Locating and Identifying Bats Based on Their Echolocation Cries

A Co-op Work Term Report

Submitted to:
Dr. Pierre Zakarauskas

Submitted by:
Daniel L. Lu
UBC Student Number:
75592063

Date Submitted:
1 May 2010

Summary

This report details several methods employed to detect and determine the location and species of different bats based on acoustic data recorded on four microphones. By determining the signal-to-noise ratio and analyzing the spectrogram of the data, it was possible to accurately detect bat cries. Moreover, by cross-correlating spectrograms across four channels, the times-difference-on-arrival across all four channels could be determined, allowing the calculation of the three-dimensional coordinates of the bat's position through multilateration. By comparing contour curves of spectrograms at several signal-to-noise ratio thresholds, we could determine a degree of correlation between an unknown cry and one emitted by a confirmed species, enabling the identification of species.

Table of Contents

List of Figures

1. Introduction

Since structures such as wind turbines may pose a serious hazard to bats due to barotrauma (Baerwald, D'Amours, Klug, & Barclay, 2008), it is important to conduct an environmental assessment survey in the area prior to constructing these structures to determine the risk of damaging wildlife. To that end, it is useful to be able to detect, locate, and identify bats based on purely acoustic data. Using only acoustic data has the advantage of functioning even in the darkness of night, when bats are most likely to be found. In addition, analysing acoustic data can also yield the species of the bat, something which other technologies like radar cannot do.

To develop algorithms to perform these tasks, we used the MATLAB programming language to write scripts. We then tested the algorithms on previously recorded multi-channel raw data from four microphones.

Each MATLAB function described in this report may have several versions, based on the history of how the function was developed. This report only describes the latest version of the functions at the time of writing. Function versions are numbered, with the largest number being the latest and best version. For example, when this report refers to `findCries`, it is referring to `findCries3`, which is the latest version.

2. Detecting Bats

The detection of cries emitted by bats may be considered a classic signal-detection problem. These cries are typically very distinct – as such, an algorithm can be implemented to discern them from noise and other artefacts. The method employed to detect bats is broken into several passes to accurately identify bat cries without giving false positives.

2.1 FindCries

The first pass, known by the function name `findCries`, detects areas of high signal-to-noise ratio by first calculating a spectrogram from the data. To ensure that the spectrogram is uniform along the vertical frequency axis, it also calculates an array known as `noise` by taking the x-axis median for each ordinate value. The reason why `noise` is needed is illustrated in Figure 1. Note how the strong artefacts at 0-20 kHz are greatly neutralised.

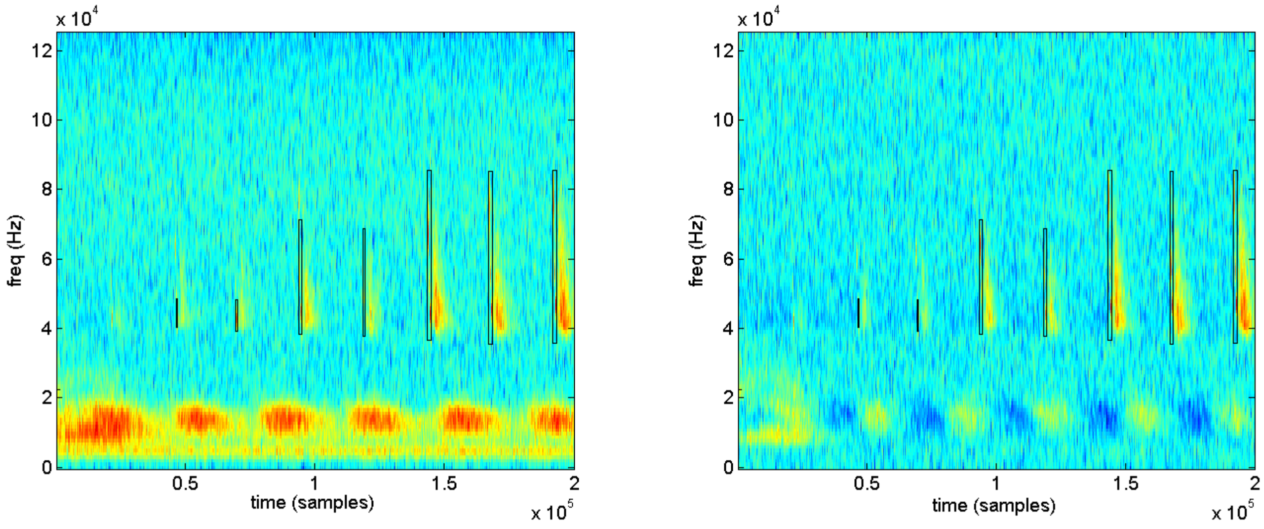


Figure 1 Left: Spectrogram before subtracting `noise`; Right: After subtracting `noise`.

In addition, the spectrogram is smoothed in two dimensions using a Hanning mask to mitigate noise; the amount of smoothing done is governed by the variable `smoothingSize`, which is sometimes also referred to as `smoothSize1`. Having thus ensured the uniform background noise for the spectrogram, the algorithm proceeds to identify areas where the spectrogram exceeds a certain threshold known as `SnrThresh1`. It does so by making use of a window sliding along the time axis to look for a grouping of events above this threshold. This window only exists in frequencies above `minFreq`, so it ignores any signals that are lower than that. As it slides from left to right, if it encounters any such event for which the start time is a period of time less than `minGap` behind the end time of the previous cry, it would add the event to the previous cry by extending the end time of the previous cry. If the gap between an event and the previous one is more than `minGap`, it is declared a new cry. Cries that are shorter than `minDuration` or longer than `maxDuration` are discarded.

Then, for each detected cry, the algorithm finds the cry's maximum frequency and minimum frequency, known as `cryMaxFreq` and `cryMinFreq`. This is achieved by analyzing the spectrogram for a single cry along the frequency axis to find where it exceeds `SnrThresh1`.

Ultimately, the function `findCries` outputs `numCries`, `cryMinFreq`, `cryMaxFreq`, `cryStart`, `cryStop`, `medianSnr`, and `noise`. It is useful for `noise` to be outputted by the function because it may be needed later by `findFMsweeps` or other routines that require spectrograms. The variable `numCries` represents the number of cries detected by `findCries`. The arrays `cryMinFreq`, `cryMaxFreq`, `cryStart`, and `cryStop` are all one-dimensional arrays for which the length is equal to `numCries`. For the nth cry, its minimum frequency, maximum frequency, start time, and stop time are given by `cryMinFreq(n)`, `cryMaxFreq(n)`, `cryStart(n)`, and `cryStop(n)`, respectively.

2.2 FindFMSweeps

To understand why a second pass is needed, we must first consider the physical properties of the echolocation cry used by bats of interest. Although `findCries` is generally excellent at finding the cries, the presence of an *echo* behind certain cries may cause it to trigger false alarms or give an imprecise estimate of `cryStop`. Because the cry itself is a frequency-modulated sweep emitted by the bat, this second pass is known as `findFMsweeps`.

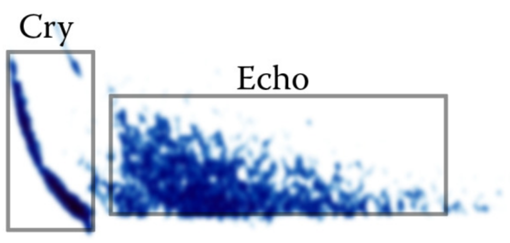


Figure 2 A cry emitted by *Myotis lucifugus*.

The second pass analyzes the output of the first pass at several signal-to-noise ratio thresholds, stored in a vector called `SnrThresh2`. At each of these thresholds, the algorithm calculates the contours or level curves of the spectrogram.

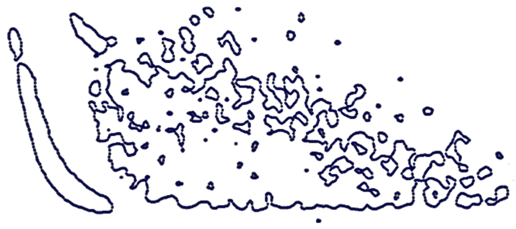


Figure 3 Contours of a *Myotis leibii* cry at a threshold of 8 dB.

For each `SnrThresh2` level, each contour is subject to many tests that are designed to filter out spurious contours, particularly those in the echo. To do this, the function `unpackContour` is called, to extract each closed shape as a polygon. Contours that are holes within other contours are identified and eliminated through use of the function `topoContour2`. Then, three properties calculated by the function `analyzeContour2` are used. The first property is the circumference. In general, contours that have a circumference greater than `circumThresh` would pass the test. The second property, `Theta`, represents the general slope of the contour as an angle, and is calculated from segments on the polygon; the said segments' lengths are governed by the variable `segLen`. FM sweeps are always sloped downwards. To filter out cries that are not sloped downwards, the algorithm calculates a value known as `SlantRatio` using `Theta` and a variable `thetaBoundary`. Only cries for which `SlantRatio` exceeds a parameter known as `ratioThresh` pass the test. The third property is `Compactness`, and is the ratio of the circumference to the area. Ironically, the lower the `Compactness`, the more compact the shape. The lowest possible `Compactness` is achieved by a circle. Since cries usually take the shape of a crescent, only those with `Compactness` above a certain limit known as `compactThresh` would be accepted.

The function `findFMsweeps` is repeatedly called by the function `applyFMsweeps` in order to find the cries at several `SnrThresh2` values. Moreover, `applyFMsweeps` only calls `findFMsweeps` on the areas identified by `findCries`. There are many such areas, so `findFMsweeps` may be called many times.

The reason why several `SnrThresh2` values are needed is because there may be some extremely strong cries for whose contours may connect with the contours of their echos or other noises at lower `SnrThresh2` levels. In that case, the lower threshold contours would not pass the tests, whereas a contour generated at a higher `SnrThresh2` level would.

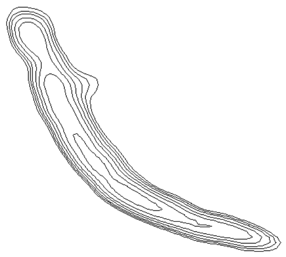


Figure 4 Contours at several threshold values of a *Myotis leibii* cry that pass the tests in `findFMsweeps`.

The output of `applyFMsweeps` is the accumulation of the outputs of all the instances of `findFMsweeps` that are called. These are `numCries2`, `cryStart2`, `cryStop2`, `cryMinFreq2`, `cryMaxFreq2`, and `slantRatio`. The first five outputs have the same purposes and structures as those in `findCries`, except they are more refined. The last output is a one-dimensional array whose size is `numCries2`, containing the `slantRatio` of each cry. It may be useful in identifying the bat's species, though an algorithm to do so has not been developed yet.

2.3 Detection of Weak Cries

The synthesis of `findCries` and `findFMsweeps` is capable of detecting cries that are strong enough to have a recognizable sweep. However, there may be very weak cries that would not pass the tests in `findFMsweeps`. It was mentioned before that the main drawback of `findCries` is that echos may distort the output; however, for very weak cries, the echo is not strong enough to be detected by the microphone. As such, it suffices to call `findCries` again with a different set of parameters to detect weak cries. These parameters are optimized to find weak cries and not strong ones by using a low value for `maxDuration`, a higher value for `minFreq`, and so on. Parameters used for detecting weak cries have the prefix '`W`' in their identifiers – for example, the `minFreq` for detecting weak cries is known as `WminFreq`.

2.4 MergeCries

Occasionally, a cry may be broken into two, or have distinct harmonics. In that case, more than one contour might be identified by `findFMsweeps` for a single cry.

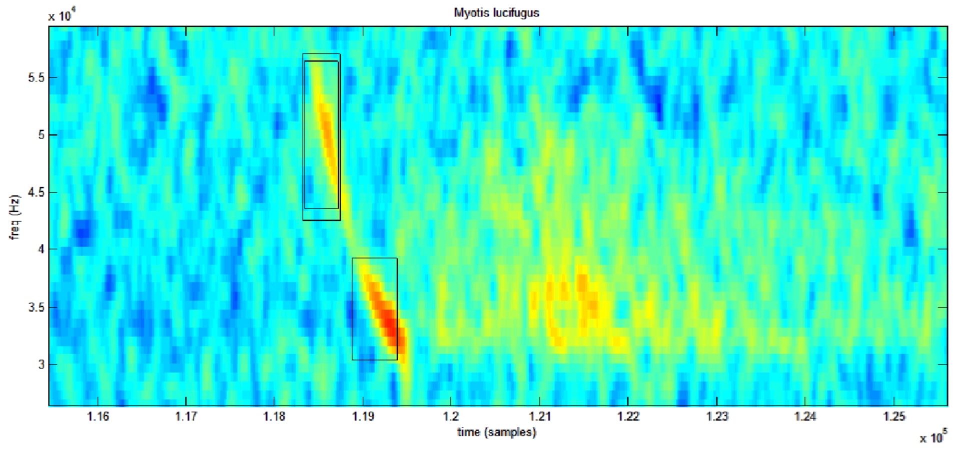


Figure 5 A cry emitted by *Myotis lucifugus*. Cries detected by `findFMsweeps` are boxed.

The function `mergeCries` remedies the problem by merging any cries that are closer together than the threshold `mergeThresh`. Cries are declared as separate cries only if they are further apart than `mergeThresh2`. If there are two cries whose distance from each other lies between `mergeThresh` and `mergeThresh2`, then the second cry is ignored as it is most likely to be an echo. If cries are to be thought of as boxes in a spectrogram, then merged cries take the form of the minimum bounding box of all the cries that were merged together to form it – i.e. the `cryMinFreq` of a merged cry is the minimum of all the `cryMinFreqs` of the cries that were merged together to form it, the `cryMaxFreq` is the maximum of all the `cryMaxFreqs`, the `cryStart` is the start of the first cry of those that were merged together, and the `cryStop` is that of the last.

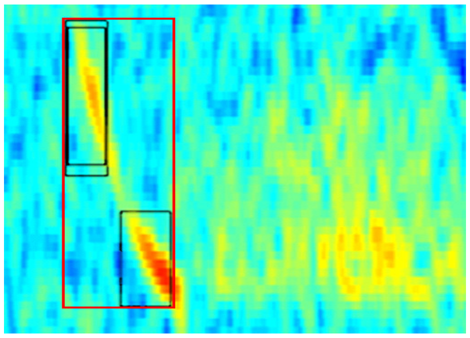


Figure 6 The cry in Figure 5, showing output of `mergeCries` (red).

In addition to merging broken cries and removing echos, the function `mergeCries` has the integral role of merging the results of weak cry detection and strong cry detection. As such, it is only called after all the previous cry detection passes.

2.5 DetectCries

All the cry detection functions described so far are utilized in the function `detectCries`, which takes as input the time series `data`, plus all the parameters needed for cry detection, and then invokes a combination of `findCries`, `applyFMsweeps`, and `mergeCries`. Ultimately, the purpose of `detectCries` is to make it possible to detect cries just by using one line of code, thus saving space in bigger programs wherein cries need to be detected several times or at many places.

3. Locating Bats

Since there are four microphones, it is possible to calculate the location of bats in three dimensions by discerning the times-difference-on-arrival (TDOA) of signals across the different channels. The mathematics of multilateration is greatly simplified by placing one of the microphones, which we call `mic0`, on the origin of the coordinate system.

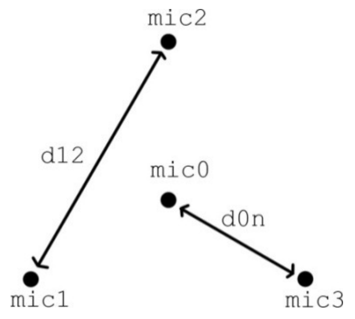


Figure 8 Relative placement of microphones.

Microphones `mic1`, `mic2`, and `mic3` are on an equilateral triangle centred on mic0. The distance of any other microphone to `mic0` is `d0n`; the length of the edge of the triangle is `d12`.

3.1 Summary of Multilateration Mathematics

The problem of finding the coordinates of an emitter based on the relative times at which the signal was received on multiple detectors can be reduced to a solvable system of equations (Bucher & Misra, 2002). Let there be an emitter (i.e. a bat) at $(x_{bat}, y_{bat}, z_{bat})$. The the distance D_0 and time t_0 taken to reach `mic0`, which is centered on $(0,0,0)$, is given by

$$D_0 = ct_0 = \sqrt{x_{bat}^2 + y_{bat}^2 + z_{bat}^2}$$

where c is the speed of sound through air. The time t_n taken to reach any of the other microphones is given by

$$D_n = ct_n = \sqrt{(x_{bat} - x_n)^2 + (y_{bat} - y_n)^2 + (z_{bat} - z_n)^2}$$

where x_n, y_n, and z_n are the coordinates of the microphone. Let us consider the time-difference-on-arrival (TDOA) for one of the three outer microphones, τ_n:

$$c\tau_n = c(t_n - t_0) = D_n - D_0$$

Rearranging and squaring the above, we obtain:

$$D_n^2 = (c\tau_n + D_0)^2$$

Rearranging and expanding the above, we obtain:

$$0 = (c\tau_n)^2 + 2c\tau_n D_0 + D_0^2 - D_n^2$$

Dividing by $c\tau_n$, we obtain:

$$0 = c\tau_n + 2D_0 + \frac{D_0^2 - D_n^2}{c\tau_n}$$

Let us for now restrict n to 2 or 3, as we compare the case for `mic1` with `mic2` or `mic3`.

$$0 = c\tau_n + 2D_0 + \frac{D_0^2 - D_n^2}{c\tau_n}$$

$$0 = -c\tau_1 - 2D_0 - \frac{D_0^2 - D_1^2}{c\tau_1} 0 = c\tau_n - c\tau_1 + \frac{D_0^2 - D_n^2}{c\tau_n} - \frac{D_0^2 - D_1^2}{c\tau_1}$$

Let us consider D_n^2. Expanding, we obtain:

$$\begin{aligned} D_n^2 &= x_n^2 + y_n^2 + z_n^2 - 2x_{bat}x_n - 2y_{bat}y_n - 2z_{bat}z_n + x_{bat}^2 + y_{bat}^2 + z_{bat}^2 \\ &= x_n^2 + y_n^2 + z_n^2 - 2x_{bat}x_n - 2y_{bat}y_n - 2z_{bat}z_n + D_0^2 \therefore D_0^2 - D_n^2 \\ &= -x_n^2 - y_n^2 - z_n^2 + 2x_{bat}x_n + 2y_{bat}y_n + 2z_{bat}z_n \end{aligned}$$

Combining this result with the previous equation, we obtain:

$$\begin{aligned} 0 = c\tau_n - c\tau_1 &+ \frac{-x_n^2 - y_n^2 - z_n^2 + 2x_{bat}x_n + 2y_{bat}y_n + 2z_{bat}z_n}{c\tau_n} \\ &- \frac{-x_1^2 - y_1^2 - z_1^2 + 2x_{bat}x_1 + 2y_{bat}y_1 + 2z_{bat}z_1}{c\tau_1} \end{aligned}$$

Note that the above equation takes the form of below:

$$0 = x_{bat}A_n + y_{bat}B_n + z_{bat}C_n + D_n A_n = \frac{2x_n}{c\tau_n} - \frac{2x_1}{c\tau_1} B_n = \frac{2y_n}{c\tau_n} - \frac{2y_1}{c\tau_1}$$

$$C_n = \frac{2z}{c\tau_n} - \frac{2z_1}{c\tau_1}$$

$$D_n = c\tau_n - c\tau_1 - \frac{x_n^2 + y_n^2 + z_n^2}{c\tau_n} + \frac{x_1^2 + y_1^2 + z_1^2}{c\tau_1}$$

for each case where n is 2 or 3. As such, this forms a set of homogeneous linear equations, demonstrating that it is mathematically possible to derive the coordinates using three TDOA values generated from four microphones.

In the MATLAB algorithm that we are using, τ_1, τ_2, and τ_3 are known respectively as `dt1`, `dt2`, and `dt3`.

3.2 Cross-Correlation

In order to accurately find the TDOA, a method more precise than using `cryStart` or `cryStop` must be used. The method known as cross-correlation involves sliding two signals across each other to see where they overlap the most. Because of the great amount of noise, it is impractical to perform this on the time series data; as such, the cross correlation is done on the spectrograms of the two signals. This yields the additional advantage of being able to analyze for Doppler shift.

For maximum efficiency, the cross-correlation should only be done to the extent where it is possible. The maximum amount of difference in detection times between `mic0` and any of the other microphones is physically limited by `maxDelay0`. The maximum amount of difference in detection times between any of the other microphones is limited by `maxDelayn`. The algorithm only performs cross-correlation on cries for which their `cryStart` times satisfy this criterion; matching cries are indexed in the array `Imatch`. Moreover, the sliding of the spectrograms only takes place within physically possible limits.

Vertical sliding of the spectrograms can account for frequency shift. Since there is a limit to how fast bats can fly, the Doppler shift is limited to a fixed amount; hence, vertical cross-correlation is also limited.

3.3 3D Distance Matrix

Having thus found the TDOA, a computationally inexpensive method must be used to determine the coordinates of the bat's location. The algorithm calculates a four-dimensional array known as the 3D distance matrix, `Dt`. For any set of coordinates `x`, `y`, and `z`, the distance matrix stores the TDOA values as `dt1=Dt(x,y,z,1)`, `dt2= dt1=Dt(x,y,z,2)`, and `dt3=Dt(x,y,z,3)`. To convert the coordinates to physical units, there are three one-dimensional arrays `X`, `Y`, and `Z` so that the coordinates in meters on a coordinate system centered on mic0 are `X(x)`, `Y(y)`, and `Z(z)`.

This matrix `Dt` is of course of finite resolution, and hence to find the best estimate for `x`, `y`, and `z`, there must be a search algorithm to find the closest TDOA triplet in `Dt`. This involves calculating another three-dimensional matrix `E` that contains the sum of the squares of the differences between the detected TDOA and every TDOA value inside `Dt`. The coordinates `x`, `y`, and `z` are obtained by finding the minimum of `E`.

It is perhaps easier to think of `Dt` as dividing the three-dimensional space into little rectangular prisms (i.e. volumetric cells or "pixels"), and then finding which one the bat is most likely to be in. The dimensions, in meters, of `Dt` is governed by `maxX`, `maxY`, and `maxZ`; and the size of each volumetric cell, also in meters, is given by `dx`, `dy`, and `dz`.

The function `distMatrix` generates `Dt`, `X`, `Y`, and `Z`, taking as input the positions of the microphones, `mic1x`, `mic1y`, `mic2x`, `mic2y`, `mic3x`, and `mic3y`; the speed of sound; `maxX`, `maxY`, and `maxZ`; and `dx`, `dy`, and `dz`. It also outputs the number of cells in each dimension.

4. Identifying Bats

The cries emitted by different species may be distinguished by the shape of the spectrogram, the minimum frequency of the cry (the maximum frequency is usually lost due to attenuation in the air), and the time between cries. The algorithm mainly tries to match the shape of the spectrogram of an unknown cry against that of verified templates, though `cryMinFreq` and time between cries is also important in the algorithm's decision.

4.1 Identification Based on Shape

It is impractical to match the spectrograms of the cries directly, since spectrograms are three-dimensional (time, frequency, and intensity). The cries have a wide range of intensity, and the algorithm must not be biased based on intensity. Hence, it is better to focus upon the contours of the spectrogram, just like in `findFMsweeps`. For each cry that has been found by `detectCries`, several contours of the relevant spectrogram are calculated at different thresholds determined by the array `SnrThresh3`. The many levels of `SnrThresh3` eliminate the problem of different loudness of cries, as well as the problem of "broken" cries, since it is most likely that at least one threshold level will produce a contour that is not broken.

As with `findFMsweeps`, there is the problem of eliminating spurious contours that arise from noise and the "echo". However, this problem is greatly simplified with the knowledge that a cry actually exists in the spectrogram – rather than trying to verify whether there is at all a cry, we can now simply take the largest contour for each `SnrThresh3` value. Most often, the largest contour is the cry itself that has been detected by `detectCries`. As such, a set of relevant contours are generated for each cry.

First, such a set of contours are generated for some cries for which the species is already known. These will be used as templates against which the contours of unknown cries will be compared. The templates are chosen by a human so as to give the most precise identification of cries, though `detectCries` is still executed on them to ensure that the algorithm can detect these cries.

The algorithm compares each unknown cry with several templates by comparing every possible combination of different `SnrThresh3` levels, and then outputting the best match for each template. The template for which the best match is the highest is most likely to be the same species as the unknown cry.

For each combination of `SnrThresh3` levels, to compare two contours, the algorithm first aligns the two contours. For each contour, the algorithm calculates the coordinates of the average of all the points that make up the polygonal contour. Then, the algorithm shifts them to line these 2 points up. The shift in the horizontal time axis is irrelevant; however, the shift in the frequency axis, referred to as `Shifty`, is important in determinining the species. The more the `Shifty`, the less likely the two contours are emitted by the same species.

Then, the algorithm calculates the Boolean union and intersection of that pair of contours. The crux of this species identification algorithm is the `overlapRatio`, given by the area of the intersection divided by the area of the union. If the two contours are the same, then the ratio of the intersection to the union is one. Otherwise, the higher the `overlapRatio` is, the more alike the two contours are.

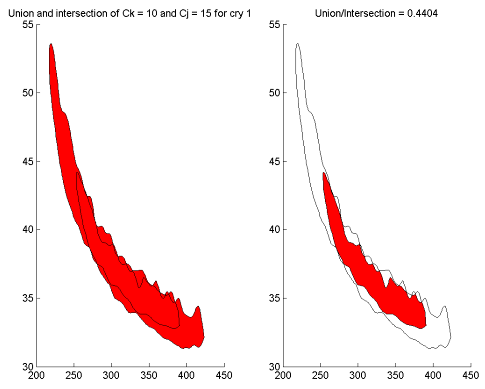


Figure 9 Boolean union and intersection of two contours at different thresholds. In this case, `overlapRatio` is 0.4404.

For each cry, the algorithm takes the maximum `overlapRatio` for each template. The algorithm then calculates the weighted sum of the `overlapRatio` and the inverse of `Shifty` for each template. This is accumulated for all the cries in the file, and is finally stored in a vector called `SPECIES`. The maximum in `SPECIES` determines the ultimate decision of the algorithm for the entire file; the algorithm outputs the top three candidates in `SPECIES`.

4.2 Identification Based on Minimum Frequency

To avoid having to match every single template with each unknown cry, which is prohibitively computationally expensive, the algorithm first finds the templates that are most possible based on `cryMinFreq`. A human specifies several ranges of possible minimum frequencies, and which species can be found in these frequency ranges. This is done by organizing "groups" of species using the cell array `GROUP`. For instance, `GROUP{1}`, the first group, might consist of species 1, 2, and 3, and `GROUP{2}`, the second group, might consist of species 1, 2, 3, 4, and 5. Note that each species may be part of more than one group. The frequency range for each group is determined in the array `SpeciesFreq`. For each group `iGroup`, the frequency range lies between `SpeciesFreq(iGroup)` and `SpeciesFreq(iGroup+1)`, inclusive. Let us consider an example in which `SpeciesFreq(1:3)` is equal to `[0, 30500, 31000]`. Taking our previous example for the groups, it would mean that species 1, 2, and 3 can range from 0 to 31000 Hz, and species 10 and 11 can only range from 30500 to 31000 Hz. So, if a cry is less than 30500 Hz, it is pointless to compare it with species 4 and 5, so comparing it only with species 1, 2, and 3 suffices. Conversely, if a cry lies between 30500 and 31000 Hz, it is just as possible for it to be species 4 and 5 as it is for it to be species 1, 2, or 3, so it must be compared with all five of them.

The more the number of groups, the more precise; the number of groups is determined by `numGroups`. For each unknown cry, the algorithm finds the frequency range in which it belongs, and only matches it against the group corresponding with the range. Since computation time increases linearly with the number of templates, avoiding comparison with unlikely templates can save a significant amount of time.

4.3 Multiple Templates

Some species have a large amount of biological variance within that species, and the cries emitted by different individuals in the species may be sufficiently different to put many individuals of that species at a disadvantage when being compared by the algorithm. Thus, for some species, multiple templates are used. Usually secondary or tertiary templates are sourced from files that were routinely misidentified as other species. Near the end of the algorithm, the results of all the templates for that particular species are summed prior to the calculation of the maximum in `SPECIES`. It is important to realize that *the algorithm treats each template as if they are separate species until the end*, when this step is performed. If needed, a multiplier may be used to maintain balance.

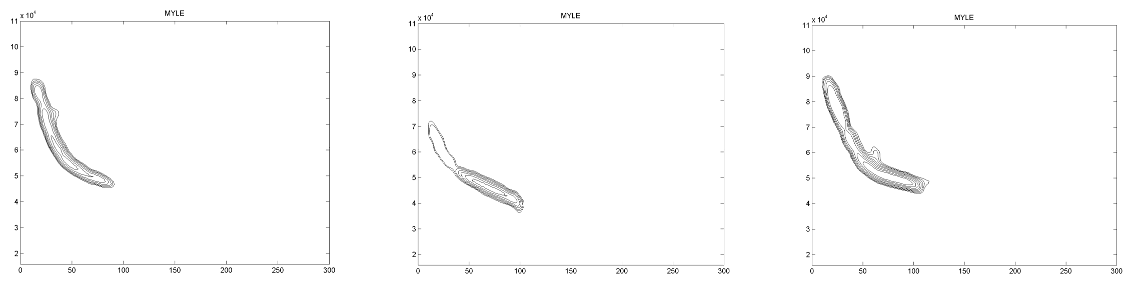


Figure 10 Three templates of *Myotis leibii*. See Appendix II.

4.4 Identification Based on Time between Cries

The algorithm calculates time between cries by finding the mean difference in the array `cryStart`. In distinguishing between *Lasiurus cinereus* and *Eptesicus fuscus*, which have a similar frequency range, the algorithm takes into account the observation that *E. fuscus* generally has a shorter time between cries, and modifies the array `SPECIES` accordingly.

Whilst a short time between cries means that the emitter was more likely to be *E. fuscus*, no conclusion can be drawn if the time between cries is long. This is because the algorithm does not distinguish between *E. fuscus* and *Lasionycteris noctivagans* (it uses the same templates for them), and *L. noctivagans* may have a significantly longer time between cries. So *E. fuscus* and *L. noctivagans* combined have a great variance in time between cries (see Appendix III). However, it is possible that in the future the algorithm may be refined so as to differentiate between *E. fuscus* and *L. noctivagans*.

The algorithm also outputs the time between cries per file so that human users may draw their own conclusions.

4.5 IDspecies

The function `IDspecies` performs everything described in this section. It outputs the decision on which species is most likely in the array `strongest`, where `strongest(1)` corresponds to the most likely species, and `strongest(2)` corresponds to the second most likely, and so on. Moreover, it outputs a number `correctness` which gives some estimate of how correct the decision is. This is calculated from the difference between the `SPECIES` of the first and second best matches, the total overlapRatio (if the total overlapRatio for all templates is very low, it means that none of the templates matched well), and the amount of `Shifty`. However, correlation between `correctness` and the actual correctness of the decision is quite minimal. See Appendix I and Appendix III.

5. Results

When the optimized detection algorithm was tested on twenty files containing verified cries from *Myotis leibii*, *Myotis lucifugus*, *Lasiurus cinereus*, and *Eptesicus fuscus*, the detector managed to accurately pinpoint the start and stop times of 93.6% of the cries (weighted by strength), and with a precision of 0.85 milliseconds. Out of about 417 cries in all the files, the detector only returned one false positive. This was performed by using a Microsoft Excel file to store the start and stop times of cries found visually by the human. For each cry, the human sets an arbitrary weight based on the perceived strength of the cry – a strong cry would have a weight close to 1 and a weak one would have a weight close to 0. Each time the algorithm finds a cry that is in the Excel file, it adds the weight to the current score. For instance, if there are three cries with weights 0.9, 0.7, and 0.2, and the algorithm detects the first two but misses the third, then the score would be $(0.9 + 0.7)/(0.9 + 0.7 + 0.2) = 89\%$. As such, the algorithm is not penalized greatly for missing very weak cries, though it is still better not to miss them. False positives, in which the algorithm finds a cry that the human did not, are counted separately. Usually, the human is considered to be more correct, so false positives are minimized. In this case, the only false positive appears to be an artefact of noise. The function `checkbat` was developed specifically to compare `cryStart` and `cryStop` times with the Microsoft Excel file.

The species identification algorithm was tested on 41 files containing cries from already identified species, including *Myotis leibii*, *Myotis lucifugus*, *Myotis septentrionalis*, *Lasiurus cinereus*, *Lasiurus borealis*, *Eptesicus fuscus* or *Lasionycteris noctivagans*, and *Perimyotis subflavus*, including the 20 files used to test cry detection. It correctly identified 38 out of 41 cries. Two of the misidentifications were when the algorithm mistook *Myotis lucifugus* for *Myotis leibii*, which are very similar-sounding. See Appendix III.

The algorithm for multilateration has not been sufficiently tested to tell whether it is working as intended.

6. Areas for Further Development

The eventual goal is to process large multichannel data. At present, the script developed so far is capable of loading very large (~2.6 GB) raw multichannel audio data, detecting cries, and outputting some coordinates for the matching cries. It detects the cries on the four channels independently, and computing memory is conserved by processing the file in smaller buffers.

6.1 'Bursts' of Cries

A *burst* of bat cries is defined as a series of regular echolocation cries emitted by one individual bat. The length of a burst may contain any number of cries, as long as the cries are more or less regular (i.e. there are no excessively large gaps between any two cries) and are deemed most likely to come from one individual. The precise definition of a burst is arbitrary.

The concept of a burst is needed when processing very large files that contain cries from many individuals at many different points in time. Firstly, it is very difficult to discern the species of a bat based on only one cry, so `IDspecies` is designed to work with bursts of cries. Secondly, the time between cries is of great interest to bat biologists, and accurately identifying bursts will greatly improve the precision of the estimate of time between cries. Thirdly, an algorithm can be used to interpolate the flight path of the bat using splines based on the coordinates generated by multilateration. Such an interpolation would hopefully mitigate inaccuracies in a fashion similar to the concept of smoothing the data. It would also make it easier to visualize the flight paths. In addition, it may be possible to extrapolate in case a few cries were too weak to be detected.

The short wave files that `IDspecies` has been tested on so far only contain single bursts; but when working with very large files, the algorithm must be capable of automatically detecting such bursts. A way to do this would be to use a function similar to `mergeCries`, by grouping cries into bursts only when the gap between cries is less than a certain threshold. Ideally, however, the algorithm should be capable of discerning such bursts even when there are multiple bats. For that, more advanced algorithms should be developed in the future.

6.2 Identification Based on Time between Cries and slantRatio

It was mentioned in Section 4.4 that the time between cries can be highly useful in identifying bat species. In the future it would be best to differentiate between *Eptesicus fuscus* and *Lasionycteris noctivagans* based on time between cries.

In addition, the variable `slantRatio` outputted by findFMsweeps may be useful to some extent in identifying bat species. It may be possible to further reduce the number of templates needed to compare with each cry by analyzing its `slantRatio`. For instance, consider the cries of *Myotis septentrionalis* and the cries of *Perimyotis subflavus*, which have similar `cryMinFreq`. It is clear that the former is much more steeply sloped than the latter.

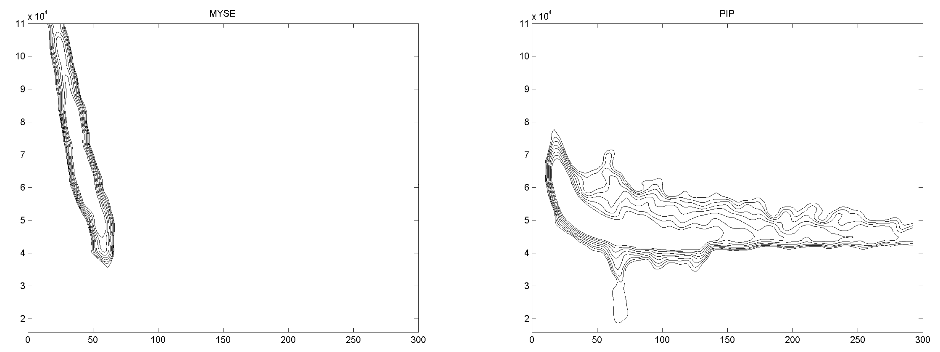


Figure 11 Contours of *Myotis septentrionalis* and *Perimyotis subflavus* cries, respectively.

7. Conclusion

The algorithms for detecting bats and identifying bat species are capable of functioning with acceptable precision and are useful in the future.

8. References

Baerwald, E. F., D'Amours, G. H., Klug, B. J., & Barclay, R. M. (2008, August 26). Barotrauma is a Significant Cause of Bat Fatalities at Wind Turbines. *Current Biology, 18*(16), pp. R695-R696.

Bucher, R., & Misra, D. (2002, August 1). A Synthesizable VHDL Model of the Exact Solution for Three-dimensional Hyperbolic Positioning System. *VLSI Design, 15*(2), pp. 507-520.

Appendix I: Function Declarations

These are the declarations of some important functions in their latest form, organized alphabetically. Function name is bolded. Note version number.

applyFMsweeps

Second pass of strong cry detection. Uses `findFMsweeps2`.

```
function [numCries2, cryStart2, cryStop2, cryMinFreq2, cryMaxFreq2, slantRatio] = ...
    applyFMsweeps2(...
        data,...%The time series data
        cryStart,...%in samples, output of findCries3
        cryStop,...%in samples, output of findCries3
        noise,...%1-dimensional array, size is height of spectrogram, in dB
        minFreq,...%Minimum freuqency for cries, in Hz
        smoothSize,...%smoothSize2, size to smooth spectrogram
        FFTlen,...%The length to perform fast fourier transform on
        sr,...$The sampling rate of data
        noverlap,...%This is calculated in detectCries
        SnrThresh2, ...%Threshold for second pass, in dB
        circumThresh,...%Minimum circumference, in second pass
        compactThresh,...%Minimum compactness, in second pass
        thetaBoundary,...%Boundary of theta in second pass
        ratioThresh,...%Threshold for slantRatio, in second pass
        segLen...%Segment across which to calculate theta in second pass
        )
```

detectCries

Detects cries. Uses `findCries3`, `applyFMsweeps2`, and `mergeCries`.

```
function [ cryStart3, cryStop3, numCries3, cryMinFreq, cryMaxFreq,...
    slantRatio, medianSNR, noise] = detectCries( ...
        data, ...%The time series data
        sr, ...%Samping rate, in Hz
        FFTlen,...%The length to perform fast fourier transforms on
        windowShift,...%The amount of shift of the window
        SnrThresh1,...%Threshold for first pass, in dB
        minFreq,...%Minimum frequency for cries, in Hz
        minDuration,...%Minimum duration of cry in first pass, in seconds
        maxDuration,...%Maximum duration of cry in first pass, in seconds
        minGap,...%Minimum gap between cries, in seconds
        smoothSize1,...%Size to smooth spectrogram
        thetaBoundary ,...%Boundary of theta in second pass
        ratioThresh,...%Threshold for slantRatio, in second pass
        circumThresh,...%Minimum circumference, in second pass
        compactThresh,...%Minimum compactness, in second pass
        SnrThresh2,...%Threshold for second pass, in dB
        segLen,...%Segment across which to calculate theta, in second pass
        smoothSize2,...%Size to smooth spectrogram, mustbe array of 2 positive numbers
        WSnrThresh1,...%SnrThresh1 for weak cries, in dB
        WminFreq,...%minFreq for weak cries, in Hz
        WmaxDuration,...%maxDuration for weak cries, in seconds
        WminDuration,...%minDuration for weak cries, in seconds
        WminGap,...%minGap for weak cries, in seconds
        mergeThresh,...%Threshold to merge, in seconds
        mergeThresh2...%Threshold to not merge, in seconds
        )
```

distMatrix

Generates a 3D distance matrix for multilateration.

```
function [ Dt, X, Y, Z, numX, numY, numZ ] ...
    = distMatrix(...
        maxX, maxY, maxZ,...%Physical size of matrix, in meters from center
        dx, dy, dz,...%Resolution of matrix, in meters
        mic1x, mic1y,...
        mic2x, mic2y,...
        mic3x, mic3y,...%Positions of microphones (other than mic0), in meters
        c... %Speed of sound, in meters per second
        )
```

findCries

First pass of cry detection. Uses `myHanning` and `smooth2D`.

```
function [cryMinFreq, cryMaxFreq, numCries, cryStart, cryStop, medianSnr, noise] = ...
    findCries3(...
        data,... %The time series data
        sr,...   %Sampling rate, in Hz
        FFTlen,... %The length to perform fast fourier transform on
        minFreq,...  %Minimum frequency that we expect a cry to be, in Hz
        SNRthresh,... %Threshold, in dB
        minDuration,... %Minimum duration of a cry, in seconds
        maxDuration,... %Maximum duration of a cry, in seconds
        minGap,... %Minimum gap between cries, in seconds
        smoothingSize...  %Size of hanning window to smooth spectrum
        )
```

findFMSweeps

Second pass of strong cry detection. Uses `analyzeContour2`, `topoContour2`, and `unpackContour4`.

```
function [ContourxOut, ContouryOut, numCOut, NumPtOut, CircumfOut, SlantRatioOut,
CompactnessOut] = ...
    findFMsweeps3(...
        Snr,...%Spectrogram calculated by applyFMsweeps
        SnrThresh2, ...%Threshold for second pass, in dB
        circumThresh,...%Minimum circumference, in second pass
        compactThresh,...%Minimum compactness, in second pass
        thetaBoundary,...%Boundary of theta in second pass
        defaultRatio,...%Initialized in applyFMsweeps, hardcoded
        ratioThresh,...%Threshold for slantRatio, in second pass
        segLen...%Segment across which to calculate theta in second pass
        )
```

IDspecies

Identifies bat species. Uses `detectCries`. Note that `templatespath` and `detectCriesStuff` must be valid paths for the function to work.

```
function [SPECIES,correctness,tBtwnCries,CryMinFreq,strongest] = ...
    IDspecies(...
        data,...%time series data of a burst of cries
        sr_file,...%sampling rate of file, in Hz
        sr_template,...%sampling rate of template, in Hz
        SnrThresh3,...%Thresholds to calculate contours, in dB
        padding,...%in seconds
        CryMinFreqTrim,...%percent, between 0 and 100.
        smoothSize3,...%Smoothing size
        yshiftrange,... %integer, in multiples of freqSpacing of sr_template
        )
```

The contents of `templatespath` and `detectCriesStuff` are as follows (please note that exact paths must be changed for this to work on different computers):

```
    load 'F:\Documents\2010W co-op\m files\templates' templatex templatey templates
        numSpecies Shiftweight SpeciesFreq GROUP numGroups;
    load 'F:\Documents\2010W co-op\m files\detectCriesStuff' FFTlen windowShift
        SnrThresh1 minFreq minDuration ...
        maxDuration minGap smoothSize1 thetaBoundary ratioThresh circumThresh ...
        compactThresh SnrThresh2 segLen smoothSize2 WSnrThresh1 WminFreq ...
        WmaxDuration WminDuration WminGap mergeThresh mergeThresh2;
```

mergeCries

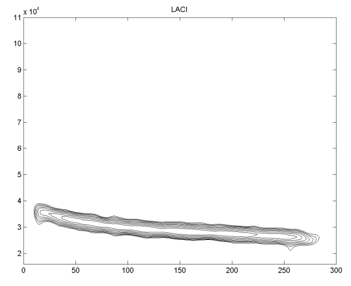
Merges bat cries that are too close together, and ignores possible echos.

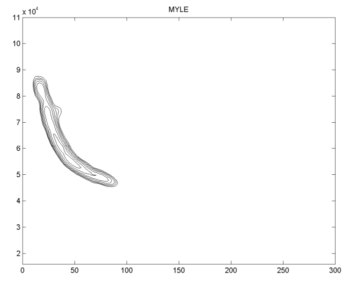
```
function [cryStart3, cryStop3, numCries3, cryMinFreq, cryMaxFreq] = ...
    mergeCries(...
        cryStart2,...%in samples
        cryStop2,...%in samples
        numCries2,...%must be integer
        mergeThresh,...%in samples. Converted from seconds to samples in detectCries.
        mergeThresh2,...%in samples. See above.
        cryMinFreq2,...%in Hz
        cryMaxFreq2...%in Hz
        )
```

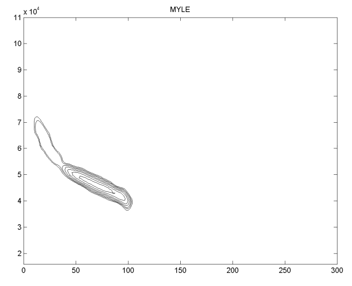
Appendix II: Templates

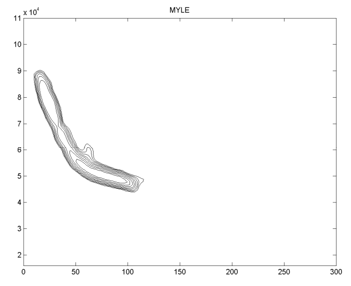
Below are the contours of all the templates being used as of writing. The number of templates, including multiple ones for the same species, is known as `numSpecies`. Recall that the algorithm treats each template as if they are all different species until the end, when it sums up the results for each actual species.

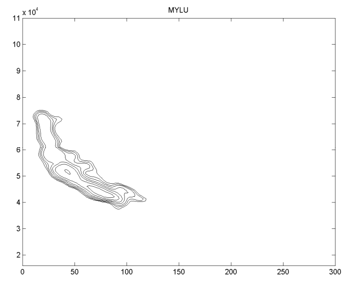
The templates are polygons, with vertices stored in the cell arrays `templatex` and `templatey`. All polygons are clockwise-oriented. For each `SnrThresh3` threshold `iSnr` for each template species `iSpecies`, there is a polygon that is defined as `templatex{iSpecies,iSnr}`, `templatey{iSpecies,iSnr}`. The nature of cell arrays allows `templatex{iSpecies,iSnr}` to vary in size with respect to `iSpecies` and `iSnr`, though it should always match `templatey{iSpecies,iSnr}`. There is a separate cell array the size of `numSpecies` called `templates` that stores the names of the species as strings.

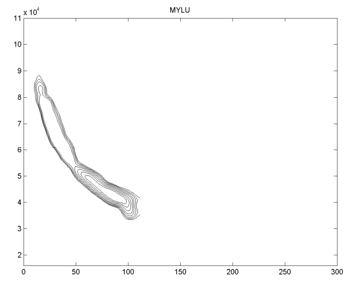


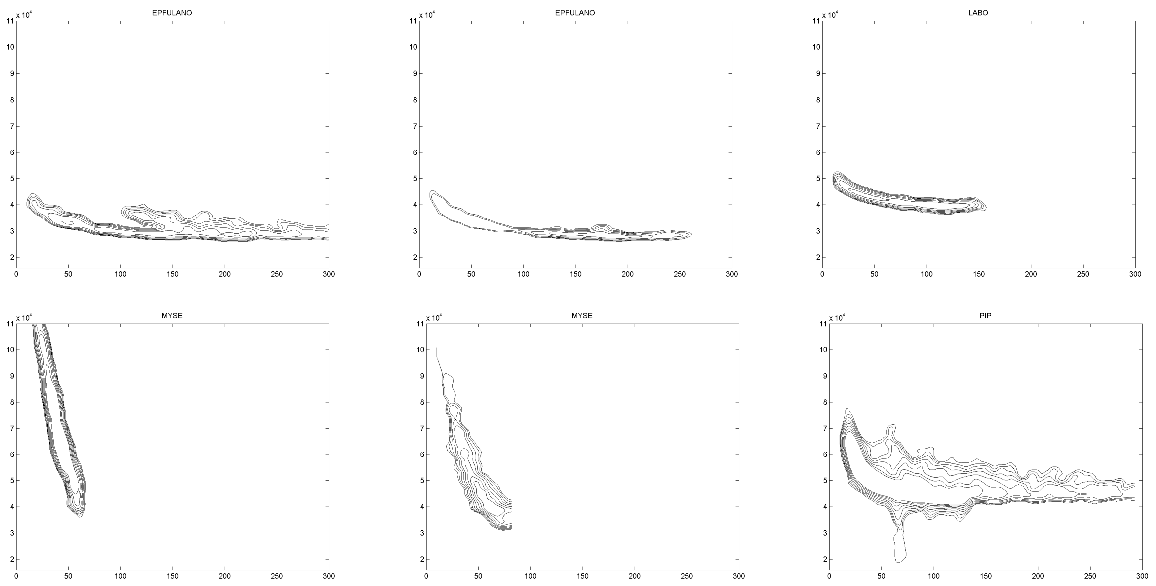











EPFULANO
EPFULANO
LABO
MYSE
MYSE
PIP
x 10^4

Appendix III: Results of Species Identification

The following shows the output of the function `IDspecies` for 47 files, of which 41 are known species. From left to right: file name; first most likely species; second most likely species; third most likely species; an estimate of correctness (the higher the better, in general); trimmed mean time between cries (in seconds); mean `cryMinFreq`. Incorrect identifications are set in red font, and unknown species are set in blue font. Note that here *Eptesicus fuscus* and *Lasionycteris noctivagans* are treated as one species `EPFULANO`. Eventually in the future it would be best to differentiate them based on `tBtwnC` (see section 4.4).

```
01. Talbot...rus cinereus ; 65%LACI; 35%EPFULANO; 0%MYLE; c=109.5764; tBtwnC=0.24624; CMinF=24KHz
02. Wolfe_...otis lebeii) ; 79%MYLE; 21%PIP; 0%LACI; c=107.334; tBtwnC=0.0617; CMinF=45KHz
03. Wolfe_...otis lebeii) ; 83%MYLE; 17%PIP; 0%LACI; c=82.8702; tBtwnC=0.080652; CMinF=46KHz
04. Wolfe_...otis lebeii) ; 81%MYLE; 19%PIP; 0%LACI; c=76.9703; tBtwnC=0.087148; CMinF=48KHz
05. Wolfe_...otis lebeii) ; 84%MYLE; 16%PIP; 0%LACI; c=128.3396; tBtwnC=0.090228; CMinF=46KHz
06. Wolfe_...s lucifugus) ; 29%MYLU; 25%MYLE; 19%MYSE; c=6.268; tBtwnC=0.082952; CMinF=38KHz
07. Wolfe_...otis lebeii) ; 80%MYLE; 20%PIP; 0%LACI; c=113.6419; tBtwnC=0.056168; CMinF=48KHz
08. Wolfe_...otis lebeii) ; 81%MYLE; 19%PIP; 0%LACI; c=87.3484; tBtwnC=0.08638; CMinF=45KHz
09. Wolfe_...otis lebeii) ; 30%MYLE; 25%MYLU; 21%PIP; c=4.7599; tBtwnC=0.091632; CMinF=35KHz
10. Wolfe_...otis lebeii) ; 26%MYLE; 25%PIP; 20%MYLU; c=3.4198; tBtwnC=0.10664; CMinF=38KHz
11. Wolfe_...otis lebeii) ; 27%MYLE; 26%PIP; 23%MYLU; c=0.70606; tBtwnC=0.10724; CMinF=36KHz
12. Wolfe_...us cinereus) ; 54%LACI; 46%EPFULANO; 0%MYLE; c=4.7815; tBtwnC=0.21832; CMinF=26KHz
13. Wolfe_...us cinereus) ; 52%LACI; 48%EPFULANO; 0%MYLE; c=6.0076; tBtwnC=0.21496; CMinF=26KHz
14. Wolfe_...us cinereus) ; 60%LACI; 40%EPFULANO; 0%MYLE; c=101.514; tBtwnC=0.3892; CMinF=24KHz
15. Wolfe_...icus fuscus) ; 61%EPFULANO; 39%LACI; 0%MYLE; c=11.0289; tBtwnC=0.067804; CMinF=29KHz
16. Wolfe_...us cinereus) ; 58%LACI; 42%EPFULANO; 0%MYLE; c=28.2111; tBtwnC=0.31736; CMinF=24KHz
17. Wolfe_...otis lebeii) ; 35%MYLE; 30%MYLU; 14%PIP; c=5.8813; tBtwnC=0.094748; CMinF=38KHz
18. Wolfe_...s lucifugus) ; 30%MYLU; 28%MYLE; 17%LABO; c=1.7222; tBtwnC=0.098568; CMinF=38KHz
19. Wolfe_...s lucifugus) ; 35%MYLU; 32%MYLE; 13%PIP; c=2.1506; tBtwnC=0.094272; CMinF=36KHz
20. Wolfe_...is lucifugus ; 35%MYLU; 32%MYLE; 12%MYSE; c=4.5652; tBtwnC=0.093824; CMinF=36KHz
21. Wolfe_...eeding buzz) ; 37%MYLE; 32%MYLU; 12%PIP; c=12.8683; tBtwnC=0.086592; CMinF=37KHz
22. Wolfe_...eeding buzz) ; 86%MYLE; 14%PIP; 0%LACI; c=87.6987; tBtwnC=0.093712; CMinF=47KHz
23. Myotis...search phase ; 64%MYSE; 18%MYLE; 7%LABO; c=49.9109; tBtwnC=0.077248; CMinF=35KHz
24. Pipist...search phase ; 37%PIP; 27%LABO; 18%MYLE; c=1.7171; tBtwnC=0.11568; CMinF=40KHz
25. L_bore...e-E4_09aug08 ; 29%LABO; 21%MYLU; 19%MYLE; c=9.9169; tBtwnC=0.13545; CMinF=37KHz
26. Lasiur...rus borealis ; 52%LABO; 15%MYLE; 12%MYLU; c=19.3817; tBtwnC=0.10257; CMinF=36KHz
27. Epfu_L...o-E3_04aug08 ; 92%EPFULANO; 51%LACI; 0%MYLE; c=112.6387; tBtwnC=0.10239; CMinF=27KHz
28. M_sept...e-E2_11aug08 ; 52%MYSE; 33%MYLU; 15%LABO; c=14.4439; tBtwnC=0.077345; CMinF=32KHz
ColtRd-T2_...s_67-70_ch_2 ; 33%MYLU; 29%MYLE; 15%PIP; c=6.6099; tBtwnC=0.09425; CMinF=35KHz
ColtRd-T2_...217-218_ch_2 ; 30%PIP; 30%LABO; 15%MYLU; c=0.5855; tBtwnC=0.21898; CMinF=37KHz
ColtRd-T2_...219-220_ch_1 ; 47%LABO; 15%MYLU; 15%MYLE; c=181.8004; tBtwnC=0.29079; CMinF=37KHz
ColtRd-T2_...807-812_ch_1 ; 25%MYLU; 25%PIP; 24%MYLE; c=0.2437; tBtwnC=0.090345; CMinF=37KHz
ColtRd-T2_...202-204_ch_3 ; 62%LABO; 20%MYLU; 18%MYSE; c=14.5202; tBtwnC=0.13445; CMinF=34KHz
ColtRd-T2_...203-205_ch_2 ; 65%LABO; 18%MYSE; 17%MYLU; c=32.288; tBtwnC=0.14946; CMinF=34KHz
Epfu_Lano_...e-E3_13aug08 ; 55%EPFULANO; 45%LACI; 0%MYLE; c=39.0301; tBtwnC=0.23849; CMinF=28KHz
Epfu_Lano_...o-E3_04aug08 ; 92%EPFULANO; 51%LACI; 0%MYLE; c=112.6387; tBtwnC=0.10239; CMinF=27KHz
Eptesicus ...search phase ; 62%EPFULANO; 38%LACI; 0%MYLE; c=3.6268; tBtwnC=0.077112; CMinF=30KHz
L_borealis...e-E4_09aug08 ; 27%LABO; 24%PIP; 20%MYLU; c=3.7714; tBtwnC=0.10738; CMinF=35KHz
L_borealis...e-E4_09aug08 ; 29%LABO; 21%MYLU; 19%MYLE; c=9.9169; tBtwnC=0.13545; CMinF=37KHz
L_cinereus...o-E4_29jul08 ; 52%LACI; 48%EPFULANO; 0%MYLE; c=1.2955; tBtwnC=0.20929; CMinF=24KHz
Lasiurus b...rus borealis ; 52%LABO; 15%MYLE; 12%MYLU; c=19.3817; tBtwnC=0.10257; CMinF=36KHz
Lasiurus c...search phase ; 53%LACI; 47%EPFULANO; 0%MYLE; c=0.30117; tBtwnC=0.21595; CMinF=19KHz
M_lucifugu...e-E2_11aug08 ; 33%MYLE; 28%MYLU; 18%PIP; c=8.8149; tBtwnC=0.10719; CMinF=38KHz
M_lucifugu...o-E2_05aug08 ; 27%MYLE; 26%MYLU; 25%PIP; c=0.33832; tBtwnC=0.10823; CMinF=40KHz
M_sep_2bat...12aug08_2008 ; 40%MYLE; 26%MYSE; 24%MYLU; c=22.0413; tBtwnC=0.0817; CMinF=41KHz
M_sep_Muri...o-E3_20aug08 ; 44%MYSE; 28%MYLE; 15%MYLU; c=19.2559; tBtwnC=0.083565; CMinF=40KHz
M_septentr...e-E2_11aug08 ; 52%MYSE; 33%MYLU; 15%LABO; c=14.4439; tBtwnC=0.077345; CMinF=32KHz
```

The code for displaying the data in the syntax shown above is as such:

```
disp([filename(1:10),'...',filename(end-15:end-4),' ; ',...
    num2str(round(SPECIES(strongest(1))*100)),'%', templates{strongest(1)},'; ',...
    num2str(round(SPECIES(strongest(2))*100)),'%', templates{strongest(2)},'; ',...
    num2str(round(SPECIES(strongest(3))*100)),'%', templates{strongest(3)},'; c=',...
    num2str(correctness), '; tBtwnC=', num2str(tBtwnCries),...
    '; CMinF=', num2str(round(CryMinFreq/1000)),'KHz'])
```