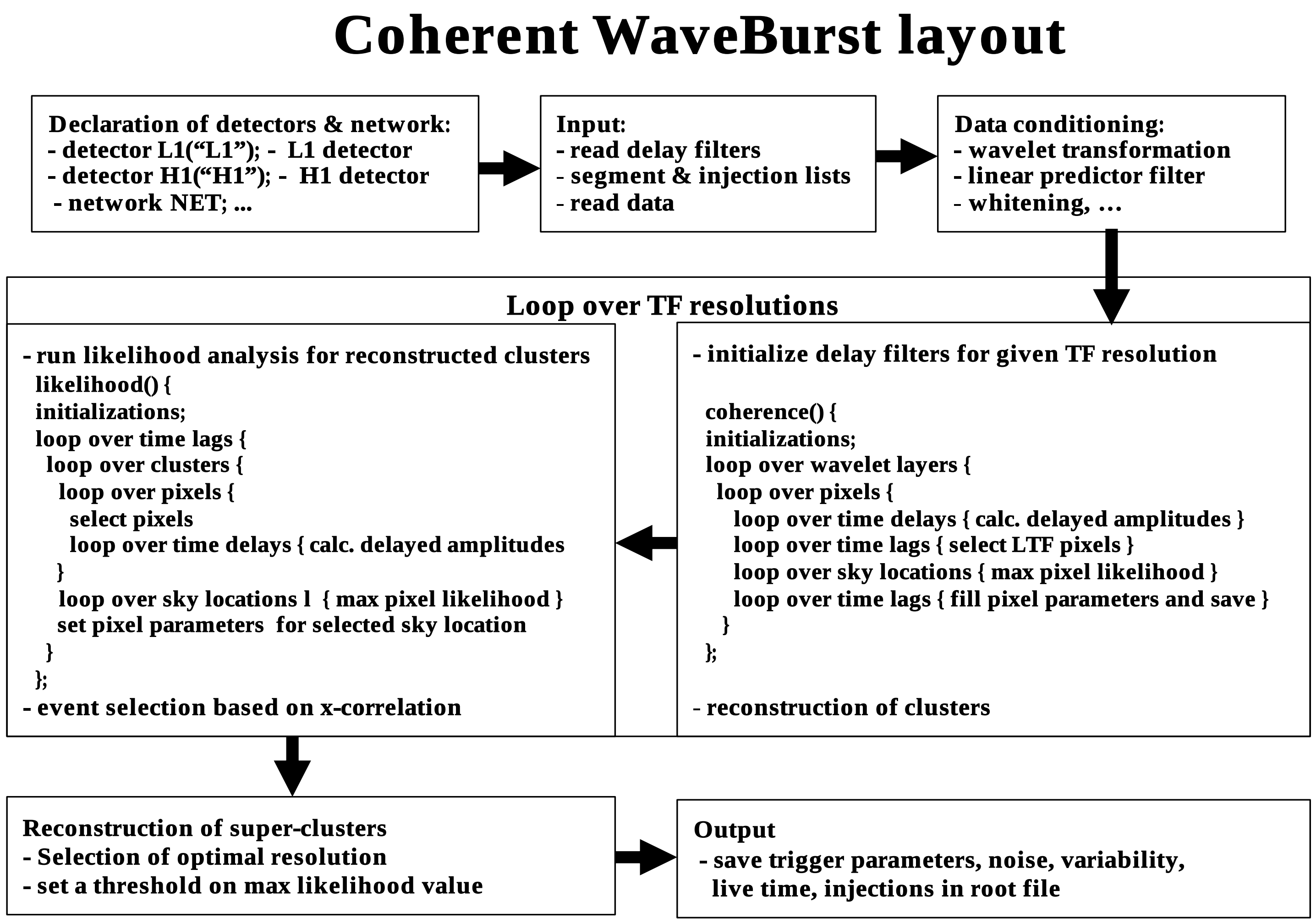
Wavelet series give a time-scale representation of data where each wavelet scale can be associated with a certain frequency band of the initial time series. Therefore the wavelet time-scale spectra can be displayed as a time-frequency scallogram, where the scale is replaced with the central frequency of the band. The initial time series sampling rate R and the scale number j determine the time resolution $\Delta t_j(R)$ at this scale. The DWT preserves the time-frequency volume of the data samples, which is equal to 1/2 for the input time series. Therefore the frequency resolution Δf_j is defined as $1/(2\Delta t_j)$ and determines the data bandwidth at the scale j.

The time-frequency resolution defined above should be distinguished from the intrinsic TF resolution of the wavelet transformation, which is determined by spectral leakage between wavelet sub-bands. In turn the spectral leakage depends on the length of the wavelet filter. To reduce spectral leakage we use Meyers wavelets where long filters can be easily calculated [5]. It allows much better localization of burst energy on time-frequency plane then for the Symlet60 wavelets used for the S2-S4 analysis [4]. Also more compact time delay filters (see Section 3.3) can be constructed in wavelet domain.

3.2. Data conditioning

Most of the data analysis algorithms are implemented as member functions in the *wavearray* and *WSeries* classes of the WAT library. This chapter describes the WaveBurst data conditioning steps, which are applied to data in the wavelet domain. We do not discuss possible data conditioning steps which could be applied to data in the time domain prior to wavelet decomposition.

3.2.1. Calibration Ideally the pipeline is designed to run on calibrated data or, so called $h(t)$ data produced with the time domain calibration [6]. It is possible to run


Coherent WaveBurst layout
Declaration of detectors & network:
- detector L1("L1"); - L1 detector
- detector H1("H1"); - H1 detector
- network NET; ...
Input:
- read delay filters
- segment & injection lists
- read data
Data conditioning:
- wavelet transformation
- linear predictor filter
- whitening, ...
Loop over TF resolutions
- initialize delay filters for given TF resolution
- reconstruction of clusters
- run likelihood analysis for reconstructed clusters
- event selection based on x-correlation
Reconstruction of super-clusters
- Selection of optimal resolution
- set a threshold on max likelihood value
Output
- save trigger parameters, noise, variability,
live time, injections in root file

```
coherence() {
initializations;
loop over wavelet layers {
  loop over pixels {
    loop over time delays { calc. delayed amplitudes }
    loop over time lags { select LTF pixels }
    loop over sky locations { max pixel likelihood }
    loop over time lags { fill pixel parameters and save }
  }
};
```

```
likelihood() {
initializations;
loop over time lags {
  loop over clusters {
    loop over pixels {
      select pixels
      loop over time delays { calc. delayed amplitudes
    }
    loop over sky locations l  { max pixel likelihood }
    set pixel parameters  for selected sky location
  }
};
```

Figure 1. Layout of the coherent Waveburst pipeline.

pipeline on raw data, but it may result in a degradation of the detection efficiency and reconstruction of the GW parameters, including incorrect reconstruction of the source coordinates. In this case for reconstruction of the strain amplitude of detected signals the standard WaveBurst calibration is used [3].

3.2.2. Linear prediction filter The linear prediction error filters are used to remove "predictable" components from an input time series. They have been used in LIGO data analysis in time domain. In this case the output of the LPE filter is a whitened time series. The LPE filters can be used also in wavelet domain. In this case individual LPE filters for each wavelet layer are constructed and applied. The coherent pipeline uses exactly the same LPE filters which have been used in the incoherent Waveburst pipeline [5].

3.2.3. Whitening Although, the linear predictor filters whiten data in the each wavelet layer, they preserve the colored structure of the detector noise. Therefore, to produce whitened wavelet series, the standard WaveBurst whitening procedure is applied [4, 5].

3.3. Time delay filters in wavelet domain

The likelihood method requires calculation of the scalar products of two detector data vectors $< x_n(t), x_m(t+\tau) >$, where one of the vectors can be shifted in time to take into account the GW signal time delay between the detectors n and m. The time delay τ in turn, depends on the source coordinates θ and ϕ.

In time domain it is straightforward to take the time delay τ into account. But in case of colored detector noise it is preferable to calculate the maximum likelihood statistics in the Fourier or wavelet (time-frequency) domains. It considerably simplifies the calculation of the network response matrix and the likelihood statistic. In case of the wavelet domain one needs to calculate the scalar products $< w_n(i,j), w_m(i,j,\tau) >$, where $w_n(i,j)$ are the wavelet amplitudes with the time-frequency indexes (i,j). If the m^{th} detector data stream $w_m(i,j)$ is delayed by time τ, it will result in a different distribution of the wavelet amplitudes on the TF plane and the TF location (i,j) acquires a new value of the amplitude $w_m(i,j,\tau)$. This amplitude can be calculated from the wavelet amplitudes $w_m(i,j)$ of the original data stream (before delay) with the help of the time delay filters $D_{kl}(\tau)$.

$$w_m(i,j,\tau) = \sum_{kl} D_{kl}(\tau,j) w_m(i+k, j+l). \tag{29}$$

where k and l are local TF coordinates with respect to the TF location (i,j). The delay filters are constructed individually for each wavelet layer, which is indicated with index j. If the inverse wavelet transformation is applied to the wavelet data $w_m(i,j)$ and $w_m(i,j,\tau)$, respectively the time series $x_m(t)$ and $x_m(t+\tau)$ can be obtained.

The construction of the delay filters is related to the decomposition of the wavelet functions $\Psi_j(t+\tau)$ in the basis of non-shifted wavelet functions $\Psi_j(t)$. The delay filter construction procedure can be described in the following steps:

- create blank wavelet series with only one coefficient at the TF location (i,j) set to unity,
- apply the inverse wavelet transformation, which will reconstruct a wavelet function $\Psi_j(t)$ in time domain,
- shift $\Psi_j(t)$ by delay time τ,
- do wavelet decomposition of $\Psi_j(t+\tau)$,
- the resulting wavelet amplitudes at TF locations $(i+k, j+l)$ give the delay filter coefficients $D_{kl}(\tau, j)$ for the wavelet layer j.

The length of the delay filter is determined from the requirement on the acceptable energy loss when the delay filter is applied. By sorting D^2_{kl} in the decreasing order the energy loss is

$$\epsilon_K = 1 - \sum_K D^2_{kl}, \tag{30}$$

where the sum is calculated over K most significant coefficients D_{kl}. The selected coefficients are described by the list of their relative TF locations (k, l) which should be stored along with the filter coefficients D_{kl}. Typically K should be greater then 20 to obtain the energy loss $\epsilon_K < 1\%$. Currently the delay filters with $K = 32$ are used in the analysis. Figure 2 shows results of the test of the delay filter. Given a time series $x(t)$, we produce a wavelet series $w(t, f)$. Then the delay filter with delay T=0.05sec is applied to $w(t, f)$ producing the delayed time series $w(t, f, T)$. After that the inverse wavelet transformation is applied to $w(t, f, T)$. The resulting time series $y(t, T)$ is delayed by -T in time domain and compared with the original time-series $x(t)$. The data $x(t)$ is a white Gaussian noise with unity variance. The residual signal has the variance of 0.013.

3.4. Generation of coherent triggers

A starting point of any burst analysis is the identification of burst events (triggers). Respectively this stage of a burst analysis pipeline is called the event trigger generator (ETG). So far the ETGs based on the excess power statistics of individual detectors were used in the analysis [1],[11]. Another example of ETG is corrPower [10], which uses cross-correlation between detector pairs to generate the triggers. The likelihood statistic utilizes both the excess power and the cross-correlation terms. Below we describe an event generator based on the likelihood statistic which we call the coherent WaveBurst trigger generator (cWB).

3.4.1. Likelihood time-frequency map In general the likelihood functional is calculated as a sum over the data samples selected for the analysis (see Eq.1). The number of terms in the sum depends on the selected TF area in the wavelet domain. In a particular case

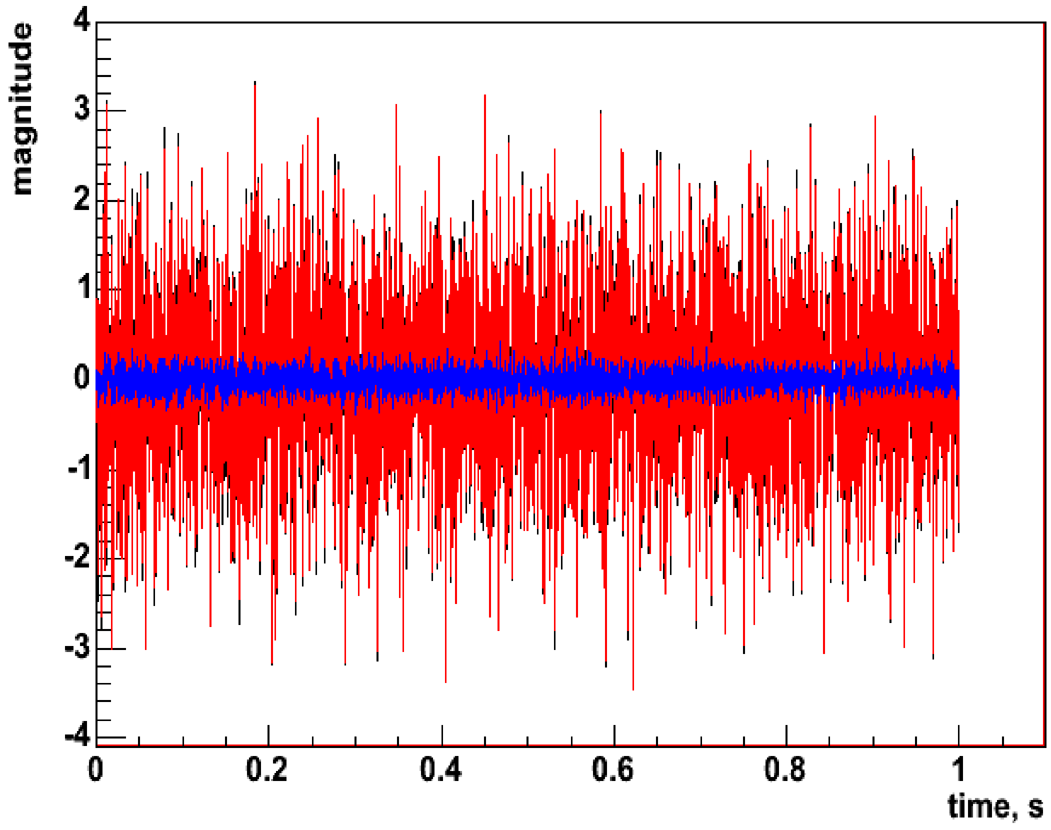


Figure 2. Black - the original time series $x(t)$, red - delayed time series $y(t)$, green - $x(t) - y(t)$

when the sum consists of only one term, one can write a likelihood functional for a single TF location (i,j):

$$\mathcal{L}_p(i,j,\theta,\phi) = \sum_k \frac{w_k^2(i,j,\tau_k)}{\sigma_k^2(i,j)} - \sum_k \frac{(w_k(i,j,\tau_k) - \xi_k(i,j,\theta,\phi))^2}{\sigma_k^2(i,j)}. \qquad (31)$$

Since the entire likelihood approach is applicable to the functional above, one can solve the variation problem and find the maximum likelihood statistic $L_m(i,j,\theta,\phi)$ for a given TF location (i,j). It can be maximized over the source coordinates θ and ϕ, resulting in a statistic

$$L_{mm}(i,j) = max_{\theta,\phi}\{L_m(i,j,\theta,\phi)\}. \qquad (32)$$

Plotted as a function of time and frequency, it gives us a likelihood time frequency (LTF) map. Figure 3 shows an example of the LTF map for the S4 data.

A single data sample of the map is called the LTF pixel. It is characterized by its TF location (i,j) and by the wavelet amplitudes $w_k(i,j,\tau(\theta,\phi))$, which are used to construct the likelihood $L_{mm}(i,j)$. The LTF pixel attributes are described in Section 5.2.

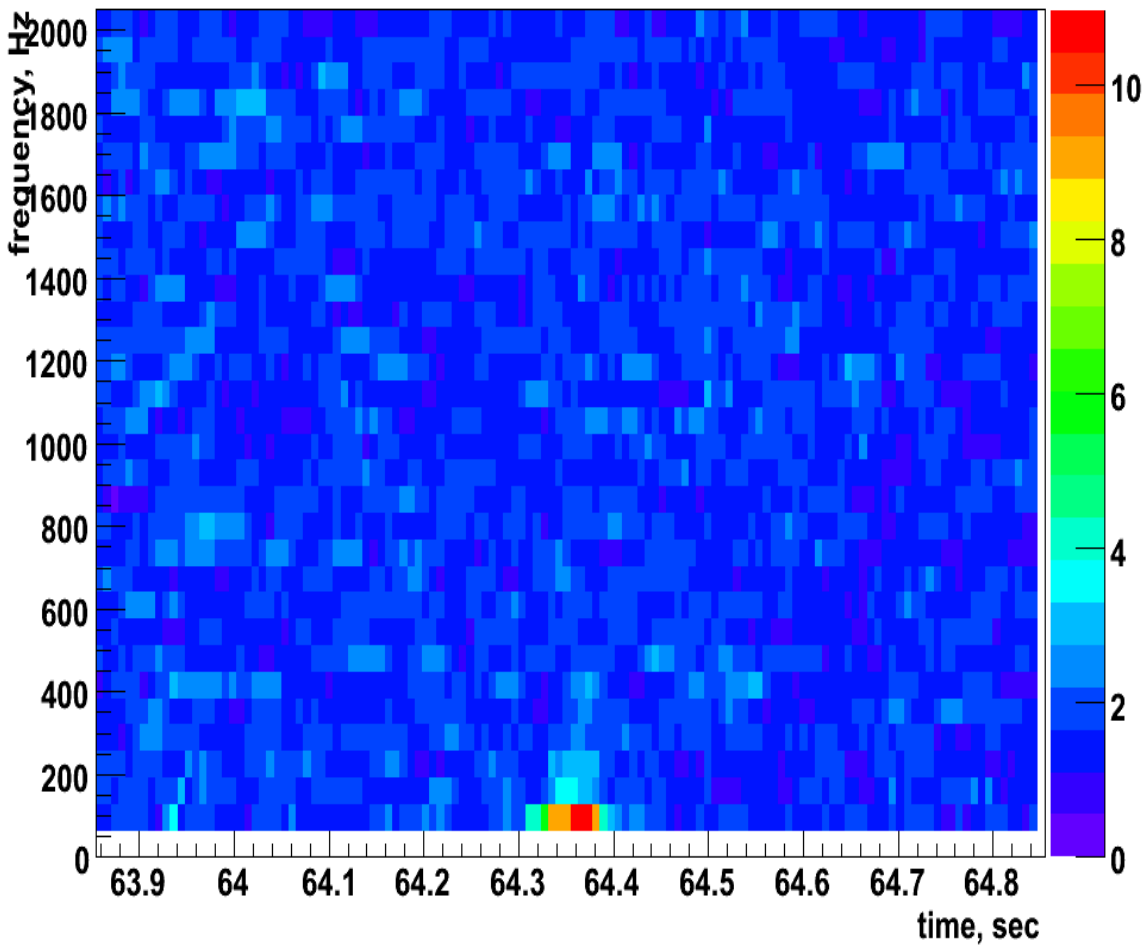


Figure 3. Example of the LTF map for a magnetic glitch in S4 L1xH1xH2 data.

3.4.2. Coherent triggers The statistic $L_{mm}(i,j)$ has a meaning of the maximum possible energy detected by the network at the TF location (i,j). By selecting the values of $L_{mm}(i,j)$ above some threshold, one can identify groups of the LTF pixels (coherent clusters) on the TF plane. The coherent clusters are reconstructed with the same clustering algorithm used in the incoherent WaveBurst pipeline [3, 4]. A coherent cluster is defined for the entire network, rather then for individual detectors. Therefore, further in the text we reserve a name "cluster" for a group of pixels selected in a single detector and refer to a group of the LTF pixels as a coherent trigger or network cluster.

3.4.3. Maximum likelihood statistic After the coherent triggers are identified, one has to reconstruct parameters of the GW burst associated with the triggers, including the reconstruction of two GW polarizations, the individual detector responses and the maximum likelihood statistic of the triggers. The likelihood of reconstructed triggers is calculated as

$$\mathcal{L}_c = \sum_{ij} \mathcal{L}_p(i,j,\theta,\phi) \tag{33}$$

where the sum is taken over the LTF pixels in the trigger. The maximum likelihood statistic L_{max} is obtained by variation of L_c over θ and ϕ. Unlike for L_p, which is calculated for a single LTF pixel, the L_{max} is calculated simultaneously for the LTF pixels forming the trigger.

However not all LTF pixels are used in the calculation of the maximum likelihood statistic, but only the *core* LTF pixels which satisfies the condition

$$\sum_k w_k^2(i,j,\tau(\theta,\phi)) > E_c. \tag{34}$$

The sum above is taken over all detectors in the network and the E_c is the threshold.

3.5. Multi-resolution analysis

Exactly as for the incoherent WaveBurst analysis the coherent pipeline is run at different time-frequency resolutions. The coherent triggers reconstructed at each resolution are combined into super-triggers. The optimal resolution is defined by selecting a trigger with the maximum likelihood. Also it is required in the algorithm that the triggers are reconstructed at least at two adjacent resolutions. Since the multi-resolution analysis algorithm is exactly the same as for the incoherent pipeline [4], we do not explain the algorithmic details in this note.

3.6. Time-shift analysis

For the coherent WaveBurst pipeline the time-shift analysis for estimation of the background is performed at run time. Namely, background triggers are generated by running pipeline on time-shifted data streams. This is the only available option to perform the time-shift analysis for a pipeline which uses maximum likelihood statistics for generation of triggers. Note, we distinguish time-shifts from time-delays, which take into account the GW signal delays $\tau(\theta,\phi)$ between different detectors.

Below, we describe the time-shift analysis on example of a network with three detectors. The time shifts are applied to the reference detector, which is the first detector declared in the network (see section 5.4). Namely, the reference detector data stream is shifted with respect to the other detector streams by a time interval $T_s = kt_s$ characterized by the time-shift step t_s and by the lag number k. The time-shift step is selected to be proportional to the time resolution Δt_j of the TF map:

$$T_s = m\Delta t_j, \tag{35}$$

where m is the integer. To take into account the time delays, the delay filters are applied to the second and third detectors. The LTF map at the TF location (i,j) and at zero lag is characterized by three wavelet amplitudes ($w_1(i,j)$,$w_2(i,j,\tau_{12})$,$w_3(i,j,\tau_{13})$). In parallel, one can also calculate non-zero lag LTF maps characterized by wavelet amplitudes ($w_1(i+mk,j)$,$w_2(i,j,\tau_{12})$,$w_3(i,j,\tau_{13})$), where k is the lag number. In case when the index $i+mk$ runs outside of the data segment selected for the analysis, we perform cycle time shifts starting from the beginning of the segment. The diagram below illustrates the time-shift algorithm.

det 1: ****$w_1(i+mk,j)$*******$w_1(i,j)$****$w_1(i+m,j)$****$w_1(i+2m,j)$.....***
det 2: **********************$w_2(i,j,\tau_{12})$***********************************
det 3: **********************$w_3(i,j,\tau_{13})$***********************************

The maximum time shift t_sK, where K is the total number of time lags defines the minimum duration of a data segment processed with the coherent WB pipeline.

The multiple time shifts described above are performed only for the first detector. However if other detectors have to be time-shifted, it is possible to introducing a constant time shift δT_k for each detector. In this case the zero lag triggers ($k=0$) correspond, in fact, to time-shifted triggers with time shifts δT_k between the detectors. The constant time shifts are set up during the declaration of the detector objects (see section 5.3) and they can not by modified during the cWB runs.

3.7. Calculation of the observation time

In this section we describe how the observation (live) time is calculated for zero and non-zero time lags. The live time is determined by a coincident lock segments of all detectors in the network and by the data quality flags. We distinguish between the primary data segments (science mode segments with some DQ flags included) which are used to generate a list of shorter segments in order to run the cWB jobs, and the final data segments with all DQ flags included. In the case when no final data quality flags are used in the analysis the live time is the same in each lag and it is equal to the total duration of the cWB data segments. However, when the final data quality flags are applied, the calculation of the live time can be quite non-trivial, particularly for non-zero time lags. To simplify this procedure we calculate the live time during the WaveBurst run time. Given a list of the data segments with the data quality flags included, we prepare a special time-frequency array ($veto[i,j]$) where pixels are set to be 0 or 1 if their time is outside or inside of the data quality intervals respectively. When the analysis of the LTF pixels is performed in the *coherence()* function, they are accepted (if $veto[i,j]=1$) or rejected (if $veto[i,j]=0$) and the total number of accepted LTF pixels is calculated for each time lag. The live time is calculated as the ratio of the accepted LTF pixels and the total number of data samples taken into the analysis multiplied by the duration of the data segment processed by the pipeline. The live time is stored in a root file together with the triggers produced by the cWB job. This procedure insures a correct calculation of the live time for any time shift algorithm, but it requires the final data quality segments to be known before the trigger production. At this moment we do not have any procedure for the post-production application of the final data quality flags. As soon as the data quality flags are finalized, we simply re-run the production of triggers with a new list of DQ segments.

4. Structure of the coherent pipeline

4.1. Input data

We read data from frame files with the ROOT script readframes.C [4] which uses FrameLib library. For example, in case of tree LIGO interferometers we read h(t) time series for L1, H1 and H2 interferometers. There is no specific data time stride. The time series can be as long as it is limited by the computer RAM memory. For example, for the analysis with three LIGO detectors and input time series of 1000sec long, typically 1Gb of RAM is used. However, for very non-stationary data it may increase to several Gbs. The minimum segment duration is defined by the number of time lags (see Section 3.6). We also require additional 8 seconds of data in the beginning and the end of the time series. This data is processed with wavelet transformations but not used for trigger generation to exclude the wavelet boundary artifacts. We prepare the list of the WaveBurst data processing segments by splitting the LIGO science segments into shorter segments, which is done outside of the cWB job. To avoid loss of data in the beginning and the end of the cWB segments, we read data with the 8 seconds overlap.

4.2. Trigger production

The coherent triggers are generated by running the cWB pipeline script in a ROOT session. The ROOT session is started in the following way:

```
cat string.in root -l -b -q parameters.C net.C &
```

where string.in contains a string with the information about the input and output directories, parameters.C is the WaveBurst configuration file and net.C is the pipeline script. The example of the configuration file and the production script for four detectors is show in the Appendix A and B respectively. The layout of the cWB pipeline which is implemented in the script is shown in Figure 1. In the beginning of the pipeline script the input string is parsed. Then, various WAT objects are initialized, including wavelet transformation, detectors, network, etc. For each specified detector, the data is read from frames, downsampled, transformed into wavelet domain and the data conditioning is applied.

The coherent search starts with the loop over the TF resolutions. Inside the loop the following algorithms are executed:

- *coherence3()* - calculation of LTF maps and pixel selection. For details see Section 5.5.
- *cluster(n,m)* - cluster reconstruction for all time lags. The input parameters define gaps between pixels in time (n) and frequency (m) in units of pixels.
- *likelihood3()* - calculation of maximum likelihood statistics. For details see Section 5.7.
- *corrcut(C,N)* - cut on reconstructed network correlation coefficient. The input parameters define the threshold on the network correlation (C) and the minimum

number of pixels (N) in the reconstructed clusters (the correlation cut is applied if the number of pixels in a cluster is greater than N).

- *setRank()* - calculation of rank statistics for reconstructed clusters.

Several calls of *likelihood3()* are executed with various parameters in order to get rid of large glitches by using a loose cut on reconstructed network correlation coefficient. The rank statistics is calculated and the selected pixels are stored in the buffers WC[j] individually for each lag j. For the next loop iteration the selected pixels are appended to the WC structures, so in the end of the loop each WC[j] contains a list of pixels detected at different TF resolutions.

After the loop over the TF resolutions, the pixel lists are copied into the corresponding structures in the detector network NET object and the super-clusters are reconstructed. In the end of the script various data objects are output into the root files as described in the next section.

4.3. Output data

The ROOT data format allows to store arbitrary data divided into output data streams. Each data stream is stored in the ROOT structure called a data tree. For WaveBurst runs we distinguish five data streams:

- waveburst - coherent WaveBurst triggers,
- noise - noise rms for each wavelet layer,
- variability - noise variability for all detectors,
- livetime - live time estimated for each time lag
- injection - list of software injections

Respectively there can be up to five data trees in each ROOT file with one file generated per WaveBurst job. The corresponding tree names are: “waveburst”, “noise”, “variability”, “livetime” and “injection”. The structure of the waveburst tree is shown in the Appendix C.

5. Pipeline Implementation in WAT

Wavelet Analysis Tool is a C/C++ library which serves as a foundation for several DMT monitors: WaveMon, BurstMon and LineMonitor, and for the WaveBurst analysis pipelines. The WAT toolbox consists of several data containers:

- wavearray< T > - generic template array. It is used in WAT for representation of linear arrays and time series.
- WSeries< T > - generic template sliced array. It is used in WAT to store sliced arrays and wavelet data.
- skymap - class for representation of a rectangular patch on the sky (or entire sky). It is used in WAT to store various parameters, like detector antenna patterns, as a function of polar angles θ and ϕ.

- netcluster - class for the time-frequency pattern recognition. It is used in WAT to store patterns of wavelet pixels (coherent triggers) for the coherent burst analysis.
- detector - class describing a GW detector. It has methods to calculate and store skymaps of the detector antenna patterns, WSeries array to store the detector data in time and wavelet domains, arrays to store reconstructed detector responses, etc.
- network - class describing a network of detectors. Most of the coherent algorithms for calculation of the LTF maps and generation of coherent triggers are implemented here. Apart from the initialization and data access utilities, there are two main functions in the network class: coherence() and likelihood(). The coherence() function calculates LTF maps for specified time lags and create lists of selected (black) LTF pixels (see Sections 3.4.1,3.4.2). The likelihood() function calculates the likelihood of reconstructed triggers (see Section 3.4.3).

There are several wavelet transformations:

- Symlet$< T >$ - Symlet template class.
- Daubechies$< T >$ - Daubechie template class.
- Biorthogonal$< T >$ - Bi-orthogonal template class.
- Meyer$< T >$ - Mayer template class.
- Haar$< T >$ - Haar template class.

Currently the Meyer wavelet with 1024 filter coefficients is used in the coherent pipeline.

The data processing algorithms used in WaveBurst are implemented as member functions in the data container classes mentioned above.

5.1. Sky maps

Sky maps are two dimensional arrays representing some quantity as a function of polar angles θ and ϕ. Sky maps are used to store antenna patterns, time delays, network likelihood, etc. The sky maps are implemented in the *skymap* C++ class. The number of sky locations is defined by the angular resolution of the search. For example, for all sky search, with one degree resolution the search is performed over 356×180 sky locations.

5.2. Netcluster

The *netcluster* class defines the structure of the coherent triggers and methods for the TF pattern recognition. Essentially the *netcluster* object is a list of the LTF pixels, which have the following structure:

```
{
  size_t clusterID;                // cluster ID
  size_t time;                     // time index for reference detector
  size_t frequency;                // frequency index (layer)
  float  rate;                   // wavelet layer rate
  float  likelihood;             // likelihood
  float  theta;                  // source angle theta index
  float  phi;                    // source angle phi index
  bool   core;                   // pixel type: true - core , false - halo

  std::vector<double> wave;        // vector of pixel's wavelet amplitudes
  std::vector<double> asnr;        // vector of pixel's whitened amplitudes
  std::vector<float>  rank;        // vector of pixel's rank amplitudes
  std::vector<double> noiserms;    // average noise rms
  std::vector<float>  variability; // average noise variability
  std::vector<int>    neighbors;   // vector of links to neighbors
  std::vector<int>    index;       // pointers to pixels in the TF data
}
```

Usually the coherent WB pipeline is run at several time-frequency resolutions. Therefore, multiple triggers can be reconstructed for the same burst. The collection of such triggers is called a super-cluster. The algorithm of super-cluster reconstruction is exactly the same as for the incoherent WB [4]. The only differences are: a) for the incoherent WB super-clusters are reconstructed for each detector, b) for the coherent WB the optimal trigger is defined as a trigger with maximum value of likelihood.

The parameters of the coherent triggers can be extracted from the lists of the LTF pixels with the method *get()* defined in the netcluster class.

5.3. Detector

A detector object is described in the *detector* C++ class. A detector is characterized by its position and orientation of interferometer arms with respect to Earth centered coordinate frame [16]. The positions and orientations of existing detectors (LIGO L1, LIGO H1, LIGO H2, GEO G1, TAMA T1 and Virgo V1) are taken from [17]. For example, a declaration

```
detector L1(``L1'');
```

defines the object L1 for the LIGO Livingston "L1" detector.

The detector antenna patterns are calculated according to [17] and they are in agreement with results from other existing packages [18]. The detector object stores the data vector (TFmap) and reconstructed detector responses.

5.4. Network

A network object is described in the *network* C++ class. Essentially it is a list of detectors. Detectors can be added to the network with the *add(detector*)* method. The first detector in the list is the reference detector. Its output can be artificially shifted in time to perform the time-shift background analysis (see section 3.6). To synchronize GW signals between the detectors the second, third, etc, detectors are delayed with respect to the reference detector by means of the delay filters. The coherent algorithms are implemented in two network methods: a) *coherence* and b) *likelihood*. Both methods are very CPU consuming, therefore to optimize the pipeline performance they are implemented separately for networks with 3 and 4 detectors. Currently there is no implementation for networks consisting of 2, 5 and more detector and they can be easily implemented later if necessary. Below we describe the implementation of the coherence and the likelihood methods for 3 detectors.

5.5. Implementation of the coherence method

The coherence3(a_E,a_C) function takes two parameters: a_E is the threshold on the average amplitude a_p of the LTF pixels and a_C is the threshold for selection of loud (core) pixels. The amplitude a_p is calculated as

$$a_p = \sqrt{2L_{mm}/K}, \tag{36}$$

where L_{mm} is the likelihood of the LTF pixel (see Eq. 32) and K is the number of detectors in the network. The parameter a_E is the main WaveBurst threshold which defines the pipeline output rate at the production stage. It is set individually for each wavelet resolution to yield approximately the same number of the LTF pixels selected at each resolution (see Section 5.6).

The threshold a_C does not affect the pipeline sensitivity. It is used for better handling of loud glitches which are present in real data. Such glitches may consist of thousands of pixels and they may overflow the computing resources at the final stages of the trigger production. To make the pipeline usable for processing of real data we eliminate such glitches at the early stage of the analysis by using a loose cut (see description of the corrcut() function in Section 5.8) on the network correlation coefficient, which is introduced in Section 6.1.1.

The algorithm of the coherence function is outlined below

```
coherence3($a_L$,$a_E$){
initializations;
loop over wavelet layers $j$ {
    loop over pixels $i$ {
        loop over time delays $\tau$ {
            calculate amplitudes $w_2(i,j,\tau)$, $w_3(i,j,\tau)$;
        }
        loop over time lags $k$ {
            estimate maximum pixel energy:
            $E(i,j,k) = w_1^2(i,j,k) + max_\tau(w_2^2(i,j,\tau)) + max_\tau(w_3^2(i,j,\tau))$;
            if($E(i,j,k) > 3a_L^2$) select LTF pixels:
        }
        loop over sky locations $l$ {
            estimate maximum pixel energy for detectors 2 and 3:
            $E_{23}(i,j) = max_l(a_2^2(i,j,l) + a_3^2(i,j,l))$;
            where $w_2^2(i,j,\tau))$ and $w_3^2(i,j,\tau)$ are mapped into
            $a_2^2(i,j,l)$ and $a_3^2(i,j,l)$ with the help of skymaps $\tau_{12}[l]$ and $\tau_{13}[l]$.
        }
        loop over time lags $k$ {
            estimate maximum likelihood as
            $L_{mm}(i,j,k) = (w_1^2(i,j,k) + E_{23}(i,j))/2$
            fill pixel parameters;
            if($L_{mm}(i,j,k) > 3a_E^2/2$) mark core pixels: ;
            save selected pixels;
        }
    }
}
```

The coherence function allows calculation of the LTF maps defined by Eq.32, but to speed up the algorithm the LTF maps are approximated with the maximum pixel energy $L_{mm}(i,j) \approx E(i,j)/2$. The $E(i,j)/2$ gives the upper limit on the likelihood $L_{mm}(i,j)$, which follows from the likelihood equation $2L = E - N$, where E is the total energy in the data streams and N is the total detector noise energy (see section **??**).

The final result of the coherence function are lists of pixels selected for zero and non-zero time lags. This is similar to selection of *black pixels* used in the incoherent methods [3], however, of course, the selection procedure is very different. The selected LTF pixels are clustered with the existing pattern recognition methods [3, 4] and the coherent triggers are reconstructed.

5.6. Calculation of the WaveBurst threshold

Given the probability p that the LTF pixel amplitude $a_p > a_E$ (produced by Gaussian detector noise), the threshold $a_E(p)$ depends on the wavelet TF resolution. To calculate the function $a_E(p)$ one needs to know the probability distribution function $PDF(E(i,j,k))$ of the maximum pixel energy $E(i,j,k)$ (see previous Section). Though, the $PDF(E(i,j,k))$ can be estimated from numerical simulation, it is desirable to have its analytic approximation. We did not find an accurate analytic expression for $a_E(p)$. We found, however, a reasonable approximation which can be derived from the $PDF(max(|x_1|,|x_2|,..,|x_n|))$ of n random Gaussian variables $(x_1,x_2,..,x_n)$. Given the probability p that $max(|x_1|,|x_2|,..,|x_n|) > X(q,n)$, the threshold $X(p,n)$ is calculated as

$$X(p,n) = \sqrt{2}Erfc^{-1}\left(1 - \exp\left(\frac{ln(1-p)}{n}\right)\right), \tag{37}$$

where $Erfc^{-1}$ is the inverse complementary error function.

A reasonably accurate approximation (within a few percents if $p > 0.01$) for the threshold a_E is calculated as

$$a_E = \sqrt{X(p, n_{eff}) + \frac{2}{K}\left[X(p,1) - X(p, n_{eff})\right]}, \tag{38}$$

where the parameter n_{eff} depends on the maximum time delay between the detectors τ_{max} and the TF sampling interval Δ_T

$$n_{eff} = 2\frac{\tau_{max}}{\Delta_T}. \tag{39}$$

The probability p (and a_E) defines the number of the LTF pixels taken into the analysis. The all-sky WaveBurst pipeline is usually limited by available computing resources if the threshold is set low. From the other side the threshold should not be too low if the final sensitisitivity of the pipeline is dominated by the post-production selection cuts (see Section 6). For all-sky searches with the LIGO and the LIGO-GEO networks, the typical value of p is 0.001.

5.7. Implementation of the likelihood method

The likelihood3(*mode*,*core*,a_C) function takes three parameters: *mode* is the character parameter, which defines a mode in which the likelihood is calculated, *core* is a logical parameter which defines if only core (*coretrue*) or all pixels (*core*=*false*) are taken into the analysis, and a_C is the threshold for selection of core pixels. The algorithm of the likelihood function is outlined below:

```
likelihood3(mode,core, $a_c$){
initializations;
 loop over time lags $k$ {
    loop over coherent triggers $n$ {
        loop over pixels in a trigger {
            pixel selection;
            loop over time delays $\tau$ {
                calculate amplitudes $w_2(i,j,\tau)$,$w_3(i,j,\tau)$;
            }
        }
        initialize maximum likelihood $L_{max} = 0$;
        loop over sky locations $l$ {
            initialize likelihood $L_l = 0$;
            loop over pixels in a trigger $m$ {
                select pixels;
                calculate pixel likelihood $L(l,m)$;
                $L_l + = L(l,m)$;
            }
            if($L_{max} < L_l$) $L_{max} = L_l$;
        save index $l_{max}$ corresponding to $L_{max}$;
      }
      for selected sky location $l_{max}$
      loop over pixels in the trigger {
        calculate amplitudes $w_2(i,j,l_{max})$,$w_3(i,j,l_{max})$;
        reconstruct GW signal: - fill vector wave;
        save pixel parameters;
      }
  }
}
```

The likelihood function performs full reconstruction of the coherent triggers and fills in all parameters of the LTF pixels, including reconstructed likelihood, null energy, source coordinates, waveforms, etc.

The reconstruction of coherent triggers is controlled by the regulator parameter δ (Eq.12) and by the input parameter "mode", which can take the following values:

- 'l' - the soft constraint is executed ($\delta = 1$). The variation over the sky is performed to search for the maximum value of the likelihood ratio.
- 'L' - the constraint defined by the parameter δ is executed. The variation over the sky is performed to search for the maximum value of the likelihood ratio.
- 'c' - the soft constraint is executed ($\delta = 1$). The variation over the sky is performed to search for the maximum value of the network correlation coefficient(see section 6.1.1).
- 'C' - the constraint defined by the parameter δ is executed. The variation over the sky is performed to search for the maximum value of the network correlation coefficient.

What mode for the reconstruction of coherent triggers is determined by the network configuration and the stationarity of the detector noise.

5.8. Rejection of loud glitches

6. Post-production analysis

When the detector noise is Gaussian and stationary, the maximum likelihood L_m is the only statistic required for detection and selection of the GW burst triggers. In this case no post-production analysis is required and the pipeline false alarm and false dismissal probabilities are controlled by the threshold on L_m at the output of the coherent event generator. In reality, the detector outputs are contaminated with glitches and additional selection cuts should be applied to distinguish glitches from genuine GW signals. This selection cuts are entirely ad hoc and depend on the network configuration and parameters of the search. Therefore we divide the pipeline into two stages: a) coherent event generator (trigger production stage) and b) post-production analysis stage. In the cWB pipeline the coherent analysis is performed at the production stage and all the coherent statistics such as likelihood, null stream, network data matrix, etc, are stored in the output trigger files. The cWB post-production stage deals with selection of the "optimal" set of statistics and cuts to reject glitches and does not require to run the coherent algorithms.

6.1. Event consistency

Empirically we found several selection cuts, which work best for the S4/S5 LIGO, GEO and Virgo data. They are divided in two groups: a) the event consistency cuts (coherent WaveBurst veto), which are used to reject glitches and typically have little or no effect on the detection efficiency, and b) the final selection cut based on the strength of detected events (see Section **??**).

6.1.1. Network correlation coefficient We can rewrite Eq.1 in the following form

$$L_m = E - N, \tag{40}$$

where E is the total normalized energy in the detector output streams

$$E = \sum_{\Omega_{TF}} |\mathbf{w}|^2 \tag{41}$$

and N is the total reconstructed energy of the noise in units of the noise RMS

$$N = \sum_{\Omega_{TF}} |\mathbf{w} - \mathbf{s}|^2. \tag{42}$$

Often N is called a null stream, where the reconstructed GW signal is subtracted from the detector output. From Eq.40 it is clear that L_m is simply the total SNR of a GW signal detected in the network.

The likelihood is a quadratic form. The sum of the off-diagonal terms is the coherent energy E_c (see Eq. 2.1.2). The coherent terms can be used to construct Pearson's, Cauchy's correlation coefficients and the network correlation coefficient.

We define the network correlation coefficient as

$$cc = \frac{E_c}{N + |E_c|}, \tag{43}$$

The network correlation coefficient is used for the post-production selection of the coherent triggers. For glitches typically little correlation energy is detected by the network and the reconstructed detector responses are inconsistent with the detector outputs which result in the large null energy. By setting a threshold on the correlation coefficient cc one effectively compares the null energy with the coherent energy. This is much safer selection cut than the null stream cut [19] (where the null energy is compared with the estimated noise energy), because in any realistic data analysis there is always some residual energy left in the null stream. For strong gravitational waves the energy of the residual signal can be much larger than the noise energy resulting in the false rejection of the GW signal.

We define Pearson's correlation coefficient for each pair of detectors

$$r_{ij} = \frac{L_{ij}}{\sqrt{L_{ii}L_{jj}}}, \tag{44}$$

where L_{ij} are the elements of the matrix for the principle likelihood component. The correlation coefficients r_{ij} are used to construct the *reduced correlated energy*

$$e_c = \sum_{i \neq j} L_{ij}|r_{ij}|. \tag{45}$$

Together with the correlated energy E_c it may be used to construct the cWB post-production cuts.

6.1.2. Penalty factor The constraint Eq.24 is not the only constraint that used in the analysis. The signal-noise orthogonality requirement should be also applied separately for each detector

$$\Lambda_k = \sum_{ij} \Big(w_k[i,j] s_k[i,j] - s_k^2[i,j] \Big) = 0. \tag{46}$$

The reason for this is to prevent the reconstruction of the un-physical detector responses when the energy of the response is greater than the total energy in the detector data stream. Namely, the constraints above (Eq.46) ensure that

$$E_k > S_k; \quad E_k = \sum_{ij} w_k^2[i,j], \quad S_k = \sum_{ij} s_k^2[i,j]. \tag{47}$$

These constraints are particularly important for all sky searches to find a correct source location and help to reduce the false alarm rate.

The constraints (Eq.46) can be applied during the likelihood variation procedure with the help of the Lagrange multipliers. However, due to computational complexity we use them in the form of a penalty factor, by penalizing those points in the sky where the constraint is not satisfied

$$P_f = \max_k P_k; \quad P_k = \sqrt{\frac{E_k}{S_k}}, \; E_k < S_k; \quad P_k = 1, \; E_k > S_k. \tag{48}$$

Also, the penalty factor is used for rejection of the background events. In the post-production we require that $P_f > 0.6$.

6.1.3. H1H2 penalty factor For better rejection of the H1H2 correlated glitches we introduce a penalty factor designed specifically for the H1 and H2 detectors

$$P_{hh} = \frac{|\Lambda_{H1} - \Lambda_{H2}|}{E_c}. \tag{49}$$

Typically, correlated glitches are characterized by a large value of P_{hh} and the GW signals have P_{hh} close to zero. In the analysis we require that $P_{hh} < 0.3$ which rejects significant number of correlated H1H2 glitches due to magnetic disturbances.

6.2. Effective correlated SNR

There are several detection statistics reconstructed by coherent WaveBurst: likelihood, coherent and incoherent energies, total SNR of the detector data streams (Eq.41), rank statistics, etc. Empirically we found that the most effective detection statistic is based on the correlated energy E_c (or e_c). It is used in combination with the null energy to construct the network correlation coefficient (see Eq.43) and the *effective correlated SNR* per detector

$$\rho(E_c) = cc^2 \frac{E_c}{K}, \tag{50}$$

$$\rho(e_c) = cc \frac{e_c}{K}. \tag{51}$$

Figure 4 shows a scatter plot of $\rho(E_c)$ vs cc. Glitches (black dots) with large value of ρ typically have low value of the correlation coefficient and they can be removed by the cWB veto cuts. The background triggers with large value of cc are usually produced by fluctuations of the detector noise and they can be effectively removed by setting a threshold on ρ.

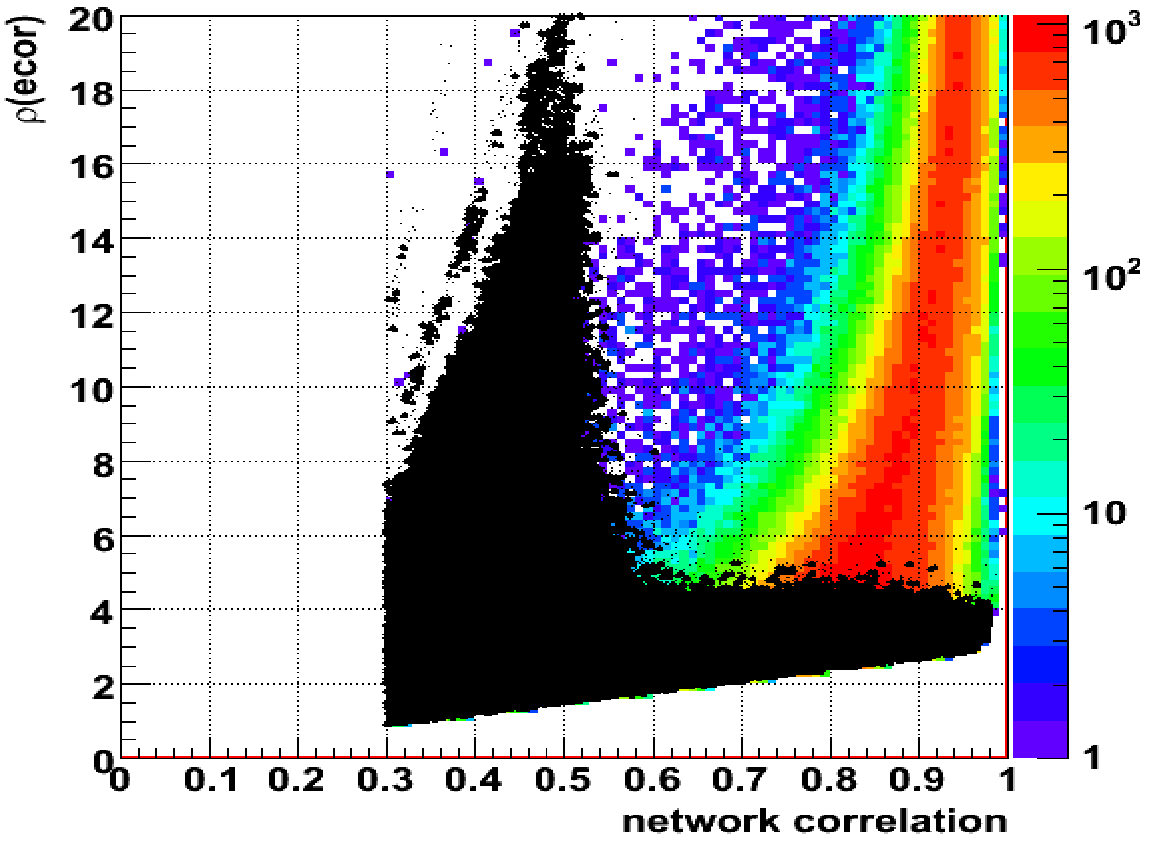


Figure 4. Scatter plot $\sqrt{\rho}$ vs network correlation coefficient cc: black dots - background triggers, colored distribution - sine-Gaussian injections, Q9.

7. Estimation of burst parameters

Coherent triggers are reconstructed for the same time-frequency area in the wavelet domain. Unlike for incoherent waveburst clusters in different detectors consist of the same number of pixels. Therefore there are two groups of the trigger parameters: a) reconstructed for the entire coherent event (likelihood, source coordinates, etc) and b) reconstructed individually for clusters in each detector (time, h_{rss}, etc).

7.1. Time and frequency

There are the following timing parameters calculated for a coherent event:

- start and stop GPS time,
- event duration,
- event central GPS time,
- low and high frequency,
- event bandwidth,
- event central frequency.

The *start* and *stop* are calculated from the coherent event boundaries in time defined by the core pixels. For non-zero lag events the *start* and *stop* time can be different for different detectors. For example, the difference $start(H1) - start(L1)$ gives the time shift between the H1 and L1 detectors. The event central time is calculated as

$$T_n = \frac{\sum_{i=1}^{k} t_i * L_{max}[i]}{\sum_{i=1}^{k} L_{max}[i]} + \tau_n(\theta, \phi), \tag{52}$$

where t_i is the GPS time of core pixels, $L_{max}[i]$ is the pixel's maximum likelihood ratio and k is the number of core pixels in the cluster. The term $\tau_n(\theta, \phi)$ accounts for the time delay between the n-th detector and the first (reference) detector in the network, which is calculated for reconstructed sky coordinates θ and ϕ. The *low* and *high* parameters are calculated from the cluster boundaries in frequency. The central frequency is

$$F_c = \frac{\sum_{i=1}^{k} f_i * L_{max}[i]}{\sum_{i=1}^{k} L_{max}[i]}, \tag{53}$$

where f_i is the frequency of the core pixels. The trigger *duration* and *bandwidth* are calculated as $stop - start$ and $high - low$ respectively.

7.2. Burst strength

There are several measures of the burst strength:

- normalized energy of the detector streams integrated over the TF area of the event (see Eq.41 and Eq.**??**),
- maximum likelihood ratio L_{max}, which is proportional to the total signal SNR detected in the network,

- reconstructed energy in individual detectors $2L_k$ (see Eq.**??**),
- root-square-sum of the detector responses (h_{rss}).

7.2.1. Calibrated hrss The analysis uses calibrated $h(t)$ data, which takes into account the time-dependent calibration of the LIGO detectors. For a sampled signal $h(t_i)$ the signal h_{rss} is defined as

$$h_{rss} = \sqrt{\sum_i \frac{h^2(t_i)}{f_s}}, \tag{54}$$

where f_s is the sampling rate. For the orthogonal wavelet transformation it holds that

$$\sum_i h^2(t_i) = \sum_{i,j} w_h^2(t_i, f_j) \tag{55}$$

where the second sum is taken over the time-frequency plane in the wavelet domain and w_h is the wavelet transform of the signal h.

The coherent WaveBurst reconstructs the GW polarizations h_+ and $h_\times$ in the wavelet domain. Then given a cluster, the gravitational wave h^2_{rss} can be estimated as

$$h^2_{rss} = \sum_{i,j} \frac{h^2_+[ij] + h^2_\times[ij]}{f_s}, \tag{56}$$

where the sum is taken over all pixels in the cluster. The root-square-sum of the detector responses is calculated accordingly

$$\xi^2_{rss} = \sum_{i,j} \frac{\xi^2[ij]}{f_s}, \tag{57}$$

where ξ is given by Equation 2. The inverse wavelet transformation of ξ gives the reconstructed detector responses in time domain.

8. Appendix A: WaveBurst configuration file for four detectors.

```
{
  int simulation = 1;    //1 for simulation, 0 for production
  int runID    = 0;      // run number

  // WB threshold settings

  double bpp   = 0.001;  // probability for pixel selection
  double Acore = 1.67;   // threshold for selection of core pixels
  double Tgap  = 0.03;   // time gap between clusters (sec)
  double Fgap  = 64.;    // frequency gap between clusters (Hz)

  // wavelet transform settings

  int levelR   = 2;      // resampling level
  int levelD   = 8;      // decomposition level
  int l_low    = 3;      // low frequency resolution level
  int l_high   = 8;      // high frequency resolution level
  int lpfcut   =64;      // low pass filter cut-off [Hz]
  int w_mode   = 1;      // whitening mode
  double waveoffset = 8.;// wavelet boundary offset [sec]
  bool parity = true;    // use odd-parity wavelet

  // time shift analysis settings

  int lags     = 1;      // number of lags including zero lag
  double step=3.25;      // time interval between lags [sec]
                         //  L1  H1  H2  G1 constant time shifts
  double shift[4] =         {0., 0., 0., 11.125};

  // logNormal parameters

  double pln[27]={2.700,0.254,0.200,   /* level 1 */
  2.500,0.274,0.200,   /* level 2 */
  2.304,0.298,0.171,   /* level 3 */
  2.144,0.322,0.137,   /* level 4 */
  1.965,0.336,0.145,   /* level 5 */
  1.772,0.356,0.142,   /* level 6 */
  1.587,0.371,0.146,   /* level 7 */
  1.430,0.380,0.156};  /* level 8 */
  // simulation parameters

  int   nfactor = 21;
  double factor = 1.0;
  double factors[] ={0.106, 0.150, 0.211, 0.298, 0.422,
     0.596, 0.841, 1.19, 1.68, 2.37, 3.35,
     4.73, 6.68, 9.44, 13.3, 18.736, 26.395,
     37.184, 52.383, 73.794, 103.957};

  // directories

  char data_dir[]  = "/usr1/igor/waveburst/data64";
  char nodedir[50] = "/usr1/igor/waveburst";
  char finalSegmentList[100]="1146592.542-S4H1H2L1G1_final_DQ.A.txt.plain";
  char injectionList[50]="BurstMDC-SG21_V4_S4-Log.txt";

  char fileNamesRaw[4][50]={"llo.lst.","lho1.lst.",
    "lho2.lst.","geo.lst."};

  char channelNamesRaw[4][50]={"L1:LSC-STRAIN","H1:LSC-STRAIN",
      "H2:LSC-STRAIN","G1:DER_DATA_H"};

  char channelNamesMDC[4][50]={"L1:GW-H","H1:GW-H",
      "H2:GW-H","G1:GW-H"};
}
```

9. Appendix B: WaveBurst script for L1xH1xH2.

```
{
  int  i,j,m;
  char input_dir[512],input_label[512];
  char output_dir[512], output_label[512];

// parse input

  cin>>runID>>input_dir>>input_label>>output_dir>>output_label;
  cout<<"job ID: "<<runID<<endl;
  cout<<"Input : "<<input_dir<<",  label: "<<input_label<<endl;
  cout<<"Output: "<<output_dir<<",  label: "<<output_label<<endl;
  gSystem->Exec("date");
  gSystem->Exec("hostname");
```

```
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// declarations
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  Meyer<double> B(1024);       // set wavelet for resampling
  Meyer<double> S(1024,2);     // set wavelet for production

  netcluster WC[lags];         // array of cluster structures
  netcluster wc;
  netcluster* pwc;

  wavearray<double> x,y;       // temporary time series
  WSeries<double> wB(B);       // original WSeries
  WSeries<float>   v[4];       // noise variability

  detector        L1("L1");    // detector
  detector        H1("H1");    // detector
  detector        H2("H2");    // detector
  detector        G1("G1");    // detector
  network         NET;         // network

  NET.add(&L1);
  NET.add(&H1);
  NET.add(&H2);
  NET.add(&G1);
  NET.setSkyMaps(1.);          // set network skymaps
  NET.setIndex(&H1);
  NET.setAntenna(&L1);
  NET.setAntenna(&H1);
  NET.setAntenna(&H2);
  NET.setAntenna(&G1);
  NET.setbpp(0.1);
  NET.setRunID(runID);
  NET.Edge = waveoffset;
  NET.pLLL = 0.1;

  injection mdc(4);
  livetime live;
  netevent netburst(4);
  variability wavevar;
  wavenoise noiserms;
  // read input framelist file

  char file[512], buFFer[1024];

  sprintf(file,"%s/%s%d",input_dir,fileNamesRaw[0],runID);
  FILE *pFile = fopen(file,"r");
  fgets(buFFer,512,pFile);
  fgets(buFFer,512,pFile);
  fgets(buFFer,512,pFile);
  double Tb = atof(buFFer)+waveoffset; // WB segment start
  fgets(buFFer,512,pFile);
  double Te = atof(buFFer)-waveoffset; // WB segment stop
  double dT = Te-Tb;                   // WB segment duration
  fclose(pFile);

  if(simulation) {         // reag MDC log file
    i=NET.readMDClog(injectionList,double(long(Tb)));
    printf("GPS: %16.6f saved,  injections: %d\n",double(long(Tb)),i);
  }
  else {                   // setup constant shifts
    L1.shift(shift[0]);
    H1.shift(shift[1]);
    H2.shift(shift[2]);
    G1.shift(shift[3]);
  }

  NET.setTimeShifts(lags,step);
  if(strlen(finalSegmentList)>1) NET.readSEGlist(finalSegmentList,2);

// read delay filters

  detector        Do[9];       // dummy detectors
  for(i=3; i<9; i++) {
    sprintf(file,"%s/Meyer1024_L%1d.dat",data_dir,i);
    cout<<file<<endl;
    Do[i].readFilter(file);
  }

  char outFile[1024];
  char tmpFile[1024];
  char endFile[1024];
  FileStat_t fstemp;

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// loop on factors
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++

  for(int iii=0; iii<nfactor; iii++) {

    factor=factors[iii];
    cout<<"factor="<<factor<<endl;

    if(simulation) {
      sprintf(outFile,"%s/wave_%d_%d_%s_%g_id%d_.root",
      nodedir,int(Tb),int(dT),output_label,factor,runID);
      sprintf(endFile,"%s/wave_%d_%d_%s_%g_id%d_.root",
      output_dir,int(Tb),int(dT),output_label,factor,runID);
      sprintf(tmpFile,"%s/wave_%d_%d_%s_%g_id%d_.root.tmp",
      nodedir,int(Tb),int(dT),output_label,factor,runID);

      if(!gSystem->GetPathInfo(endFile,fstemp)) {
printf("The file %s already exists - skip\n",endFile);
fflush(stdout);
continue;
      }

    }
    else {
      sprintf(outFile,"%s/wave_%d_%d_%s_id%d_.root",
      nodedir,int(Tb),int(dT),output_label,runID);
      sprintf(endFile,"%s/wave_%d_%d_%s_id%d_.root",
      output_dir,int(Tb),int(dT),output_label,runID);
      sprintf(tmpFile,"%s/wave_%d_%d_%s_id%d_.root.tmp",
      nodedir,int(Tb),int(dT),output_label,runID);
    }

    cout<<"output file on the node: "<<outFile<<endl;
    cout<<"final output file name : "<<endFile<<endl;
    cout<<"temporary output file  : "<<tmpFile<<endl;

    for(int j=0; j<NET.nLag; j++) WC[j].clear();

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// input data
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    double start, end, rate, duration;
// L1

    sprintf(file,"%s/%s%d",input_dir,fileNamesRaw[0],runID);
    readframes1(file,channelNamesRaw[0],x);
    printf("file=%s\n",file);

    start    = x.start();
    rate     = x.rate();
    end      = start+x.size()/rate;
    duration = end-start-2*waveoffset;

    fprintf(stdout,"start=%f end=%f duration=%f rate=%f\n",
    start,end,duration,rate);

    if(simulation) {
      sprintf(file,"%s/%s.lst.%d",input_dir,input_label,runID);
      readframes1(file,channelNamesMDC[0],y);
      y*=factor;
      x[slice(int((y.start()-x.start())*x.rate()),y.size(),1)]+=y;
      y.resize(1);
    }

    wB.Forward(x,levelR);
    wB.getLayer(x,0);
    L1.getTFmap()->Forward(x,S,levelD);
    L1.getTFmap()->lprFilter(1,0,120.,4.);
    L1.white(60.,w_mode,8.,30.);
    v[0] = L1.getTFmap()->variability();

    cout<<"After L1 data conditioning"<<endl;
    gSystem->Exec("date");

// H1

    sprintf(file,"%s/%s%d",input_dir,fileNamesRaw[1],runID);
    readframes1(file,channelNamesRaw[1],x);

    if(start!=x.start() || rate!=x.rate()) {
      fprintf(stderr,"H1/L1 mismatch: %f, %f, %f, %f\n",
      start,x.start(),rate,x.rate()); exit(1);
    }

    if(simulation) {
      sprintf(file,"%s/%s.lst.%d",input_dir,input_label,runID);
      readframes1(file,channelNamesMDC[1],y);
      y*=factor;
      x[slice(int((y.start()-x.start())*x.rate()),y.size(),1)]+=y;
      y.resize(1);
    }

    wB.Forward(x,levelR);
    wB.getLayer(x,0);
    H1.getTFmap()->Forward(x,S,levelD);
    H1.getTFmap()->lprFilter(1,0,120.,4.);
    H1.white(60.,w_mode,8.,30.);
    v[1] = H1.getTFmap()->variability();

    cout<<"After H1 data conditioning"<<endl;
    gSystem->Exec("date");

// H2

    sprintf(file,"%s/%s%d",input_dir,fileNamesRaw[2],runID);
    readframes1(file,channelNamesRaw[2],x);

    if(start!=x.start() || rate!=x.rate()) {
      fprintf(stderr,"H2/L1 mismatch: %f, %f, %f, %f\n",
      start,x.start(),rate,x.rate()); exit(1);
    }

    if(simulation) {
      sprintf(file,"%s/%s.lst.%d",input_dir,input_label,runID);
      readframes1(file,channelNamesMDC[2],y);
      y*=factor;
      x[slice(int((y.start()-x.start())*x.rate()),y.size(),1)]+=y;
      y.resize(1);
    }

    wB.Forward(x,levelR);
    wB.getLayer(x,0);
    H2.getTFmap()->Forward(x,S,levelD);
    H2.getTFmap()->lprFilter(1,0,120.,4.);
    H2.white(60.,w_mode,8.,30.);
    v[2] = H2.getTFmap()->variability();

    cout<<"After H2 data conditioning"<<endl;
    gSystem->Exec("date");

// G1

    sprintf(file,"%s/%s%d",input_dir,fileNamesRaw[3],runID);
    readframes1(file,channelNamesRaw[3],x);

    if(start!=x.start() || rate!=x.rate()) {
      fprintf(stderr,"H1/L1 mismatch: %f, %f, %f, %f\n",
      start,x.start(),rate,x.rate()); exit(1);
    }

    if(simulation) {
      sprintf(file,"%s/%s.lst.%d",input_dir,input_label,runID);
      readframes1(file,channelNamesMDC[3],y);
      y*=factor;
      x[slice(int((y.start()-x.start())*x.rate()),y.size(),1)]+=y;
      y.resize(1);
    }

    wB.Forward(x,levelR);
    wB.getLayer(x,0);
    G1.getTFmap()->Forward(x,S,levelD);
    G1.getTFmap()->lprFilter(1.,0,120.,4.);
    G1.white(60.,w_mode,8.,30.);
    v[3] = G1.getTFmap()->variability();

    cout<<"After G1 data conditioning"<<endl;
    gSystem->Exec("date");

    double R = x.rate();          // original data rate

    wB.resize(1);
    x.resize(1);

    printf("size=%d, segment start=%16.6f \n",
   L1.getTFmap()->size(), H1.getTFmap()->start());
```

```
    cout<<"live time: "<<NET.setVeto(4.)<<endl;  // set veto array

//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// initialization of output root file
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    TFile *froot = new TFile(tmpFile, "RECREATE");

    TTree* net_tree = netburst.setTree();
    TTree* live_tree= live.setTree();

    if(simulation) {
      TTree* mdc_tree = mdc.setTree();
    }
    else {
      TTree* var_tree = wavevar.setTree();
      TTree* noise_tree = noiserms.setTree();
    }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// low pass filtering
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    int low = (int(R+0.5)>>(levelD))/2; // finest frequency resolution
    int n   = lpfcut/low;               // number of layers to zero

    for(i=0; i<n; i++){
      L1.getTFmap()->getLayer(x,i); x = 0.; L1.getTFmap()->putLayer(x,i);
      H1.getTFmap()->getLayer(x,i); x = 0.; H1.getTFmap()->putLayer(x,i);
      H2.getTFmap()->getLayer(x,i); x = 0.; H2.getTFmap()->putLayer(x,i);
      G1.getTFmap()->getLayer(x,i); x = 0.; G1.getTFmap()->putLayer(x,i);
    }


//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// start of the coherent search
//++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
    double  Ao;
    double* pLN;

    // loop over TF resolutions
    for(i=levelD; i>=l_low; i--) {
      gSystem->Exec("date");

      if(i<=l_high) {

pLN = pln+3*(i-1); // pointer to lognormal parameters

L1.setFilter(Do[i]);
H1.setFilter(L1); H2.setFilter(L1); G1.setFilter(L1);
NET.setFilter(&H1,0.031);
NET.setDelay(&H1); NET.setDelay(&H2); NET.setDelay(&G1);

Ao = logNormArg(bpp,pLN[0],pLN[1],pLN[2])+0.1*(l_high-i);
cout<<"pixel threshold in units of noise rms: "<<Ao<<endl;

cout<<"core pixels: "<<NET.coherence4(Ao,4.,0.)<<"  ";

n = size_t(2.*Tgap*L1.getTFmap()->resolution(0)+0.1);
m = size_t(Fgap/L1.getTFmap()->resolution(0)+0.1);
if(n<1) n = 1;
if(m<1) m = 1;

cout<<"clusters: "<<NET.cluster(n,m)<<"  "<<endl;

cout<<"  pixels: "<<NET.likelihood4('l',true,Acore);
NET.corrcut(0.1,10,0);            // remove large glitches

cout<<"|"<<NET.likelihood4('c',false,Acore);
NET.corrcut(0.1,0,0);             // remove large glitches

for(j=0; j<NET.nLag; j++) {
  pwc = NET.getwc(j);
  wc = *pwc; *pwc = wc;
}
NET.setRMS();

cout<<"|"<<NET.likelihood4('C',false,Acore)<<" \n";
NET.setRank(8.);
NET.corrcut(0.3,0,0);             // pipeline correlation cut

for(j=0; j<NET.nLag; j++) {
  wc = *(NET.getwc(j));
  cout<<wc.csize()<<"|"<<wc.size()<<"|"<<WC[j].append(wc)<<" ";
}
cout<<endl;
      }

      if(i>l_low) NET.Inverse(1);

    }

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// supercluster analysis
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    size_t nevent=0;
    for(j=0; j<lags; j++){
      pwc = NET.getwc(j); *pwc = WC[j];
      m = pwc->supercluster('L',15,true);
      cout<<m<<"|"<<pwc->size()<<" ";
      nevent += m;
    }

    if(simulation) NET.printwc(0);

    cout<<"\nSearch done\n";
    cout<<"number of events: "<<nevent<<endl;
    gSystem->Exec("date");

//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
// save data in root file
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

    live.output(live_tree,&NET);

    if(simulation) {
      netburst.output(net_tree,&NET,factor);
      mdc.output(mdc_tree,&NET,factor);
    }
    else {
      netburst.output(net_tree,&NET);
      wavevar.output(var_tree,&v[0],1,waveoffset);
      wavevar.output(var_tree,&v[1],2,waveoffset);
      wavevar.output(var_tree,&v[2],3,waveoffset);
      wavevar.output(var_tree,&v[3],4,waveoffset);
      noiserms.output(noise_tree,&L1.nRMS,1,R/2);
      noiserms.output(noise_tree,&H1.nRMS,2,R/2);
      noiserms.output(noise_tree,&H2.nRMS,3,R/2);
      noiserms.output(noise_tree,&G1.nRMS,4,R/2);
    }

    froot->Write();
    froot->Close();

    char command[512];
    sprintf(command,"mv %s %s", tmpFile, outFile);
    gSystem->Exec(command);
    sprintf(command,"cp %s %s",outFile,output_dir);
    gSystem->Exec(command);
  }

  cout<<"Stopping the job "<<runID<<endl;
  gSystem->Exec("date");

  return 0;

}
```

10. Appendix C: Structure of the output ROOT file

```
Int_t           run;        // run ID
Int_t           nevent;     // event count
Int_t*          ifo;        // ifo ID: 1/2/3/.. - L1/H1/H2/..
Int_t*          eventID;    // event ID: [0]-prod, [1]-sim
Int_t*          type;       // event type: [0] - prod, [1]-sim
Int_t*          rate;       // 1/rate - wavelet time resolution
```

```
Int_t*          volume;     // cluster volume
Int_t*          size;       // cluster size (black pixels only)
Int_t           usize;      // cluster union size

Float_t*        gap;        // time between consecutive events
Float_t*        lag;        // time lag [sec]
Double_t*       strain;     // sqrt(h+*h+ + hx*hx)
Float_t*        phi;        // reconstructed/injected phi angle
Float_t*        theta;      // reconstructed/injected theta angle
Float_t*        bp;         // beam pattern coefficients for hp
Float_t*        bx;         // beam pattern coefficients for hx

Double_t*       snr;        // energy/noise_variance
Float_t*        rSNR;       // rank SNR
Float_t*        gSNR;       // gaussian SNR
Float_t*        rCF;        // rank confidence
Float_t*        gCF;        // Gaussian confidence
Float_t*        rSF;        // rank significance
Float_t*        gSF;        // Gaussians significance

Double_t*       time;       // average center_of_L time
Double_t*       START;      // segment start GPS time
Float_t*        right;      // cluster start relative to segment start
Float_t*        left;       // cluster stop relative to segment stop
Float_t*        duration;   // cluster duration = stop-start
Double_t*       start;      // GPS start time of the cluster
Double_t*       stop;       // GPS stop time of the cluster

Float_t*        frequency;  // average center_of_L frequency
Float_t*        low;        // min frequency
Float_t*        high;       // max frequency
Float_t*        bandwidth;  // high-low
Double_t*       hrss;       // log10(hrss)
Double_t*       noise;      // log10(noise rms)
Float_t*        chi2;       // Gaussian chi2 statistics
Double_t*       ndm;        // network data matrix (core pixels)
Float_t*        null;       // un-biased null statistics
Float_t*        nill;       // biased null statistics

Double_t        gnet;       // network sensitivity
Double_t        anet;       // network alignment factor
Double_t        likelihood; // network likelihood
```

[1] S. Klimenko and G. Mitselmakher, Wavelet method for GW burst detection *Class. Quantum Grav.* **21**, S1819 (2004).

[2] S. Klimenko I.Yakushin, and G. Mitselmakher*et al*, Preliminary S2 waveburst results *Class. Quantum Grav.* **21**, S1685 (2004).

[3] S.Klimenko, I.Yakushin and G.Mitselmakher, WaveBurst, LIGO note T040040-00-Z

[4] S.Klimenko, I.Yakushin and G.Mitselmakher, WaveBurst, version 5, LIGO note T050222-00-Z

[5] S.Klimenko, I.Yakushin and G.Mitselmakher, WaveBurst, S5 version, LIGO note T060112-00-Z

[6] `{ H.Siemens }`
`http://www.lsc-group.phys.uwm.edu/\~siemens/ht.html`

[7] S.Klimenko and I.Yakushin, Waveburst CVS,
`"http://www.lsc-group.phys.uwm.edu/cgi-bin/Analysis/Waveburst"`

[8] S.Klimenko and I.Yakushin, Wavelet analysis tool,
`"http://www.lsc-group.phys.uwm.edu/cgi-bin/Analysis/Waveburst/S4/wat",`

[9] S.Klimenko, S.Mohanty, M.Rakhmanov, G.Mitselmakher, Phys. Rev. D **72**, 122002 (2005).

[10] L. Cadonati, Coherent waveform consistency test for LIGO burst candidates. *Class. Quantum Grav.* **21**, S1695 (2004).

[11] S. Chatterji, L. Blackburn, G. Martin, E. Katsavounidis, Multiresolution techniques for the detection of gravitational-wave bursts. Class. Quantum Grav. **21**, S1809 (2004).

[12] B. Vidakovic, Statistical modeling by wavelets, 1999.

[13] W.Press et al., Numerical recipes, 1992

[14] R.Adhikary et al., Calibration of the LIGO detectors for the first LIGO scientific run, LIGO note T030097-00-D

[15] P.Sutton and K.Schlaufman, SenseMonitor,
`http://blue.ligo-wa.caltech.edu/gds//dmt/Monitors/SenseMonitor`

[16] All LIGO coordinates are verbatim from LIGO-P000006-D-E: Rev. Sci. Instrum., Vol. 72, No. 7, July 2001

[17] Anderson, Brady, Creighton, and Flanagan, PRD 63 042003, 2001

[18] `http://www.lsc-group.phys.uwm.edu/cgi-bin/bag-enote.pl?nb=burs4simulations&action=view&page=23`

[19] L. Wen and B. F. Schutz,Coherent network detection of gravitational waves: the redundancy veto, Class. Quantum Grav. **22**, S1321 (2005).