SANDIA REPORT
SAND2018-9700
Unlimited Release
August 2018

Bloodhound 0.8: A Python package for infrasound data analysis

Stephen Arrowsmith, Sam Tarin, Sarah Albert

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: http://www.ntis.gov/search

SAND2018-9700
August 2018
Unlimited Release

Bloodhound 0.8: A Python package for infrasound data analysis

Stephen Arrowsmith
6752
Sandia National Laboratories
P. O. Box 5800
Albuquerque, New Mexico 87185-MS0404

Abstract

This report provides details of the algorithms in the Bloodhound package for infrasound data analysis. The report provides a detailed description of the algorithms, general instructions on tuning Bloodhound for different signal types, and a complete listing of all input parameters and the complete output schema. Several Jupyter notebooks are provided with the distribution for illustrating how to use Bloodhound for different workflows.

1. OVERVIEW

Bloodhound contains a large set of algorithms for reading and processing infrasound data to detect events. It contains various functions to display input and output data. It also contains algorithms for propagation modeling, and an interface to external propagation modeling tools.

Hellhound implements a subset of the algorithms and functions in Bloodhound 0.8. A set of companion Jupyter notebooks are provided to illustrate how to run Bloodhound as an alternative to using Hellhound, and how to implement algorithms that are not currently implemented in Hellhound.

A key design feature in Bloodhound is the use of a profile that contains all the user-defined parameters for a given run (Note: the same profiles are used in Hellhound). The profile defines the data and all the station and network level parameters used to run Bloodhound/Hellhound.

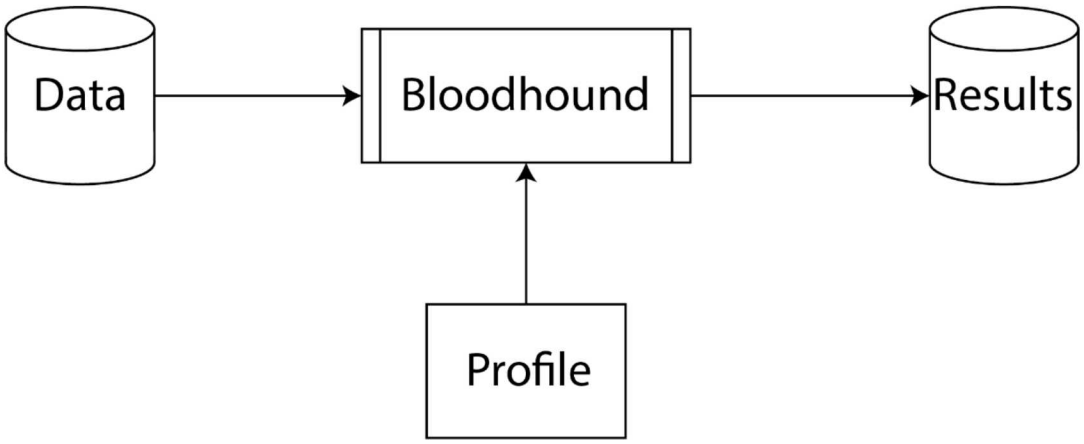

Figure 1-1 A high-level flowchart illustrating the inputs and outputs for a typical Bloodhound run for detecting signals and/or events.

1.1. Defining a profile

The profile is specified as a TOML file, a standard configuration file format that is designed to be flexible and easy-to-use.

A profile consists of the following tables: `[data]`, `[time]`, `[[filter]]` , `[[station-data]]`, `[[station-detection]]`, `[network]`. Tables denoted with a single pair of square brackets, `[]`, indicate that each parameter is specified only once (e.g., only one time-period is specified in a profile via `[time]`). Tables denoted with a double pair of square brackets, `[[]]`, indicate that multiple values of each parameter can be provided (e.g., unique detection parameters can be defined for each station via `[[station-detection]]`). The complete list of parameters for each table are provided in Appendix A. An example of a profile to process data from two stations (a

single component station and an array), using two different frequency bands for both arrays, and

```
[data]
type = "pickle"
file = "utah_data.p"

[time]
start = 2007-08-01T19:30:00+00:00
end = 2007-08-01T20:30:00+00:00

[[filter]]
low = 1.0
high = 5.0
window_length = 15.0

[[filter]]
low = 0.2
high = 1.0
window_length = 30.0

[[station-data]]
lat = 40.6528335
lon = -112.1194165
channels = "*"
elements = [ "NOQ3", "NOQ4", "NOQ5", "NOQ6"]
name = "NOQ"

[[station-data]]
lat = 39.455
lon = -114.0156
channels = "*"
elements = [ "P13A1"]
name = "P13A1"

[[station-detection]]
name = "NOQ"
minimum_duration = 5.0
p_thres = 0.05
lta = 60.0
signal_type = "regional"
sta = 3.0

[[station-detection]]
name = "P13A1"
minimum_duration = 5.0
p_thres = 0.01
lta = 60.0
signal_type = "regional"
sta = 3.0

[network]
lon_min = -115.0
lon_max = -109.0
lat_min = 36.0
lat_max = 42.0
azimuth_dev = 8.0
confidence_value = 0.95
```

a different p-value detection threshold at each array, is provided below.

More examples of TOML files for different scenarios can be generated using the Jupyter notebooks provided with the distribution. Bloodhound contains some convenience functions for generating TOML files from Oracle databases or ObsPy Stream objects.

2. PIPELINE DATA PROCESSING ALGORITHMS

This section provides the details of how Bloodhound performs pipeline data processing. As illustrated in Figure 1-1, Bloodhound 0.8 is designed to process data given a profile that defines the complete set of parameters for processing a certain signal type.

2.1. Station-level processing

Station-level processing is based on the Multivariate Adaptive Learning Detector (MALD) algorithm. MALD is designed to detect very low SNR signals, and to provide preliminary signal categorization, by combining multiple signal features as a function of both time and frequency.

The MALD algorithm has the following features:

- Multiple signal properties are combined to produce ensemble detection statistics for different source types as a function of time and frequency.
- The detection threshold is dynamic and adapts to a changing background.
- No assumptions of background distributions are made.

MALD is comprised of two main procedures, which are outlined below.

Procedure 1: The waveform data (either single-station or array data) are processed in time windows, or time-frequency bins, with a series of data transforms (refer to Figure 2-1).

Procedure 2: The data transforms are converted to p-values using Kernel Density Estimation, and combined with Fisher's combined probability test (refer to Figure 2-4). Detections are obtained from the ensemble detection statistics.

2.1.1. Processing modalities for station-level processing

As shown in Figure 2-1, Bloodhound 0.8 includes two processing modalities for station-level processing: broadband processing and narrowband processing. A third method that supports frequency-domain processing will be added in a future version.

- For broadband processing, Bloodhound can process data in multiple frequency bands as defined by the user. Detections are built separately in each frequency band. Prior to network processing, duplicate detections of the same signal (in different frequency bands) are associated by defining master detections (only master detections are used in association processing).
- For narrowband processing, Bloodhound can process data in multiple frequency bands as defined by the user (utility functions exist to define 1/3 Octave bands, and to define the default PMCC logarithmic bands). For narrowband processing, detections are constructed by clustering and aggregating processing results across multiple frequency bands.

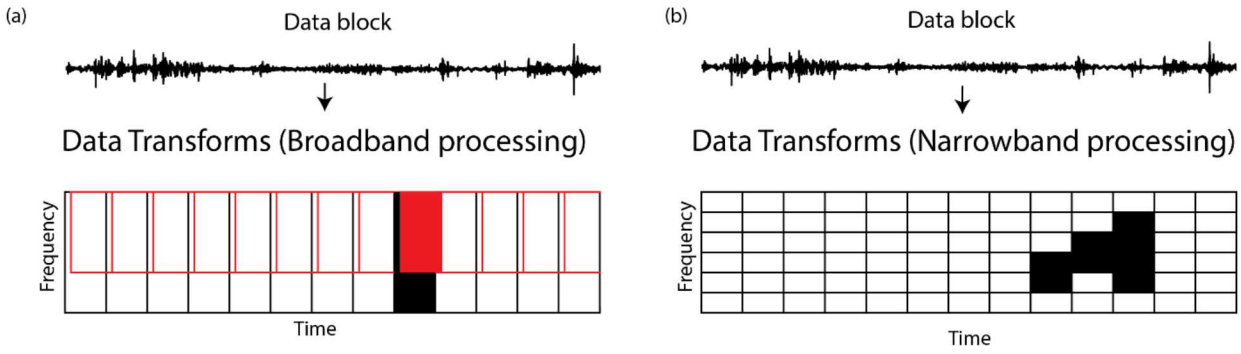

Figure 2-1 Schematic of processing modalities supported by the MALD algorithm in Bloodhound v0.8. A data block is either processed using broadband (left) or narrowband (right) processing. (left) Broadband processing may consist of one or multiple filter bands, which may overlap, where detections are built in each filter band separately. Detections are associated where they overlap in time and contain additional properties (e.g., backazimuth), with each set of associated detections comprising one master detection (shown by the solid black rectangle in a) and one or more associated detections (e.g., the solid red rectangle in a).

We note that Figure 2-1 is a schematic and there are three important features that it does not represent:

1) While the time windows in Figure 2-1 are depicted to be the same for each transform), this is not necessarily the case in practice. In fact, different time windows are recommended for different frequency bands.
2) Time bins will typically overlap. It is recommended that time windows overlap by a minimum of 50% to adequately capture the time evolution of waveform properties.
3) Frequency bins may overlap when performing broadband processing. For narrowband processing, they must be adjacent and cannot overlap.

2.1.2. Procedure 1: Data transforms

Bloodhound 0.8 uses three primary data transforms, depending on whether the user is processing single-component data, three-component data, or array data. Additional data transforms (secondary transforms) are applied as further constraints on the primary transforms. The primary data transforms are, for the most part, standard signal processing tasks used in seismic (and infrasound) data processing, and subsequently are only discussed at a high level, with references provided to supporting literature.

Single-component data

The primary data transform for single-component data is either the STA/LTA transform (Withers, et al. 1998) or the STA transform. The STA/LTA transform takes the difference between the mean square value of a filtered time series in a short-time window, and the mean

square value of a filtered time series in a preceding long-time window. Thus, the STA/LTA ratio can be written as:

$$STA\_LTA = \frac{\frac{1}{N_S}\sum_{i=1}^{N_S} a_i^2(t)}{\frac{1}{N_L}\sum_{i=1}^{N_L} a_i^2(t)}$$

where a is the acoustic pressure (or the seismic displacement/velocity/acceleration), N_S is the number of samples in the short-time window, and N_L is the number of samples in the long-time window. The STA transform is simply:

$$STA = \frac{1}{N_S}\sum_{i=1}^{N_S} a_i^2(t)$$

The STA transform is not commonly used by itself, because the units are not normalized, but can be directly used in the MALD framework because the KDE mapping to p-values (described below) provides normalization. By using STA instead of STA/LTA, Bloodhound can properly capture long-duration signals as well as impulsive onsets.

As with all the data transforms, the STA/LTA or STA transforms can be applied in a series of discrete frequency bands. The output of STA/LTA processing is stored in the sta_lta table, and STA processing is stored in the sta table (Appendix B).

Three-component data

Three-component data are further processed with polarization processing (Jurkevics, 1988). If we denote the bandpass filtered three-component data in a time window of length $N\Delta t$ as:

$$\underline{X} = \begin{bmatrix} e_1 & n_1 & z_1 \\ \vdots & \vdots & \vdots \\ e_N & n_N & z_N \end{bmatrix}$$

(where $\underline{e}$, $\underline{n}$ and $\underline{z}$ denote the east, north, and vertical components) then the covariance matrix can be calculated and represented as:

$$\underline{S} = \begin{bmatrix} S_{ee} & S_{ne} & S_{ze} \\ S_{en} & S_{nn} & S_{zn} \\ S_{ez} & S_{nz} & S_{zz} \end{bmatrix}$$

where $S_{jk} = \left[\frac{1}{N}\sum_{i=1}^{N} x_{ij}x_{ik}\right]$. The eigenvalues $(\lambda_1,\lambda_2,\lambda_3)$ of the covariance matrix, where $\lambda_j \geq \lambda_k$ for $j < k$, allow us to calculate the rectilinearity in the time window from:

$$r = 1 - \left(\frac{\lambda_2 + \lambda_3}{2\lambda_1} \right)$$

The eigenvector corresponding to the largest eigenvalue (u_{1e}, u_{1n}, u_{1z}) allows us to calculate an unambiguous backazimuth from:

$$\theta = atan2\left(u_{1e} sign(u_{1z}), u_{1n} sign(u_{1z})\right)$$

Note that the sign convention in Bloodhound assumes that the signal is a ground-coupled infrasound arrival (which has the opposite sign to an upcoming p-wave). For example, an infrasound signal arriving from due west with downward inclination angle of 45° to the horizontal will result in a particle motion where +E corresponds to -Z (positive infrasound overpressure causes downward vertical motion and positive motion to the east). In contrast, a p-wave arriving from due west with upward inclination angle of 45° will result in a particle motion where +E corresponds to +Z. For waves traveling purely horizontally, there is a 180° ambiguity in backazimuth. It should be noted that many ground-coupled infrasound signals can stimulate surface waves where the polarity is reversed, thus the sign should be treated with caution.

When computing polarizations for broadband processing, Bloodhound computes a wide-band estimate (Jurkevics, 1988) such that each estimate of the polarization is obtained by computing separate covariance matrices in third Octave bands, then normalizing and averaging. If an individual covariance matrix for the k'th band is denoted as $\underline{S}^k$, then the wide-band estimate is obtained from:

$$\overline{\underline{S}} = \sum_{k=1}^{K} \frac{\underline{S}^k}{trace(\underline{S}^k)}$$

where K is the number of third Octave bands between the minimum and maximum frequency of the broadband filter. It has been found that wide-band estimates provide improved estimates of azimuth than estimates made using broadband filtered data. The output of polarization processing is stored in the polarization table (Appendix B).

Arrays

The primary data transform for arrays is the sliding window FK method (Rost and Thomas 2002). The sliding-window FK method provides estimates of the power as a function of time and slowness vector (equivalent to trace velocity and backazimuth), through a grid search method. Bloodhound 0.8 provides two different implementations of sliding-window FK: a frequency-domain implementation, and a time-domain implementation. The time-domain implementation is slower, but can be used for incoherent array processing, which is particularly useful for processing high-frequency observations where the array aperture is too large for coherent processing.

We define a slowness vector, $\underline{u} = (u_x, u_y)$ that uniquely defines the direction-of-arrival of a plane wave arriving at an array. The trace velocity is $v_t = \left(\sqrt{u_x^2 + u_y^2}\right)^{-1}$ and the backazimuth is $\theta = \tan^{-1}\left(\frac{u_x}{u_y}\right)$. The time shift for the *i*'th array element, for a given slowness vector, is computed as:

$$\Delta t_i = x_i u_x + y_i u_y = \underline{r}_i \cdot \underline{u}$$

For the frequency domain FK, the time delays for different elements are handled through phase shifts for the different harmonic components of the waveform. The power is computed from:

$$P(u_x, u_y) = \frac{1}{N^2} \sum_{f=f_1}^{f_2} \left| \frac{1}{M} \sum_{i=1}^{M} X_i(f) e^{-i2\pi f \underline{r}_i \cdot \underline{u}} \right|$$

where $X_i(f)$ is the short-time Fourier transform of the time series of the *i*'th array element, $x_i(t)$ in a time window that contains N samples (note that the N^2 is the proper normalization, and follows from the discrete implementation of the fast Fourier transform). After applying frequency domain processing, the actual value stored by Bloodhound is the semblance – computed by normalizing the power by the total power in the seismograms. Only the semblance for the slowness direction with the maximum power is stored, along with the corresponding direction-of-arrival information.

The time-domain FK implementation is designed to operate in a pairwise strategy. This approach is more robust to noisy or corrupted channels affecting the result, and is implemented in the time domain version to complement the approach used in the frequency domain version, which uses all sensors simultaneously in an estimate of the power. The time-domain FK implementation is based on the normalized cross-correlation between each pair of traces as a function of slowness vector. For two array elements, *i* and *j*:

$$r_{ij}(u_x, u_y) = \frac{\sum x_i(t + \underline{r}_i \cdot \underline{u}) x_j(t + \underline{r}_j \cdot \underline{u})}{\sqrt{\sum (x_i(t) x_i(t)) \sum (x_j(t) x_j(t))}}$$

For a given slowness value, the mean value of r across all pairs of elements is computed. The value stored by Bloodhound in a given time window is then the maximum of the mean values, with the corresponding direction-of-arrival. For processing data with the time-domain FK implementation, the data can be first converted to an analytic envelope to remove the phase information.

Sliding-window FK transforms are applied, either in time or frequency domain, in any number of frequency bands and stored separately for each band. For each frequency band the user must specify the low and high pass corner frequencies, as well as the processing time window and overlap. For each time window, Bloodhound finds the value of the semblance, or mean

normalized cross-correlation, corresponding to the peak power across all backazimuths and trace velocities, and stores the corresponding direction-of-arrival (see the fk table in Appendix B).

2.1.3. Procedure 2: Secondary transform, KDE estimation, multivariate fusion, and detection

The processing of the data transform data in Bloodhound is quite different from conventional seismic data processing packages. The following narrative describes how data transform results are processed to detect signals for a single frequency band. The extension to detection of signals when performing processing with multiple frequency bands is discussed in Section 2.1.3.5.

2.1.3.1. Secondary transform

To enhance the detection of signals on arrays, a secondary transform is applied to the output of the FK transform. The secondary transform enhances the detection of signals with low signal-to-noise, or with low correlation between array elements. When considering the direction-of-arrival (DOA) of a signal as a function of time, a static source will typically exhibit a roughly constant backazimuth as a function of time. A moving source will show a shift in backazimuth as a function of time (following the shape of a sigmoid function if moving in a straight line and constant velocity). Given an estimate of the DOA as a function of time from FK processing, both types of signal can be represented by the following logistic function:

$$f(t, a, b, c) = c + \frac{a}{1 + e^{-b\left(t - t_0\right)}}, \qquad a \in \{0, 360\}, c \in \{0, 360\}$$

In this equation, a defines the amplitude, b controls the steepness of the function, and c controls the intercept. Note that by setting $b = 0$, the function becomes a horizontal straight line. In a moving time window of duration T_v, the best fit to the function is evaluated using nonlinear least squares. The resultant variance between the best fitting function and the observed data, $\sigma_f^2 = \sum r_i^2 / N$, where r_i is the residual between the i'th predicted and observed backazimuth, is calculated in each window. As described below in Section 2.1.3.4, when forming detections, the relative fit of a logistic function versus a horizontal straight line is used to distinguish between likely static and moving sources.

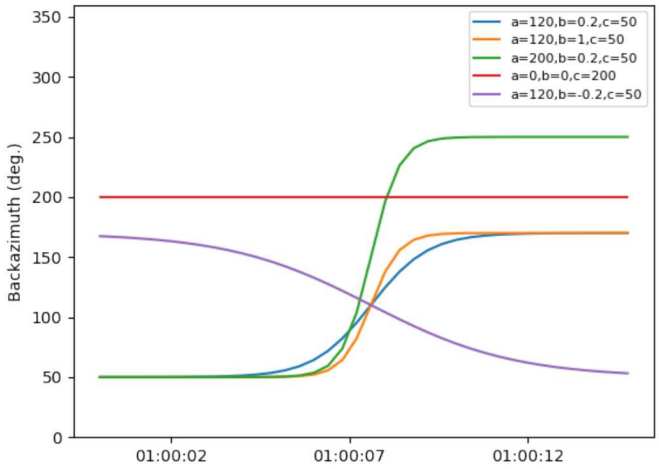

Figure 2-2 Plots of different logistic functions illustrating the effects of a, b, and c on the resultant curve.

To enhance the detection of static and moving sources, σ_f^2 is saved in memory as an additional data transform (which is converted to p-values and combined with semblance as described below). Note that, based on the users choice of `signal_type` (Appendix A), Bloodhound either computes σ_f^2 using the logistic function above, or using a simple horizontal straight-line function. The latter option detects fewer signals, as it is more restrictive, but allows for the detection of weak static sources.

The derivation of the secondary transforms from the backazimuth time series obtained from the primary FK transform is illustrated schematically in Figure 2-3.

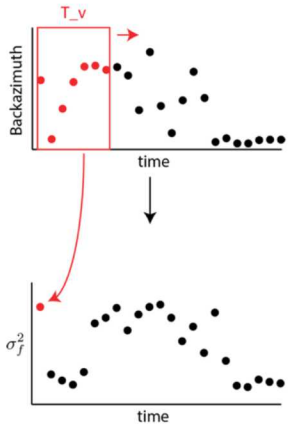

Figure 2-3 Schematic illustration of the derivation of secondary transforms from the backazimuth time series obtained from FK processing.

2.1.3.2. KDE estimation

Test statistics associated with the output of each data transform (both primary and secondary) are converted to p-values using the Kernel Density Estimation (KDE) method. The process is outlined graphically in Figure 2-4. If we denote a set of realizations of a specific transform (e.g., rectilinearity) in a time interval of duration equal to one hour as $(S_1,S_2,\ldots,S_n)$, then the KDE is:

$$\hat{f}_h(S) = \frac{1}{Nh}\sum_{i=1}^{N} K\left(\frac{S - S_i}{h}\right)$$

where $K(\cdot)$ is the kernel (typically a Gaussian kernel) and h>0 is a smoothing operator.

Given this mapping, a given value of the transform can be converted to a p-value, where the p-value is defined as:

$$p = \int_{S_{obs}}^{\infty} \hat{f}_h(S)\, dS ,$$

where S_{obs} is an individual value of the transform. Intuitively the p-value is the probability of obtaining a result that is equal to, or more extreme than, that actually observed given the empirical PDF.

2.1.3.3. Multivariate fusion

Given multiple transforms, which exploit different signal properties, we combine the p-values associated with all k transforms for a given signal type using Fisher's method to produce ensemble p-values (Figure 2-4):

$$\chi^2 = -2\sum_{i=1}^{k} \ln p_i$$

The resultant value of the test statistic is distributed as a Chi-squared distribution with $2k$ degrees of freedom. A signal detector is applied to this combined distribution for a given p-value threshold.

For processing array data, Bloodhound 0.8 combines p-values computed from semblance, S, with p-values computed from σ_f^2 (i.e., from the secondary transform). For processing three component data, Bloodhound 0.8 combines p-values computed from STA_LTA or STA with p-values computed from r (rectilinearity). For processing single component data, Bloodhound 0.8 only uses p-values computed from STA_LTA or STA.

2.1.3.4. Detection

To build detections for a single frequency band, Bloodhound starts with a set of ensemble p-values as a function of time (Figure 2-5). First, processing time blocks with successive p-values below the threshold are merged together to form candidate detections (in some cases, a candidate detection may comprise a single time block). Next, for each candidate detection, the duration, median estimates of backazimuth and trace velocity (if applicable), estimates of the backazimuth at the start and end of the detection (if applicable), and maximum estimates of relevant data transforms (semblance, rectilinearity, STA/LTA, STA etc.) are computed. Optional constraints can be applied to filter out detections that do not meet certain characteristics (these include a minimum duration, a minimum STA/LTA threshold, and a minimum semblance threshold).

For arrays, each detection is categorized as arising from either a static or moving source by computing an F-test over the duration of the detection:

$$F = \frac{\sigma_f^2}{\sigma_{stat}^2}$$

where σ_f^2 is the variance of the best fitting sigmoid function to the time series of backazimuths at peak semblance (over the detection), and σ_{stat}^2 is the variance of the best fitting horizontal line function to the same data. Each F ratio is converted to a p-value where:

H_0: $\sigma_{mov}^2 = \sigma_{stat}^2$
H_a: $\sigma_{mov}^2 < \sigma_{stat}^2$

and Bloodhound uses the standard F distribution with $N-3$ degrees of freedom on the numerator and $N-1$ degrees of freedom on the denominator, where N is the number of samples of backazimuth in the detection. Detections with p-values below a threshold are flagged as arising from possible moving sources (Appendix B).

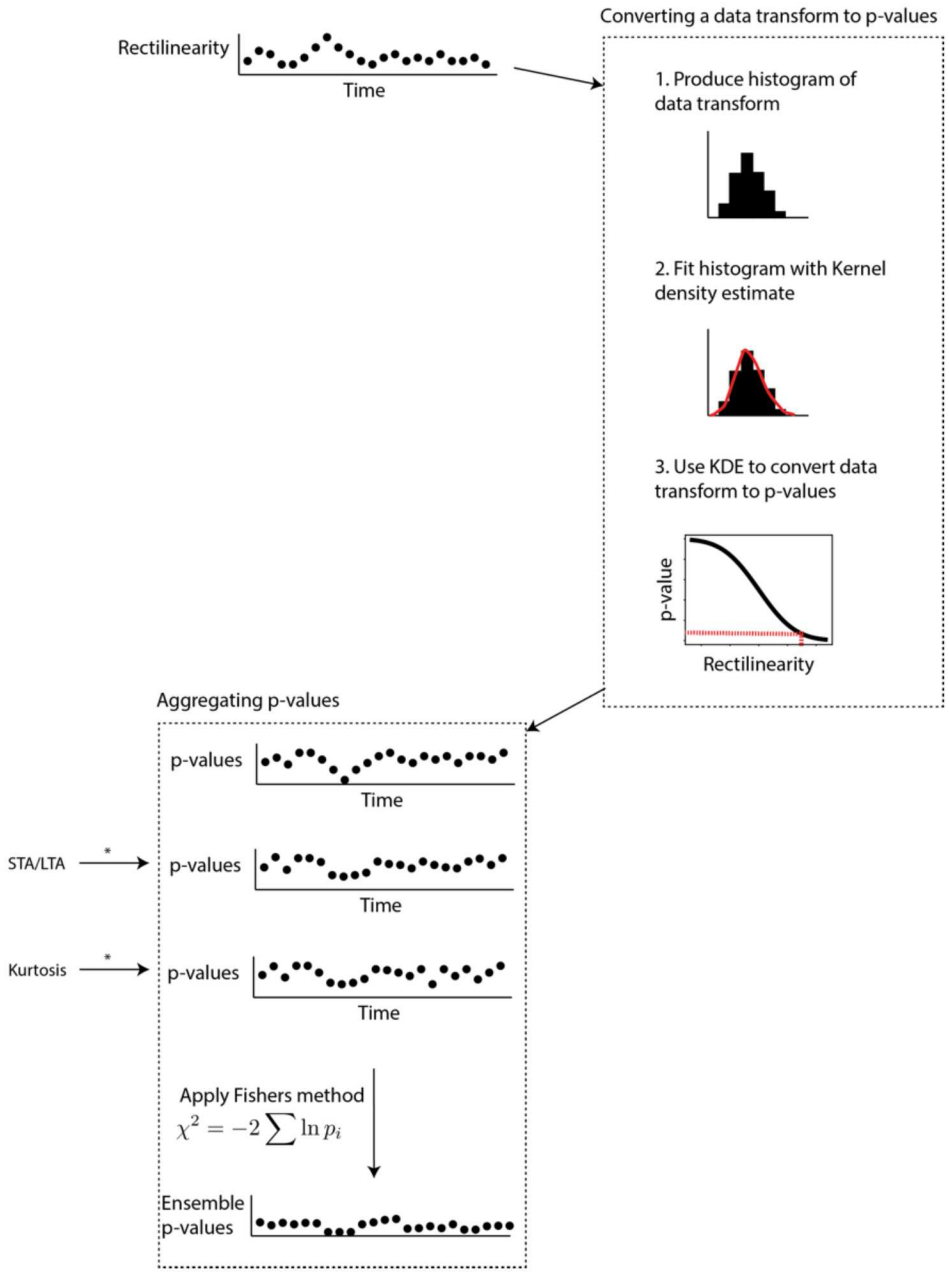

Figure 2-4 Schematic of the KDE estimation and multivariate fusion procedures of the MALD algorithm. The results of a given data transform (e.g., rectilinearity) for a specific frequency band, and in a one-hour time window, are converted to p-values using Kernel Density Estimation. P-values are aggregated with p-values from other data transforms (e.g., STA/LTA and Kurtosis) using Fishers method to produce ensemble p-values.

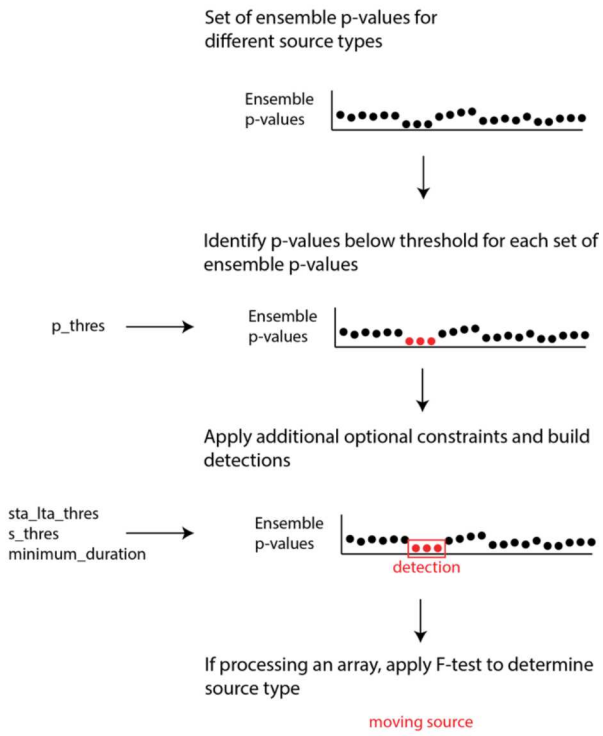

Figure 2-5 Schematic of the detection procedures of the MALD algorithm.

2.1.3.5. Handling detections in different frequency bands

Broadband processing: Defining master detections

Prior to performing network processing, Bloodhound associates detections across multiple frequency bands when using the broadband processing modality. Detections that overlap in time (and where the backazimuths are consistent, if detections contain backazimuths) are associated together. For each set of associated detections, a master_arid is defined (see Appendix B). The master_arid is the arid of the detection that has the lowest p-value (i.e., the most significant detection) of the set of associated detections. For association processing, only the detections that are master_arids are used, preventing multiple detections of the same signal from creating multiple duplicate events.

Narrowband processing: Clustering and aggregating detections

When processing using the narrowband modality, Bloodhound aggregates detections that are similar in time and adjacent in frequency bands. Bloodhound forms metadetections by querying the detection table in the database. First, the user sets the time difference threshold and Bloodhound compares the start times and end times of detections. If the start time of a later detection falls within the specified time difference of the end time of the previous detection, the two are merged into a metadetection. If the start time of the later detection does not fall within the specified time difference, a new metadetection is formed (Figure 2-6). Bloodhound then

continues to step though the detections and do this comparison until it reaches the end. Once metadetections are formed using the time difference threshold, Bloodhound goes back to the beginning of the detections and compares frequency bands in the same way (Figure 2-6). The frequency band difference is set by the user. In most cases it is useful to consider adjacent frequency bands as part of the same detection. Large frequency band thresholds are likely unnecessary as frequency bands that are not adjacent may be from different sources and would require more analysis.

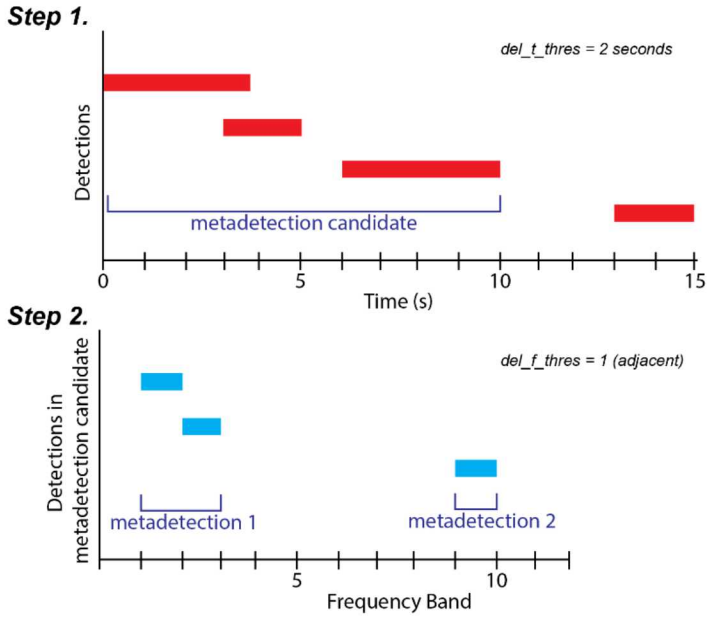

Figure 2-6 Illustration of the metadetection algorithm

After metadetections are formed, Bloodhound pulls information about each metadetection from the fk table in the database. These values are pulled based on the start times, end times, and frequency bands of each detection within the metadetection. The fk table is queried by start time, end time, and frequency band ID for each detection within the metadetection to get semblance, backazimuth, and trace velocity. Each of these semblance, backazimuth, and trace velocity values are then stored in the metadetection table with the corresponding unique metadetection ID and frequency band ID. This allows for the user to query the metadetection table for these values and run statistics such as mean and standard deviation of the backazimuth for a specific metadetection.

2.1.3.6. Detection reconciliation

Bloodhound 0.8 performs all previous steps in Procedure 2 of MALD (i.e., secondary transform, KDE estimation, multivariate fusion, detection, and multiple frequency aggregation) in a series of time windows of duration equal to one hour (by default) that overlap by 25%. The use of overlapping windows ensures that signals near the start or end of a given window are not missed (either because the secondary transform does not adequately capture the signal, or because the KDE estimation does not adequately capture the noise). As a consequence of the overlapping windows, signals may be detected twice, therefore a detection reconciliation step is required to reconcile duplicate detections. The detection reconciliation algorithm searches for detections in a

given frequency band that overlap in time, and re-runs the detector to ensure that these signals are suitably captured.

2.2. Network-level processing

Network-level processing in Bloodhound has the following features:

- Provides formal uncertainty bounds using Bayesian inference or least-squares inversion.
- Fast and parallelized.
- Includes meteorological corrections (not currently part of pipeline processing. but available as a post-processing function) - see Section 3 for more details.

Network-level processing is based on two stages, which are outlined in Figure 2-7.

Stage 1: The detections are read from a Sqlite3 database and processed with a grid-based associator to find events. The associator is designed to ensure that events are not missed by using simple bounding constraints that define the maximum possible propagation effects on the observed signals.

Stage 2: Events that are found in the association stage are located using one of two methods: Bayesian inference or iterative least-squares inversion. Both methods provide formal confidence estimates at a specified confidence level.

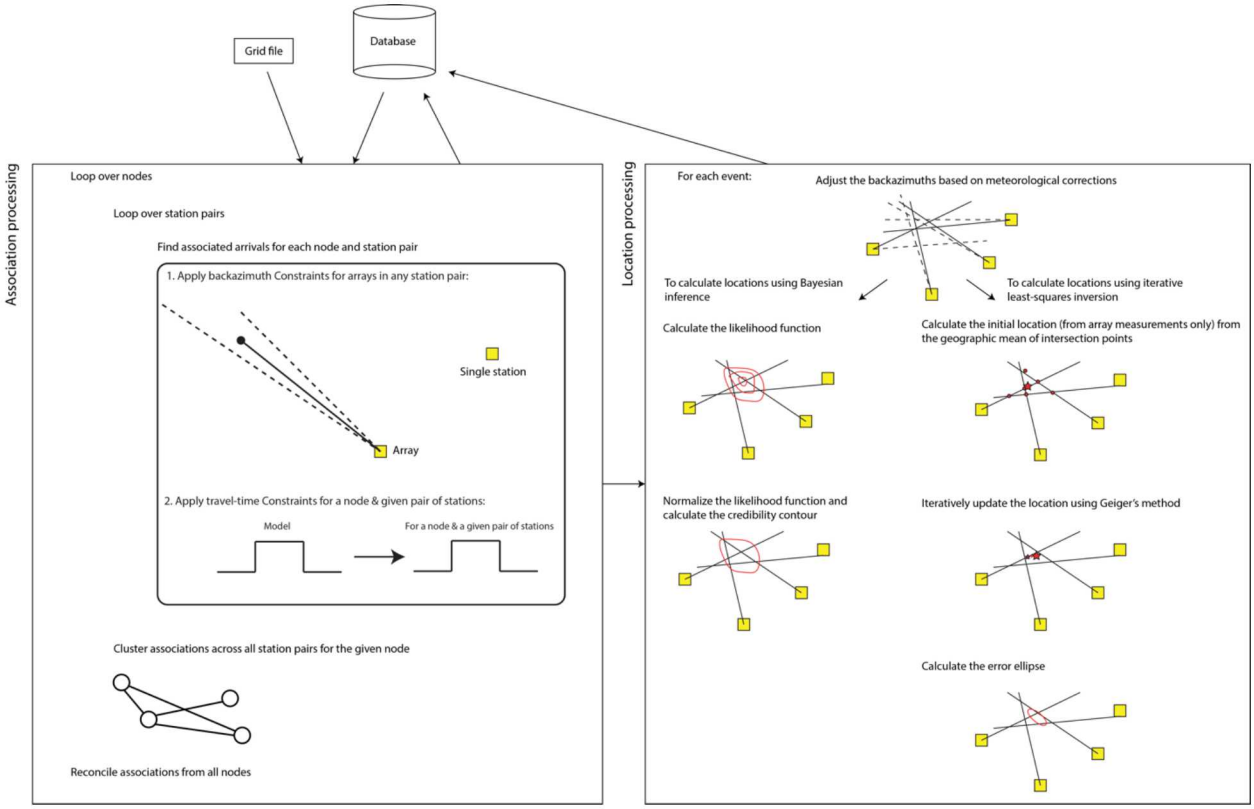

Figure 2-7 Schematic of the Bloodhound network-level algorithms.

2.2.1. Stage 1: Association

Association is performed using a grid-search method that checks for consistency between pairs of arrivals and then aggregates pairs of arrivals together using breadth-first search. The basic check for consistency between a pair of arrivals at a given vertex in the grid is based on the following:

$$\frac{|\Delta_i - \Delta_j|}{v_+} \leq dt \leq \frac{|\Delta_i - \Delta_j|}{v_-}$$
$$\phi_i - \delta\theta \leq \theta_i \leq \phi_i + \delta\theta$$
$$\phi_j - \delta\theta \leq \theta_j \leq \phi_j + \delta\theta$$

where i and j represent two arrays, dt is the time delay between the detections at each array, θ_i and θ_j are the observed backazimuths at each array, Δ is the great-circle distance from an array to a given vertex, ϕ is the corresponding azimuth of the great-circle from the array to the vertex, v_+ and v_- are maximum and minimum group velocities (or celerities), and $\delta\theta$ is an allowed deviation in azimuth due to wind bias and measurement uncertainty. This simple bounding model ensures that no events are missed as long as a sufficient range in values is defined by v_+, v_-, and $\delta\theta$.

The algorithm used for incorporating the constraints above in order to form events is explained in Figure 2-8 using three example detections (represented by the identifiers 57, 894, and 1031). The set of detection-pair associations for each vertex (e.g., Vertices 30 and 31 in Figure 2-8) can be represented as a graph such that each arrival represents a node (represented by circles in Figure 2-8) and each association represents an edge (represented by dashed lines in Figure 2-8). Using breadth-first search we find subgraphs representing connected components of the graph that constitute seed events for each vertex. For example, in Figure 2-8(b), the two-array association between arrivals 1031 and 57 is linked to the two array association between arrivals 57 and 894 because both associations have arrival 57 in common. Each seed event now contains a set of N arrivals (nodes) connected by M associations (edges). In many cases a single set of arrivals, or subsets of those arrivals, may be associated across multiple vertices (e.g., in Figure 2-8, arrivals 894 and 57 are linked at both Vertices 30 and 31). Seed events across multiple vertices that contain the same arrivals (or subsets) are reconciled by retaining only the association at the vertex that contains the largest number of arrivals (e.g., for the example in Figure 2-8, the association between arrivals 57 and 894 on Vertex 31 is rejected, because it is the subset of a larger event formed on Vertex 30). In some cases, multiple vertices may be associated with a single association of arrivals. The output of this procedure is a set of associations of arrivals for input to the location algorithm.

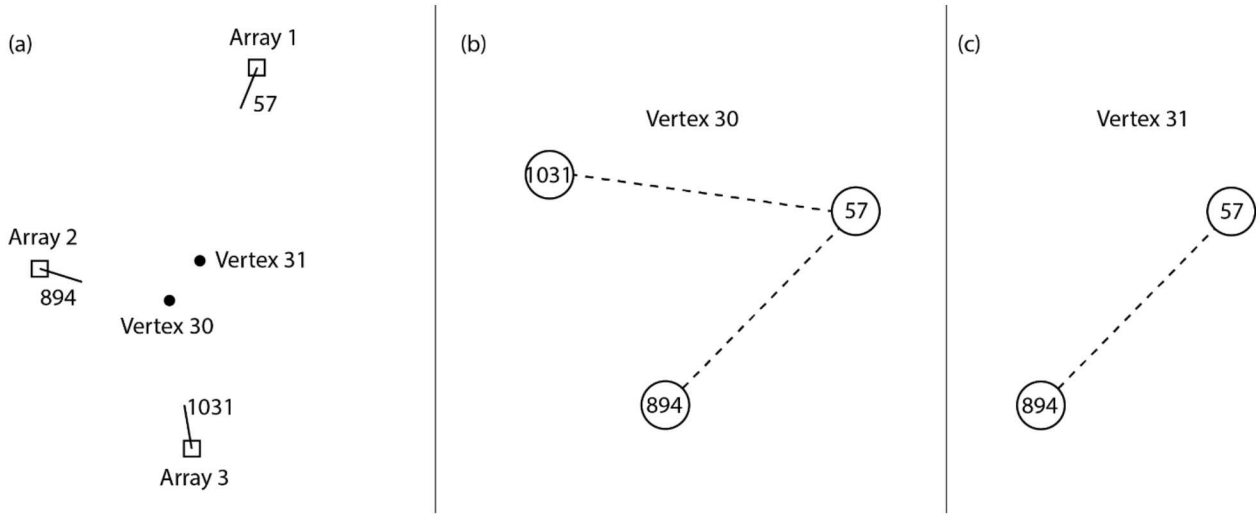

Figure 2-8 An illustration of the association algorithm used in Bloodhound. The cartoon map in (a) represents an example configuration of arrays with three hypothetical detections from a single event (detections are represented by lines pointing out from each array at the detection backazimuths, and have example arrival ID's of 57, 894, and 1031). Also shown are two hypothetical vertices (30 and 31). In (b), an example association, represented as a graph, is shown for Vertex 30. In this case, there are two two-array associations: (1) the association of arrival 1031 at Array 3 with arrival 57 at Array 1, and (2) the association of arrival 57 at Array 1 with arrival 894 at Array 2. These associations are merged to form a single event that comprises all three arrivals, on the basis of having Arrival 57 in common. In (c), an example association is shown for Vertex 31. In this case, the association is rejected because it is the subset of a larger event formed on Vertex 30.

2.2.2. Stage 2: Location

Bloodhound contains two location algorithms. The preferred algorithm is the Bayesian infrasound location method. However, since the Bayesian method is slow for global distances, the second method (iterative least-squares inversion) is provided as another option.

Bayesian Location

Location using backazimuths only

If using backazimuths only, the location is based on the construction of a Likelihood function as follows:

$$P(\underline{b}|\phi_j,\lambda_j) = \prod_{i=1}^{N} \Theta\left(b_i|\phi_j,\lambda_j\right)$$

where $\underline{b} = [b_1,b_2,\cdots,b_N]$ contains the observed backazimuths at each station, ϕ_j is the latitude of the *j*'th event hypothesis, and λ_j is the longitude of the *j*'th event hypothesis), and the individual likelihood components due to each station are either:

$$\Theta\left(b_i|\phi_j,\lambda_j\right) = \frac{1}{\sqrt{2\pi\sigma_\theta^2}} e^{-\frac{1}{2}\left(\frac{\delta_i^2}{\sigma_\theta^2}\right)}$$

or,

$$\Theta\left(b_i|\phi_j,\lambda_j\right) = \frac{e^{\kappa\cos\delta_i}}{2\pi I_0(\kappa)}$$

depending on whether one chooses either a Gaussian or Von Mises distribution (i.e., the Gaussian distribution on a circle) to describe the azimuthal dependence of the likelihood distribution. For both distributions, δ_i is the residual between the observed and predicted backazimuth at a given station $\left(\delta_i = \theta_i^{obs} - \theta_i^{pre}\right)$ given a specific choice of model parameters. For the Gaussian distribution, σ_θ is the a-priori estimate of the standard deviation in backazimuth (which includes both measurement and model error). For the Von Mises distribution, $1/\kappa$ is analogous to σ_θ^2 and $I_0(\kappa)$ is the modified Bessel function of order 0. The Von Mises distribution is more applicable than the Gaussian distribution for location over large regions (where the Earths curvature becomes important).

With a prior, $P(\phi_j,\lambda_j)$, defined as a uniform distribution over some bounded region (i.e., a regional area of study), the posterior distribution is evaluated by numerical integration of the likelihood function:

$$P(\phi_j,\lambda_j|\underline{b}) = \frac{\prod_{i=1}^{N} \Theta\left(b_i|\phi_j,\lambda_j\right) P(\phi_j,\lambda_j)}{\iint \Theta\left(\theta_i|\phi,\lambda\right) P(\phi,\lambda)\, d\phi\, d\lambda}$$

Credibility contours are obtained directly from the posterior distribution, since the total volume enclosed by the distribution, after normalization, is unity. Contours are obtained from a grid search over trial values of the posterior distribution to find the value that most closely encloses a volume equal to the credibility value chosen (e.g., 0.95).

If using backazimuths, the origin time is estimated after calculating a maximum likelihood location by assuming an average celerity (or group velocity) to all stations given the maximum and minimum possible celerities. The mean origin time given all the observed arrival times is taken as the estimated origin time.

Location using arrival times only

If using arrival times only, and without performing propagation modeling, a uniform distribution on celerity is used. The uniform distribution is bounded by the minimum and maximum celerities that are possible (e.g., $v_{min} = 0.22$, $v_{max} = 0.35$ km/s would capture the slowest thermospheric returns through to direct propagation along the ground).

For arrival times, the equation for the likelihood is:

$$P(\underline{t}|\phi_j,\lambda_j,\tau_j) = \prod_{i=1}^{N} \Phi\left(t_i|\phi_j,\lambda_j,\tau_j\right)$$

where $\underline{t} = [t_1,t_2,\cdots,t_N]$ contains the observed arrival times at each station, ϕ_j is the latitude of the *j*'th event hypothesis, λ_j is the longitude of the *j*'th event hypothesis, and τ_j is the origin time of the *j*'th event hypothesis. The individual likelihood components due to each station are:

$$\Phi\left(t_i|\phi_j,\lambda_j,\tau_j\right) = \begin{cases} \dfrac{1}{\left(\Delta_{ij}/v_{min} + \Delta t\right) - \left(\Delta_{ij}/v_{max} - \Delta t\right)} & if\ \left(\Delta_{ij}/v_{max}\right) + \Delta t \le t_i - \tau_j \le \left(\Delta_{ij}/v_{min}\right) - \Delta t \\ 0 & otherwise \end{cases}$$

where Δ_{ij} is the great-circle distance from the *j*'th event hypocenter to the *i*'th station and Δt is a term that accounts for both measurement uncertainty and the spatial and temporal resolution of the grid of event hypotheses. In this case, the posterior distribution has three dimensions and one must marginalize the posterior distribution to plot the marginal distribution over either location or origin time.

Mathematically, the posterior distribution is defined as:

$$P(\phi_j, \lambda_j,\tau_j|\underline{t}) = \frac{\prod_{i=1}^{N} \Phi\left(t_i|\phi_j, \lambda_j,\tau_j\right) P(\phi_j, \lambda_j,\tau_j)}{\iiint \prod_{i=1}^{N} \Phi\left(t_i|\phi, \lambda,\tau\right) P(\phi, \lambda,\tau)\, d\phi\, d\lambda\, d\tau}$$

with the marginal posterior distribution over location defined as:

$$P(\phi_j, \lambda_j|\underline{t}) = \int P\left(\phi_j, \lambda_j,\tau|\underline{t}\right) d\tau$$

and the marginal posterior distribution over time defined as:

$$P(\tau_j|\underline{t}) = \iint P\left(\phi, \lambda,\tau_j|\underline{t}\right) d\phi\, d\lambda$$

Location using backazimuths and arrival times

For arrival times, the equation for the likelihood is:

$$P(\underline{t},\underline{b}|\phi_j,\lambda_j,\tau_j) = \prod_{i=1}^{N} \Theta\left(b_i|\phi_j,\lambda_j\right)\Phi\left(t_i|\phi_j,\lambda_j,\tau_j\right)$$

The posterior distribution is defined as:

$$P(\phi_j,\lambda_j,\tau_j|\underline{t},\underline{b}) = \frac{\prod_{i=1}^{N} \Theta\left(b_i|\phi_j,\lambda_j\right)\Phi\left(t_i|\phi_j,\lambda_j,\tau_j\right)P(\phi_j,\lambda_j,\tau_j)}{\iiint \prod_{i=1}^{N} \Theta\left(b_i|\phi_j,\lambda_j\right)\Phi\left(t_i|\phi_j,\lambda_j,\tau_j\right)P(\phi,\lambda,\tau)\,d\phi\,d\lambda\,d\tau}$$

As before, the marginal posterior distribution over location is defined as:

$$P(\phi_j,\lambda_j|\underline{t}) = \int P(\phi_j,\lambda_j,\tau|\underline{t})\,d\tau$$

and the marginal posterior distribution over time defined as:

$$P(\tau_j|\underline{t}) = \iint P(\phi,\lambda,\tau_j|\underline{t})\,d\phi\,d\lambda$$

When using the Bayesian location algorithm, Bloodhound stores the polygon of the coverage ellipse at the user defined `confidence_value` (Appendix A) in the origpoly table. It also stores the parameters of the best approximating ellipse in the origerr table. The locations stored in the origin table include the most probable location (lat, lon) and the location of the center of the ellipse (mean_lat, mean_lon).

Iterative least-squares location

The iterative least-squares inversion method is based on iterative adjustment to a starting location and closely follows the method of Bratt and Bache (1988). The starting location is taken as the geographic mean of the intersection points between all pairs of backazimuths, projected back along their respective great-circle paths. Given an initial estimate of the location given by $\underline{m} = [\phi,\lambda]$, the method iteratively solves for changes to the event location. Based on the derivation in Bratt and Bache (1988), the system of equations solved is:

$$\begin{bmatrix} \left(\theta_1 - \theta_1^p\right) \\ \vdots \\ \left(\theta_n - \theta_n^p\right) \end{bmatrix} = \begin{bmatrix} -\dfrac{\cos a_1}{\sin \Delta_1} & \dfrac{\sin a_1}{\sin \Delta_1} \\ \vdots & \vdots \\ -\dfrac{\cos a_n}{\sin \Delta_n} & \dfrac{\sin a_n}{\sin \Delta_n} \end{bmatrix} \begin{bmatrix} \delta\lambda \\ \delta\phi \end{bmatrix}$$

or, in matrix form:

$$\underline{r} = \underline{A}\Delta\underline{x}$$

where, Δ_i is the great-circle distance between a trial location and the i'th array, a_i is the great-circle azimuth from the trial location to the i'th array, θ_i is the observed backazimuth at the i'th array, θ_i^p is the great-circle backazimuth for the trial location at the i'th array, $\delta\lambda$ is the change in longitude of the trial location, and $\delta\phi$ is the change in latitude of the trial location. The iterative least squares method solves for $\Delta\underline{x}$ using standard least squares inversion.

To estimate the location uncertainty, an a-priori estimate of the combined model and measurement uncertainty, σ_θ (the same as used for Bayesian inversion), is used to calculate the coverage ellipse for a given probability by finding the eigenvalues and eigenvectors of the model variance-covariance matrix:

$$\sigma_m^2 = M\sigma_\theta^2 F[M,N-M]\left(\underline{A}^T\underline{A}\right)^{-1}$$

where $M = 2$ is the number of model parameters, N is the number of arrivals that constitute the events, and $F[M,N-M]$ is the value of the F-statistic with M and $N-M$ degrees of freedom.

2.3. Near-real-time processing

This section describes the implementation of near-real-time processing using Bloodhound.

2.3.1. MALD Procedure 1: Data transforms

For processing data in near-real-time, Bloodhound applies data transforms in user-configurable intervals. Before processing a particular station in an interval, Bloodhound queries the database for data in that interval to see if the data at that station or array exists. If data do not exist, or if the interval is not yet filled in with data, it sleeps for a period of time, before re-querying. The sleep/re-query cycle is repeated `no_retry` times (where `no_retry` is user-configurable) before aborting.

Each interval overlaps by 50%. This mitigates against processing artifacts caused by (a) the cosine taper applied to the data in each interval before filtering (the cosine taper affects the first and last 5% of the interval), and (b) the burn-in time for STA/LTA processing (the burn-in time equals STA+LTA). For each interval, I, Bloodhound saves raw transform results in the center 50% of the interval, as illustrated in Figure 2-9.

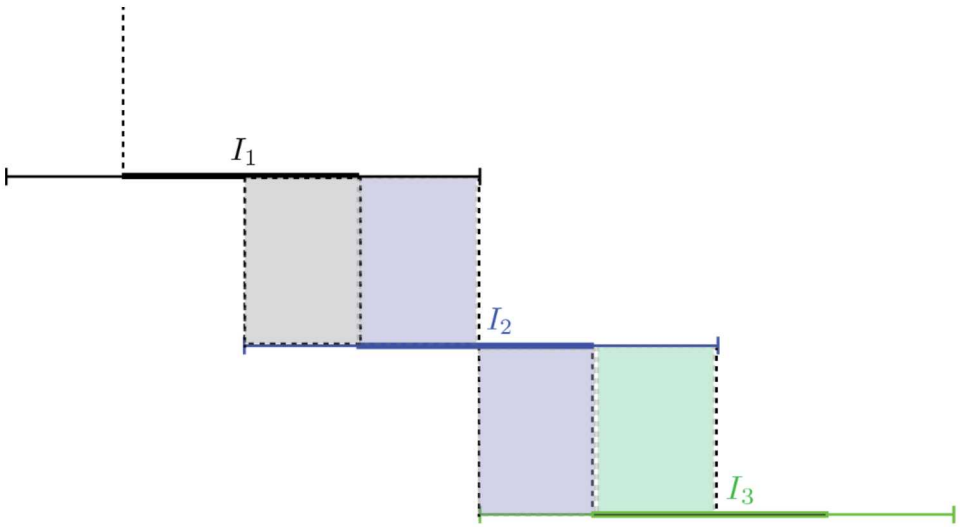

Figure 2-9 Illustration of the overlapping processing intervals used in Bloodhound ($I_1, I_2, and\ I_3$ represent successive intervals) and of the time ranges where data transform results for each time window are saved (colors denote windows associated to respective intervals).

2.3.2. MALD Procedure 2: Secondary transform, KDE estimation, multivariate fusion, and detection

For detection processing, Bloodhound requires a sufficient time history of data transform results in order to build empirical distributions of each detection statistic using KDE estimation. For each processing interval, I, Bloodhound uses data transform results in the prior hour to build an empirical distribution. If starting Bloodhound processing where the prior hour has not been processed, Bloodhound will first process that window before it begins near-real-time processing. This represents a burn-in time for Bloodhound near-real-time processing. However, for each subsequent processing interval, Bloodhound will always utilize the prior history of data transform results.

Because Bloodhound uses overlapping processing intervals for near-real-time processing, Bloodhound implements an additional detection reconciliation step when running in this mode (see Section 2.1.2.6).

2.3.3. Network processing

Network processing is not yet implemented for near-real-time processing. However, the user can always run network processing on near-real-time detection results by applying the same functions described above for archived data to the SQLite3 database containing near-real-time detections.

3. USER-DRIVEN ALGORITHMS

Bloodhound contains an extensive set of additional data processing and modeling tools that are not part of the pipeline processing workflow described in Section 2.

3.1. Data analysis tools

3.1.1. Three-component FK

Bloodhound can produce pseudo-FK plots using three-component data. The FK plots are computed by calculating the three-component power associated with each individual slowness vector over a grid of slowness vectors. For a given slowness vector, $\underline{u} = (u_x, u_y)$, the three-component power is given by:

$$P(u_x, u_y) = \frac{1}{N}\sum_{i=1}^{N}\left(rcos(\theta)e_i + rsin(\theta)n_i + cos(\sin^{-1} r)z_i\right)^2$$

where $\theta = \tan^{-1}\left(\frac{u_x}{u_y}\right)$, $r = v\sqrt{u_x^2 + u_y^2}$ with v as the local sound speed, and $\underline{e}$, $\underline{n}$, and $\underline{z}$ denote the east, north, and vertical components. For a polarized wave, the power is maximized when the slowness vector points in the direction of polarization.

3.1.2. Yield estimation

Bloodhound currently contains two types of yield estimation algorithm: Hydrodynamic scaling and semi-empirical overpressure methods (ANSI and BOOM) (see examples in the companion notebook).

Hydrodynamic scaling equations

Scaling equations for the source-receiver range, r, the pulse duration, t^+, the peak overpressure, Δp, and the impulse, i_{area}, are given by:

$$R = \left(\frac{f_d}{W^{1/3}}\right)r$$

$$T^+ = \left(\frac{f_t}{W^{1/3}}\right)t^+$$

$$\Delta P = \frac{\Delta p}{p_0}$$

$$I_{area} = \frac{i_{area}}{f_d f_t W^{1/3}}$$

where p_0 is the ambient atmospheric pressure,

$$f_d = \left(\frac{p_{obs}}{P_{ref}}\right)^{1/3}\left(\frac{t_{obs}}{T_{ref}}\right)^{-1/3}$$

and

$$f_t = \left(\frac{p_{obs}}{P_{ref}}\right)^{1/3}\left(\frac{t_{obs}}{T_{ref}}\right)^{1/6}$$

scale for ambient pressure and yield given reference pressure P_{ref}=1013 mbar and temperature T_{ref}=15°.

Semi-empirical methods

The ANSI and BOOM equations can be reformulated into the following form:

$$P = A\left(10^{B/106}\right)S^{\alpha}W^{\beta}R^{\gamma}$$

where P is in Pa, A is a constant multiplying factor, B is a meteorological term, S is the ambient meteorological pressure in mbar, W is the charge weight in kg, R is the range in km, and α, β, γ are constants defined in the table below.

Parameter	**ANSI**	**BOOM**
A	0.55200618	0.6630714
B	0	
α	0.556	0.633
β	0.444	0.3667
γ	-1.333	-1.1

3.1.3. Alternative processing functions

The pipeline processing functions described in Section 2 are designed for finding events in data. For the case where there is a known event and the user wants to see if there are any signals observed, Bloodhound includes the event module. The function event.find_signals, searches for signals from a ground-truth event by running the station-level processes in infra.process_data on time-windowed data around the predicted time windows. Results are saved to a database, with two additional tables: gtevent and gtassoc, which contain the ground-truth event information and corresponding associations to detections.

3.1.4. Synthetic tests

Generating synthetic events using propagation modeling

To generate a synthetic event using the Tau-P method, Bloodhound (the synthetic.make_synthetic_event_taup function) calculates ray bounce points given a user-defined event location and a corresponding met file. A plot of bounce points is displayed to the user, allowing them to pick individual bounce points to use as synthetic receivers. For each synthetic receiver, the 'observed' backazimuth is simply taken as $\theta_i^{obs} = \{\vartheta_i + 180°\}$, where ϑ_i is the launch azimuth and $\{\}$ denotes an angle wrap operator (we can do this because the Tau-P method - described in Appendix A - uses an effective sound speed approximation such that the ray is horizontally translated by cross winds but the azimuth remains fixed).

To use the synthetic dataset for location, the infra.locate_event_with_met_corrections function is used to implement the met correction algorithm described in Figure 3-1. Because the wind fields were reversed in calculating synthetic sources for each receiver (Figure 3-1b), the backazimuth that would be observed from each synthetic source equals the launch azimuth used to calculate that source (i.e., $\theta = \vartheta$) (these backazimuths are used to calculate the mean predicted backazimuth in Figure 3-4).

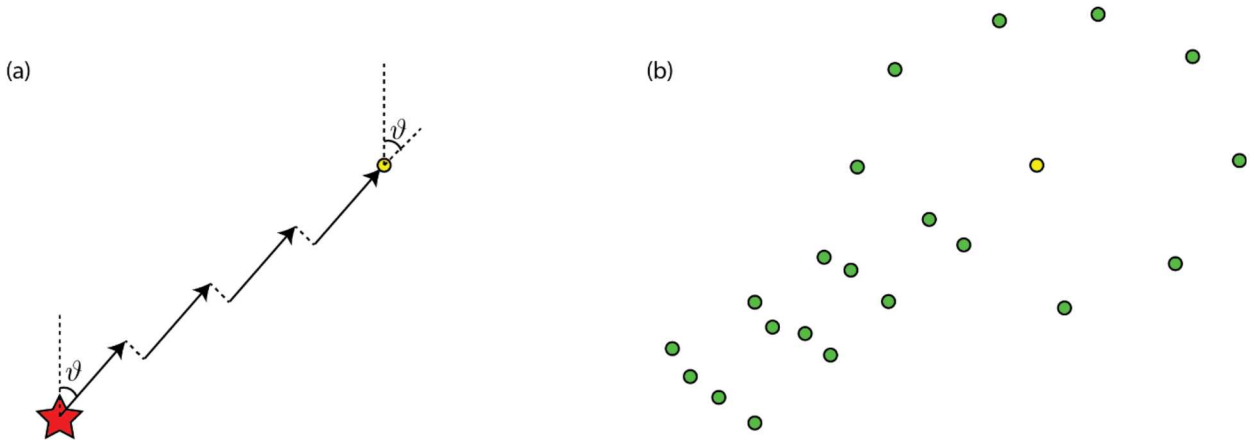

Figure 3-1 Illustration of the creation of a synthetic dataset via two steps. (a) A set of bounce points is calculated from a hypothetical source (red star); selected bounce-points are chosen as synthetic stations (e.g., yellow circle). (b) For each station (yellow circle), rays are shot in a random set of directions (with the wind fields reversed); bounce-points (green circles) are synthetic sources (the backazimuth for each synthetic source equals the launch azimuth used to predict it).

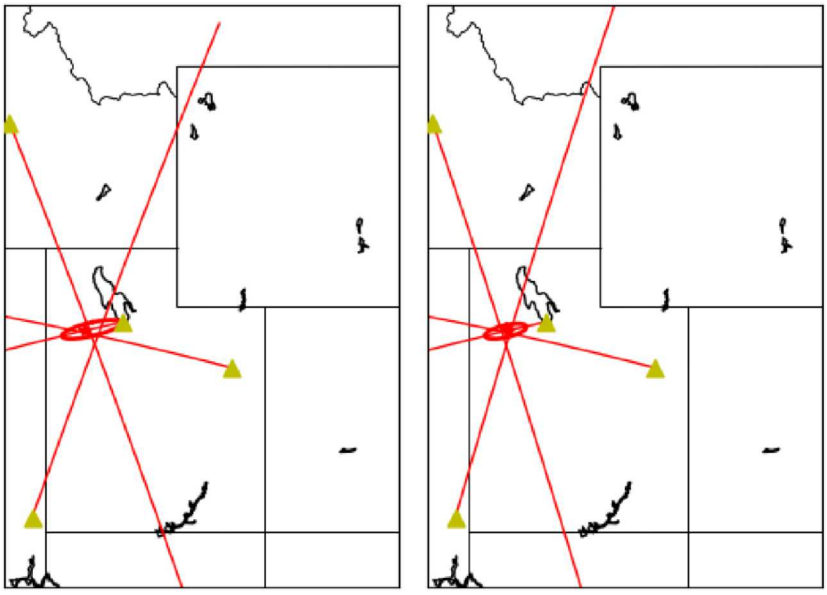

Figure 3-2 An example synthetic event location before (left) and after (right) applying met corrections. The location of the synthetic event is shown by the red star.

An example of applying met corrections to locate a synthetic event is shown in Figure 3-2 (the corresponding notebook is synthetic_simulations.ipynb).

Generating synthetic events without propagation modeling

Bloodhound contains several functions to test network level functions with synthetic data. The SyntheticEvent_examples.ipynb notebook contains details on how to run such functions.

3.1.5. Plotting functions

Bloodhound contains a large set of plotting functions including standard signal processing tasks (spectrograms, spectra, record sections) as well as more specialized tasks (PMCC plots). See the RealEvent_examples.ipynb notebook for examples.

3.2. Propagation modeling

Bloodhound includes a native implementation of the Tau-P equations for calculating ray paths in a moving medium. Because Tau-P is the fastest method for calculating possible source locations given a receiver, it forms the basis of the meteorological corrections in Bloodhound. However, Bloodhound also provides interfaces to two community propagation codes: the NCPAProp code

(https://github.com/chetzer-ncpa/ncpaprop) and the GeoAc code (https://github.com/LANL-Seismoacoustics/GeoAc), making these codes more accessible and directly useful to the analyst. A description of the equations solved in these external packages is described in the respective distributions.

Bloodhound has a suite of functions to run Tau-P and interface with external packages. See the bloodhound_examples.ipynb notebook for examples of running Tau-P. A short description of the functions is provided below.

Bloodhound contains a native implementation of the Tau-P equations for calculating ray paths in a moving medium. This is the fastest method for calculating ray bounce points given a source, or possible source locations given a receiver, and is therefore the basis of meteorological corrections in Bloodhound.

Tau-P uses the effective sound speed approximation. For a particular launch azimuth, the effective sound speed profile is constructed from the temperature and wind profiles. For a given launch angle, the ray parameter is estimated:

$$p = \frac{k_z}{c_0}\left(1 + \frac{k_z u_0}{c_0}\right)^{-1}$$

where $k_z = \sin\varphi$ is the vertical wave number derived from the launch angle, c_0 is the static sound speed at the receiver, and u_0 is the horizontal wind velocity at the ground. Tau-P solves the following equations for the range (R), transverse offset (Q) and travel-time (T) of the bounce point:

$$R(z,p) = 2\int_{z_0}^{z(p)} \psi(z,p)\left[\frac{p}{1-u(z)p} + u(z)\zeta(z)\right]dz$$

$$Q(z,p) = 2\int_{z_0}^{z(p)} \psi(z,p)\zeta(z)v(z)dz$$

$$T(z,p) = 2\int_{z_0}^{z(p)} \psi(z,p)\zeta(z)dz$$

with,

$$\psi(z,p) = \left[\zeta(z) - \frac{p^2}{(1-u(z)p)^2}\right]^{-1/2}$$

and,

$$\zeta(z) = \frac{1}{c^2(z)}$$

To solve these equations, Tau-P finds the first root of $\psi(z,p)$ above z_0, then the bounce point can be readily calculated.

As described in these equations, Tau-P is based on a range/transverse-offset (R,Q) coordinate system. This coordinate system is suitable because Tau-P is based on an effective sound speed approximation (wind effects are accounted for by converting the wind vectors into along-path and cross-path components). Bloodhound converts from R,Q to geographic coordinates (ϕ, λ) via a coordinate transformation, as described in Figure 3-3.

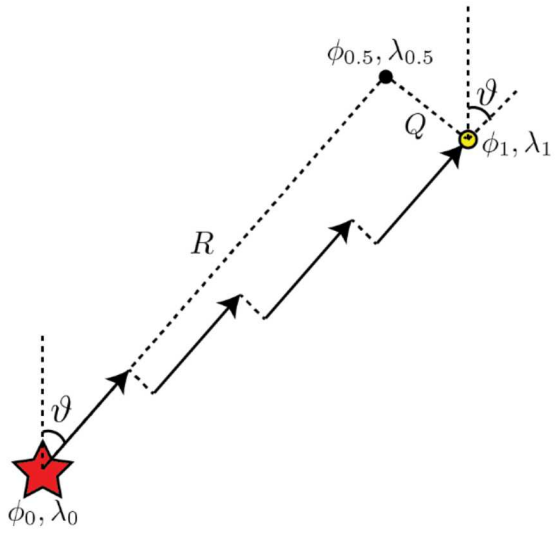

Figure 3-3 An illustration of the coordinate transformation in Bloodhound from R,Q to ϕ, λ coordinates.

The coordinate of the bounce point (yellow circle in Figure 3-3) is calculated by first calculating the position of $\phi_{0.5}, \lambda_{0.5}$ given the starting location (ϕ_0, λ_0), azimuth (ϑ), and great-circle distance (R). Next, the location of the bounce point is calculated given a second starting position, $(\phi_{0.5}, \lambda_{0.5})$, azimuth ($\{\vartheta - 90°\}$ if Q is positive or $\{\vartheta + 90°\}$ otherwise), and great-circle distance (Q).

The azimuthal deviation is given by:

$$\Delta\vartheta = arctan\left(\frac{Q}{R}\right)$$

where $\Delta\vartheta$ is equal to the difference between the launch azimuth and the azimuth of the actual bounce-point.

3.2.1. Applying meteorological corrections

Meteorological corrections are currently applied to backazimuths given an initial event location. They are applied for a given ORID in a Sqlite3 database using ray predictions obtained from the

Tau-P method (see the examples in the companion notebook). The process of correcting the observations based on an initial event location is illustrated in Figure 3-4.

First, as shown in Figure 3-4a, a maximum likelihood location is calculated using the observed backazimuths without any meteorological corrections. A threshold distance defines a circle of that radius centered on the maximum likelihood location. Next, as shown in Figure 3-4b, ray predictions for each station, which are obtained by reversing the winds and shooting backwards from the station, are collected for possible sources inside the circle defined in Figure 3-4a. The predicted backazimuths of these hypothetical sources, as observed at a given array, are averaged to obtain a mean predicted backazimuth at that array. Next, as shown in Figure 3-4c, the difference between the mean predicted backazimuth (given the initial location) and the great-circle backazimuth defines a correction to convert observed to great-circle backazimuths. Finally, as shown in Figure 3-4d, the correction is applied to each observed backazimuth to correct them for wind bias. The location method described in the previous section is subsequently applied to the corrected backazimuths to obtain the likelihood function and corresponding posterior distribution.

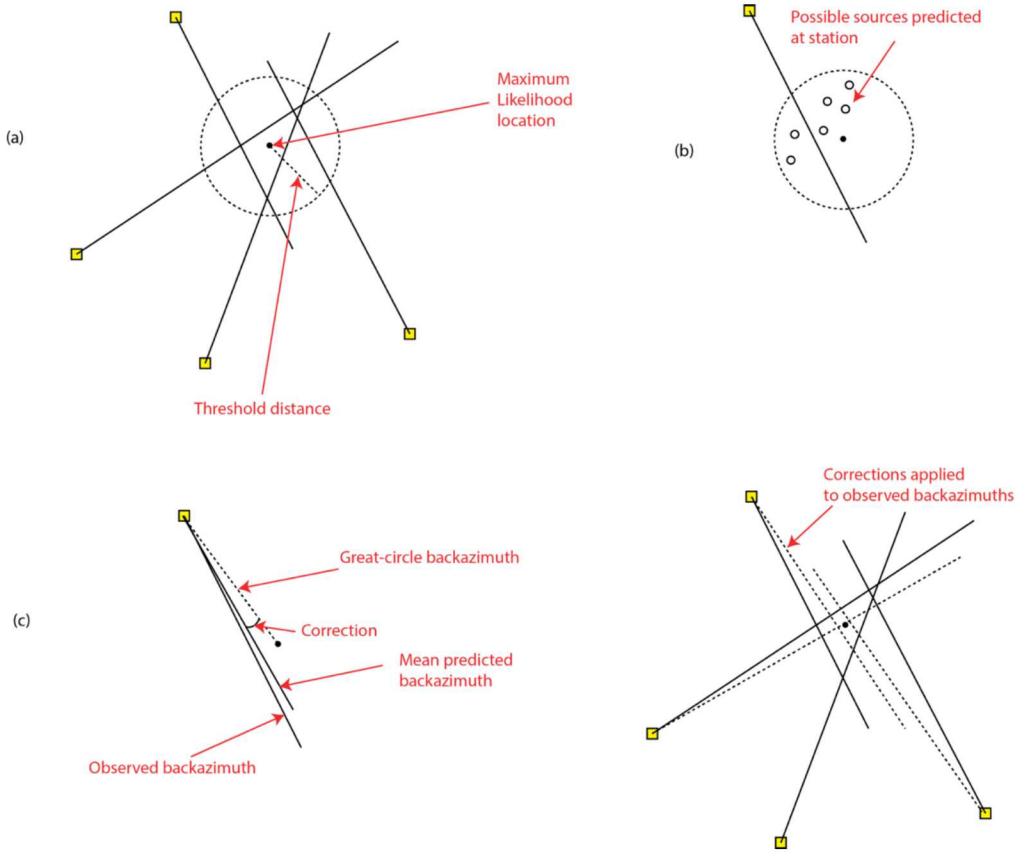

Figure 3-4 Cartoon illustrating the process used by Bloodhound to correct observed backazimuths based on a meteorological model and associated ray shooting predictions using the Tau-P method in Bloodhound.

3.2.2. Running Tau-P

The user can directly run Tau-P for a given met file (an ASCII file that contains a 1D profile of the atmosphere with the following parameters: Z (km), T (K), U (m/s), V (m/s), d (g/cm^3), p (mbar). An example of running Tau-P is provided in the companion notebook file.

3.2.3. Running GeoAc

Bloodhound contains functions to run GeoAc to produce N-by-2D maps of bounce point predictions, and to run GeoAc to compute and plot eigenrays. To use these functions, the user must install GeoAc on their system (this is not provided with Bloodhound but is available at https://github.com/LANL-Seismoacoustics/GeoAc).

3.2.4. Running NCPAProp

Bloodhound contains functions to run NCPAProp to produce N-by-2D maps of transmission loss (and of pressure for explosions of known yield), as well as profiles of predicted transmission loss along fixed azimuths. To use these functions, the user must install NCPAProp on their system (this is not provided with Bloodhound but is available at https://github.com/chetzer-ncpa/ncpaprop).

4. ACKNOWLEDGEMENTS

Several people have provided important input on this document. Useful algorithm input and review was provided by Steven Taylor, Paul Nyffenegger, Chris Young, and Brian Young.

5. REFERENCES

Arrowsmith, S., C. Young, S. Ballard, M. Slinkard, and K. Pankow (2016). Pickless Event Detection and Location: The Waveform Correlation Event-Detection System (WCEDS) Revisited, Bull. Seism. Soc. Am. 106 2037 - 2044.

Arrowsmith, S., and D. C. Bowman (2017). Explosion yield estimation from pressure wave template matching, J. Acoust. Soc. Am. 141 doi:10.1121/1.4984121.

Blom, P., O. Marcillo, and S. Arrowsmith (2015). Improved Bayesian Infrasonic Source Localization for regional infrasound, Geophys. J. Int. 203 1682 - 1693.

Bratt, S. R., and T. C. Bache (1988). Locating Events with a Sparse Network of Regional Arrays, Bull. Seism. Soc. Am. 78 780 - 798.

Douze, E. J., and S. J. Laster (1979). Statistics of semblance, Geophysics 44 1999-2003.

Jurkevics, J. (1988). Polarization analysis of three-component array data, Bull. Seism. Soc. Am. 78 1725 - 1743.

Modrak, R. T., S. Arrowsmith, and D. Anderson (2010). A Bayesian framework for infrasound location, Geophys. J. Int. 181 399 - 405.

Rost, S., and C. Thomas (2002). Array Seismology: Methods and Applications, Rev. Geophys. 40.

Withers, M., R. Aster, C. Young, J. Beiriger, M. Harris, S. Moore, and J. Trujillo (1998). A Comparison of Select Trigger Algorithms for Automated Global Seismic Phase and Event Detection, Bull. Seism. Soc. Am. 88 95 - 106.

APPENDIX A: INPUT PARAMETERS

The input parameters are provided as a TOML file, with the table and key names explained below.

Table	*Key*	*Description*	*Examples (if applicable)*
[data]	type	Type of data	"oracle" "css3" "pickle" "obspy"
	file	File name of data file (only required if type is "css3" or "pickle")	
	conn_str	Connection string for Oracle database (only required if typs is "oracle")	
[time]	start	Start date and time of data to process	2007-08-01T19:30:00+00:00
	end	End date and time of data to process	2007-08-01T20:30:00+00:00
[[filter]]	low	Low-pass corner frequency (Hz)	
	high	High-pass corner frequency (Hz)	
	window_length	Window length for FK processing	
[[station-data]]	name	Station or array name	
	lat	Station latitude (or reference array latitude)	
	lon	Station longitude (or reference array longitude)	
	channels	Channels to process	"*" "*DF" "BH?"
	elements	Elements to process (if array) or station name (if single station)	["NOQ3","NOQ4","NOQ5"]
[[station-detection]]	name	Station name	
	signal_type	Type of signal to detect	"regional" "local-static" "local-moving"
	p_thres	Detection threshold (p-value)	
	minimum_duration	Minimum signal duration	
	sta	STA window length (s)	
	lta	LTA window length (s)	
[[network]]	lat_min	Minimum latitude of region to monitor	
	lon_min	Minimum longitude of region to monitor	
	lat_max	Maximum latitude of region to monitor	
	lon_max	Maximum longitude of region to monitor	
	azimuth_dev	Allowed azimuthal deviation for association (degrees)	
	confidence_value	Confidence value for coverage ellipse	0.95 (for 95% coverage ellipse)

APPENDIX B: OUTPUT DATABASE SCHEMA

The output results are stored to a SQLite3 database, with the table and column names explained below.

Table Name	Column Name	Type	Description
fk	array	text	array name
	time	real	start of time window
	semblance	real	semblance estimate for time window
	backazimuth	real	backazimuth estimate for time window
	trace_velocity	real	trace velocity estimate for time window
	bandid	integer	Unique frequency band ID
sta_lta	array	text	array name
	time	real	time
	sta_lta	real	STA/LTA value
	bandid	integer	Unique frequency band ID
sta	array	text	array name
	time	real	time
	sta	real	STA value
	bandid	integer	Unique frequency band ID
polarization	array	text	array name
	time	real	Start of time window
	rectilinearity	real	Rectilinearity estimate for time window
	azimuth	real	Azimuth estimate for time window
	inclination	real	Inclination estimate for time window
	bandid	integer	Unique frequency band ID
detect	arid	integer	unique arrival ID
	master_arid	integer	Master arrival ID
	array	text	array name
	start_time	real	start time of detection
	end_time	real	end time of detection
	bandid	integer	Unique frequency band ID
	backazimuth	real	average backazimuth of detection
	b_start	float	Starting backazimuth
	b_end	float	Ending backazimuth
	t_vel	real	Trace velocity (km/s)
	semblance	real	Peak semblance during detection (-999 if null)
	sta_lta	real	Peak STA/LTA during detection (-999 if null)
	sta	real	Peak STA during detection (-999 if null)
	rectilinearity	real	Peak rectilinearity during detection (-999 if null)
	p_value	real	Minimum p-value during detection
	type	integer	Detection type (0=static, 1=moving)
metadetection	met_id	integer	Unique metadetection ID
	start_time	real	Start time
	end_time	real	End time
	min_freq	real	Minimum frequency (Hz)
	max_freq	real	Maximum frequency (Hz)
	mean_freq	real	Mean frequency (Hz)
	mean_semblance	real	Mean semblance
	std_semblance	Real	Standard deviation of semblance
	mean_backazimuth	Real	Mean backazimuth (degrees)
	std_backazimuth	Real	Standard deviation of

			backazimuth (degrees)
	mean_t_vel	Real	Mean trace velocity (km/s)
	std_t_vel	real	Standard deviation of trace velocity (km/s)
assoc	orid	integer	unique event ID
	arid	integer	unique arrival ID
origin	orid	integer	unique event ID
	lat	real	Maximum Posteriori latitude of event
	lon	real	Maximum Posteriori longitude of event
	lat_mean	real	Mean latitude of event
	lon_mean	real	Mean longitude of event
	t0	real	estimated origin-time of event
origerr	orid	integer	unique event ID
	theta	real	azimuth of error ellipse
	maj	real	length of semi-major axis
	min	real	length of semi-minor axis
origpoly	orid	integer	unique event ID
	lat	real	latitude point on polygon
	lon	real	longitude point on polygon
f_band	bandid	integer	Unique frequency band ID
	fmin	real	Minimum frequency of bandpass Butterworth filter
	fmax	real	Maximum frequency of bandpass Butterworth filter

DISTRIBUTION

1 US Department of Energy
Attn: Leslie A. Casey (1)
NA-222, 1000 Independence Avenue, S.W.
Washington, DC 20585

2 MS0899 Technical Library 9536 (electronic copy)

Sandia National Laboratories