Supplementary Information: High duty-cycle bats in the field do not alter echolocation calls when flying in groups

Neetash Mysuru, Thejasvi Beleyur, Aditya Krishna, Holger R Goerlitz; : joint first authors

Last Updated : 2024-11-19 09:48:29

# 1 Checking matched audio for each annotated bat flight activity

For each bat flight activity period identified in the video, we identified the corresponding region of the recorded audio. We quantified the median intensity of the pixels in the region around the LED for each frame and then cross-correlated the normalized pixel intensity with the recorded ON/OFF voltage signal in the audio files. We managed to successfully find audio matches for 1181 bat flight activity periods (55% of 2132). The low match rate is primarily due to the fluctuating camera frame rates, and because many of the matched audio files originated from non-target bat species, which could not be visually discriminated from our target species *R. mehelyi/euryale* while annotating bat activity periods in the videos. Observed non-target species were *R. ferrumequinum*, vespertilionid and miniopterid FM bats, all of which occur in the Orlova Chuka cave system (1). For the acoustic analysis we chose matched audio files that contained only calls of *R. euryale* and/or *R. meheyli*.

# 2 Audio-video synchronisation: hardware and software implementations

The audio and video data were synchronised using the protocol of (2). A Raspberry Pi 3 was used to drive an ON/OFF signal from a GPIO port. This ON/OFF signal was then split between an LED and a circuit linked to capacitor. The capacitor converted the DC ON/OFF signal into positive and negative spikes - thus allowing the signal to be correctly digitised. Not all soundcards are capable of digitising DC voltages, and thus the capacitor helps in making the protocol independent of soundcard type. The entire circuit can be assembled from easily available parts (Figure 2.1)

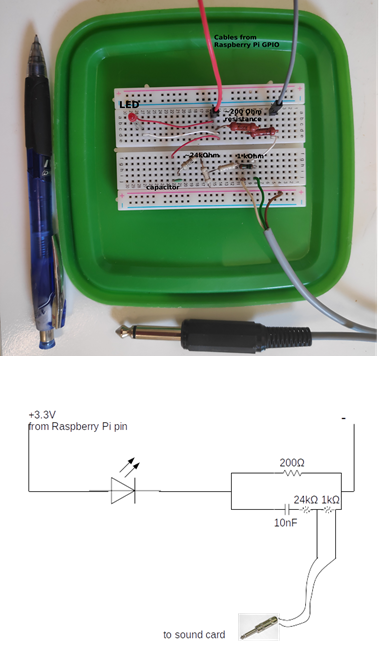


Figure 2.1: Top: The experimental realisation of the audio-video synchronisation signal splitting. The components can easily be assembled onto a hobby breadboard, and are easily portable. Here the breadboard is pasted on the inside of a lunch box lid, allowing easy and safe transport of the breadboard and the Raspberry Pi in the box itself. Bottom: The circuit diagram of the synchronisation signal splitter

The code to drive the GPIO port runs on Python 2 (and should also run on Python 3). For best results the python file can be set to automatically run on boot-up. This makes the synchronisation protocol field-friendly, and reduces the need of the experimenter manually running the code.

#!/usr/bin/python  
'''  
script that switches a RED LED on and off  
This script and the circuit used to  
run the system is based on the post at thePiHut  
'Turning on an LED with your Raspberry Pi's GPIO Pins'   
URL: https://thepihut.com/blogs/raspberry-pi-tutorials/  
27968772-turning-on-an-led-with-your-raspberry-pis-gpio-pins  
Accessed June 11 2015  
'''  
import RPi.GPIO as GPIO  
import sys  
import time  
GPIO.setmode(GPIO.BCM)  
GPIO.setwarnings(False)  
GPIO.setup(18,GPIO.OUT)  
import numpy as np  
  
time\_ranges = np.arange(0.08,0.5,0.0001)  
  
while True:  
 try:  
 #print ('LED ON')  
 GPIO.output(18,True)  
 on\_time = np.random.choice(time\_ranges,1)  
 time.sleep(on\_time)  
 #print('LED OFF')  
 off\_time = np.random.choice(time\_ranges,1)  
 GPIO.output(18,False)  
 time.sleep(off\_time)  
 except KeyboardInterrupt:  
 GPIO.output(18,GPIO.LOW)  
 sys.exit()

One optional change that can be made to the code above is to set the seed manually with np.random.seed after the numpy import. Setting a fixed seed can have the advantage that problems in audio-video file synchronisation post data collection can be better fixed. A fixed seed however means that the output signal is the same across all sessions used - which might make distinguishing audio and video recordings from different sessions difficult.

Another important aspect to pay attention to is the time\_ranges variable. In this experiment it was assumed that the camera frame rate was going to be 25 Hz, and thus the lowest ON/OFF time was set to 0.08s, which corresponds to a signal with 12.5Hz periodicity of the Nyquist frequency. However, as (2) suggest, it would have been better to set the lowest duration to a longer period, which was a few times lower than the Nyquist frequency of 12.5 Hz, eg. 0.2s (5 Hz). In our experiments, the cameras turned out to have a frame rate of 22Hz, which meant that the LED signal was aliased. However, despite the aliasing, we were still able to synchronise audio and video - showing the robustness of the methodology. The key to success in our case was probably that the ON/OFF signal had a large bandwidth (5-12.5 Hz), and thus much of the signal was not altered due to aliasing - allowing the cross-correlation to accurately match video and audio segments.

# 3 Video annotation of bat flight activity in the cave

Manual annotation of the video data was carried out to determine the group sizes of free-flying horseshoe bats in their natural habitat. We annotated bat flight activity by simultaneously viewing the video feeds from both infrared cameras using SHOTCUT (3 v 19.04.30), an open-source video editing software. The following information was documented from the video: the start and end times of bat flight activity from the burnt in timestamps from either camera 1 or 2 in “yyyy-mm-dd hh:mm:ss” format, frame number, number of bats flying and flight behavior. A bat flight activity is defined as the interval during which the number of bats flying inside the cave is constant. Successive bat flight activities were operationally defined as being separated from one another by least 6 frames.

We defined the start of bat activity from the frame a bat is observed to fly in either camera view. Similarly, the end of bat flight activity was when a bat is not observed in either of the camera views. Our video protocol originally included counts of the exact number of bats in a bat activity period. In parts of the video with rapid dynamic transitions in the number of bats, we only annotated ‘stable’ flight activities where the number of visible bats were maintained for at least 10 frames. When the number of visible bats fluctuated every few frames, such activity periods were not annotated. We prioritized obtaining a clean data set and refrained from annotating extremely difficult bat flight annotations because of how dynamic the group size shifts could be.

# 4 Individual call analysis

Individual calls were selected from the audio files based on a set of pre-defined search protocol:

* All measurements and signal processing will be done using Audacity.
* dB rms measurements made with the ‘Contrast’ function in Audacity. Highpassing done with the inbuilt highpass filter. The SNR is calculated by difference between the foreground (bat call region) and background (silent region)
  1. Load annotation audio file, and delete all non-target channels.
  2. View audio in spectrogram mode. Set dynamic range of spectrogram to 60dB.
  3. Highpass filter audio file with 12 dB roll off/octave at 80 kHz cutoff frequency
  4. For given audio file, choose a start point using a random number generator between 0-1.
  5. Go to that fraction of time corresponding to the length of the annotation audio file
  6. Choose another random number between 0-1. If it’s <=0.5 search towards left, else search towards right.
  7. Look for a horseshoebat call with no overlaps, no interference patterns in the CF or FM, that can be isolated well.
  8. While selecting horseshoe bat calls, zoom in max till 60 milliseconds of audio occupy the whole screen. Do not zoom in more or less while selecting.
  9. Check the SNR of the selected horseshoe bat call by using a ‘silent period’ of the audio file as background. If there is not suitably long ‘silent period’ to serve as background in this audio file, choose another random audio file and measure the background dB rms.
     1. If SNR >= 20 dB, this is a suitable call to measure. Note down the start and end time of this call in the audio file.
     2. If SNR < 20dB
        1. Go back to search start point calculated in 4), and begin searching in opposite direction.
        2. Look for first suitable call to measure using criteria in 7) onwards.
     3. If a suitable call is still NOT found:
        1. No measurement takes place in this audio annotation. Proceed to next audio annotation file.

Audacity version 2.3.3 was used during the manual call selection.

## 4.1 CF duration: multi-call extension

The individual call analysis in 4 may have had the issue of biasing towards short calls with shorter CF components, as they may be more likely to remain un-overlapped. To overcome this we extended the CF duration measurements by developing an automated method to detect CF components even in situations with overlapping calls. The same audio files used in 4 were used as input in the extended approach too.

The ‘multi-call extension’ CF duration measurements involved the following steps:

1. Load the audio file and choose the loudest 0.5 second segment of the audio. If the audio is shorter than 0.5 seconds, then load the entire audio.
2. Choose the ‘band of interest’, defined as the peak frequency 1.5 kHz. The band of interest typically holds the CF calls from the bat with the louder calls eg. the band of interest is 104.5-107.5 kHz for an audio clip with peak frequency at 106 kHz.
3. Make a spectrogram with 128 samples FFT window and 127 samples overlap. At 250 kHz sampling rate, this corresponds to 1.95 kHz spectral resolution and 0.5 ms temporal resolution.
4. Check the spectrogram for all ‘pixels’ that are above a predefined threshold within the band of interest. For our data, -140 dB was set as the threshold.
5. Regions of connected pixels that are above threshold and greater than 10 ms are labelled as candidate CF detections, and and image of the spectrogram with candidates is saved.
6. The candidate CF detections for each audio were manually verified and only reliable detections are kept for further statistical analysis.

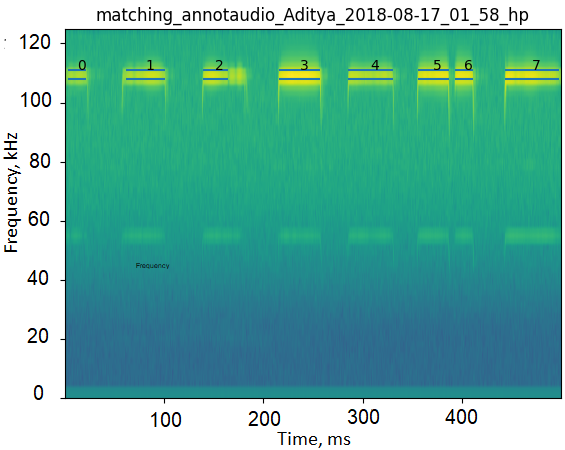


Figure 4.1: Example of CF candidate detections (blue parallel lines). In this case, detection numbers are retained, while the others are discarded () as the complete CF component is not detected.

The actual functions used to detect and measure CF durations are in the cf\_detector.py module, and the Jupyter notebook used to generate measurements for this paper is called cf-detection-notebook.ipynb.

## 4.2 *itsfm* parameters

An FM component was defined as any region of a call with greater or equal to 2 kHz/ms frequency modulation. Further settings were a maximum acceleration of 3 kHz/ms, an extrapolation window of 50 s and a window size of 375 samples. Threshold signal level and time-frequency clip range were manually tuned for each calls using the pwvd method, since each call had differing received levels.

# 5 Windowed call analysis

Each flight activity audio was split into consecutive 50ms windows. All tail-end audio that was <50 ms was discarded.

## 5.1 Choosing the ‘silent window threshold’

A series of manually annotated audio clips were used to set the reference silent window threshold. The manually annotated audio clips were the same as those used to calculate the reference ‘silence’ segments in the individual call analysis (, min-max duration=0.002-0.03s). The threshold for a window to be chosen as silent was set at 20dB above the maximum measured dB rms of all silent windows. This resulted in any window that was less than -23 dB rms as being considered ‘silent’. This is a conservative approach that prevents windows with poor signal-to-noise ratio from being analysed.

The code to execute this analysis is available in the what qualifies as a silent audio segment.ipynb notebook and its HTML printout.

## 5.2 Dominant frequency measurement

Unlike typical measures used to quantify echolocation calls like peak frequency or-10 dB frequency, the dominant frequencies provide a glimpse of what may be happening in the presence of multiple calls.

The dominant frequency was determined with the following steps:

1. Create a smoothed power spectrum. A smoothed power spectrum is generated by passing the raw spectrum (FFT size = 12500 samples) with a running-mean filter of the pre-defined spectral smoothing width. The spectral smoothing width defines the ‘width’ or the number of frequency bins of the running-mean filter. We used a smoothing width of 100 Hz, which corresponds to 5 frequency bins. The smoothing is necessary as the raw power spectrum can be very ‘jagged’ otherwise, and impede peak detection which corresponds to the CF components of calls in the input audio.
2. Extract the peaks in the smoothed power spectrum. Only peaks that are a minimum ‘distance’ from each other, and that are within a threshold of the highest peak are chosen. We chose an inter-peak distance of 250 Hz, and all valid dominant frequency peaks needed to lie within 14 dB of the peak with the highest power.
3. Map the valid peaks to the frequencies they correspond to. These are the dominant frequencies in this

The code to execute this function is available in the inbuilt\_measurement\_functions.py module.

## 5.3 FM lower frequency measurement

The FM lower frequency (Figure 5.1) is determined in the following steps:

1. Make spectrogram of the audio window (512 samples FFT, 256 samples overlap).
2. Identify all spectrogram ‘pixels’ in the FM frequency band that are above the baseline level. The FM bandwidth was defined as ranging from 70 kHz to 98 kHz. The baseline power level across pixels was calculated by calculating the 95%ile value of power in the frequency band below 70 kHz, i.e., the part of the spectrogram without any bat calls. All pixels whose power was 46 dB above the baseline power level and whose frequency was within 70 - 98 kHz were considered valid FM pixels
3. Identify contiguous clusters of FM pixels. These clusters represent single iFM or tFM components of calls.
4. From the identified continuous clusters, extract the lowest frequency pixel in a cluster

Given the current parameter values used for our analysis, the lower frequency measurements have a spectral resolution of 488 Hz. For comparison (4) had a spectral resolution of 292 Hz (300 kHz sampling rate, 1024 FFT window) in the tFM and iFM minimum frequency measurements. While the method above has a comparitively lower spectral resolution, it should be sufficient to pick up the relative large effects of ~ 5 kHz shifts reported in tFM minimum frequency by (4).

The code to execute this function is available in the inbuilt\_measurement\_functions.py module.

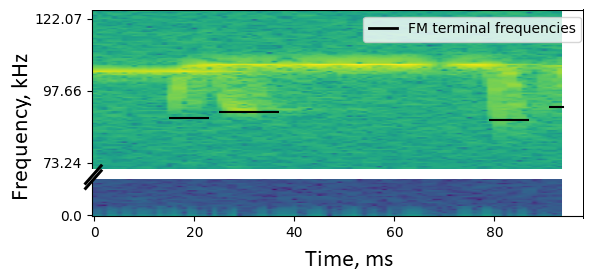


Figure 5.1: Example showing extracted FM lower frequencies from the spectrogram of a 50ms window. The method allows extraction of lower frequencies in the presence of multiple overlapping calls, though it doesn’t allow discrimination of iFM and tFM components. Note the truncated y-axis.

## 5.4 Making virtual multi bat audio files

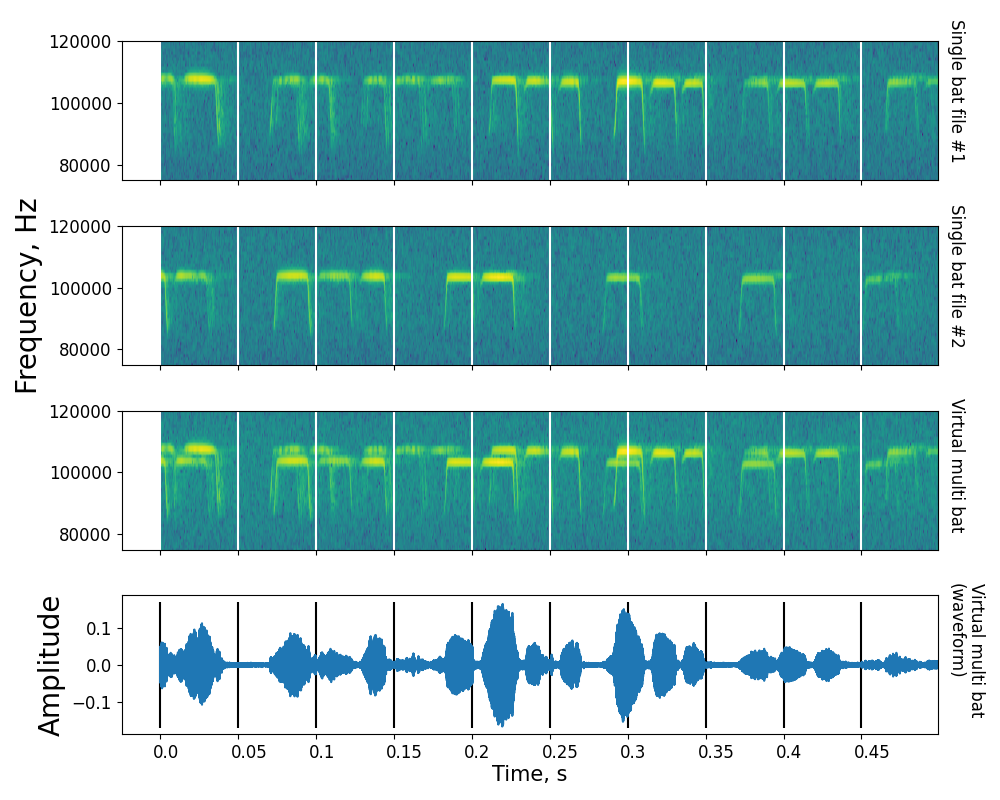


Figure 5.2: Example showing the steps involved in creating a virtual multi bat file. Shown here are spectrograms of the first 500ms of two single bat audio files (A), (B), along with the resulting virtual multi bat audio file. Vertical lines delineate 50ms windows that are used for acoustic measurements

Virtual multi bat audio files (Figure 5.2) were created with the following steps:

1. For each multi bat file generate a virtual multi bat audio file
   1. Among the pool of single bat audio file choose all files that are within 0.9-1.1 times the length of the current multi bat file.
   2. From the pool of duration matched single bat audio files, randomly select 2 or 3 files - depending on how many bats were observed in the current multi bat file
   3. Add the chosen single bat audio files together. Set the final virtual multi bat length to the length of the shortest single bat audio file.
   4. Remove the chosen single bat audio files from the pool of single bat audio files. The single bat audio files will not be used again to generate a virtual multi bat file.

The code to execute this function is available in the Making virtual multi bat audio.ipynb notebook and its HTML printout.

# 6 Supporting Results

## 6.1 Calculating expected dominant frequency ranges due to Doppler shift

The amount of Doppler shift in our audio recordings is primarily affected by multiple factors: 1) the flight speed of the bat 2) the flight direction of the bat with respect to the microphone 3) active Doppler shift compensation carried out by the bats and 4) differences the acoustic fovea of each individual bat. These factors may combine to give rise to a dominant frequency (DF) max-min range of up to around 3 kHz even when a single bat flies by the microphone. For example, a bat echolocating with a very high individual call frequency that flies fast will result in a larger DF range than a slow flying bat with the same foveal frequency but flying slower.

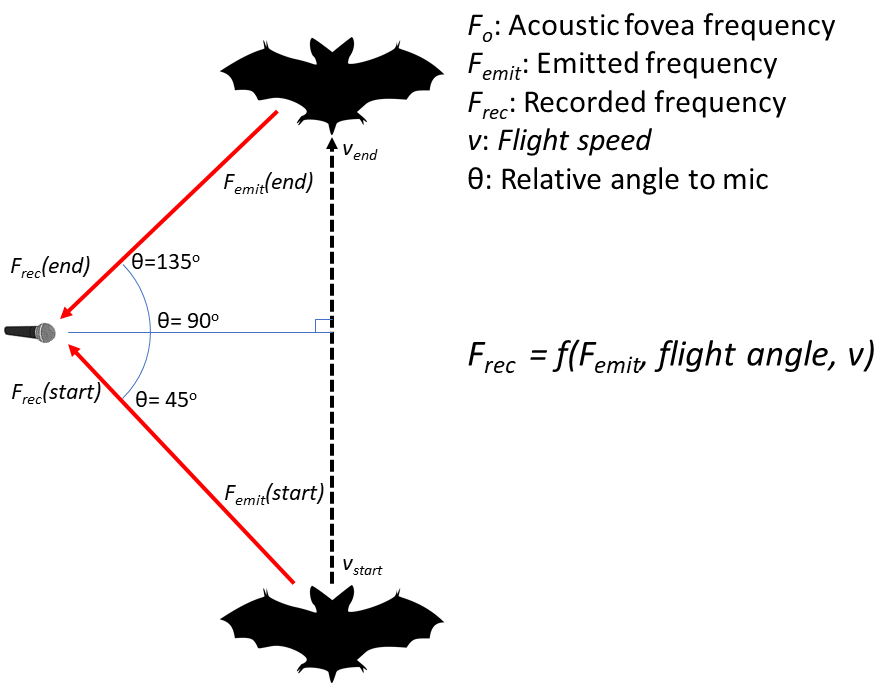


Figure 6.1: Schematic showing the simple model used to calculate the expected dominant frequency variation arising from a single bat flying past the microphone. is the doppler compensated emitted frequency. is the received frequency at the start of the flight, the received frequency at the end of the flight. and are the speed of the bat at the start and end of the flight. is a function of the emitted frequency, relative flight angle and flight speed at the start and end of the fly by.

Our simulations recreated the frequency recorded at the micrphone at the ‘start’ and ‘end’ of the bat’s flight past the microphone (Figure 6.1). The start position was assumed to be 45 degrees and end position was 135 degrees relative to the microphone (where 90 deg. corresponds to the bat flying exactly perpendicular to the microphone’s direction). The speed at the start and end flight positions of the bat was assumed to be between 1.5-4.5 m/s, and the acoustic fovea’s of the bat population was assumed to be between 100-111 kHz, matching the range of the study species’ *R. euryale/mehelyi*. The frequency recorded at the microphone due to Doppler shift from the bat flying at an angle was calculated by: . The bat’s Doppler shift compensation was modelled by assuming the bat perfectly compensated for Doppler shift due to it’s own flight speed. The was calculated at the start and end points as , where depended on the flight speed at the start and end points, m/s, and was a randomly chosen value between 100-111 kHz.The DF range was calculated as . Figure 6.2 shows that our the DF ranges from simulations match the observed DF ranges well for the single bat case.

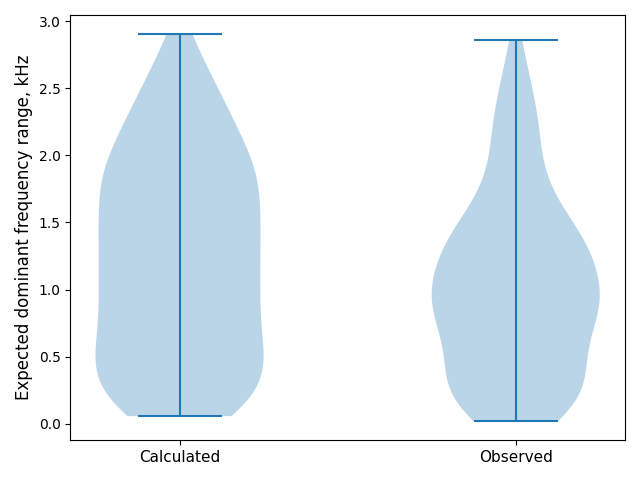


Figure 6.2: Calculated (left) and observed (right) dominant frequency range for a single bat flying past the microphone. The calculated and observed ranges match fairly well, indicating the broad processes behind the observed dominant frequency range have been captured. Violinplots describe data distribution, and horizontal bars indicate the minimum and maximum values. The shape of the simulated distribution does not match exactly, and this may require further parametrisation of the flight speeds or individual call frequency distribution.

When two or more bats echolocate in the same volume, it is expected that the DF range will increase because of the unique acoustic fovea’s each bat has. What is the expected range increase when the two bats echolocate independently however? To understand the expected DF range when multiple bats are flying we simulated the case of two bats echolocating independently in the same volume. The acoustic fovea of both bats was randomly chosen, and so were their start and end speeds. The DF range for the two bat case was thus calculated over a series of 1,000 random parameter combinations to reveal the range of dominant frequency ranges expected in two bat cases. In the two bat case, without reference to when or which bat emitted the call.

Figure 6.3 shows the dominant frequency ranges expected from single and a pair of bats. The median difference of the multi-single DF ranges is expected to be around 3.9 kHz, even though there is a wide variation in the observed DF ranges. The experimentally observed multi-single DF range difference of ~2 kHz falls within the range difference shown in Figure 6.3, however more detailed parametrisation of the flight speeds and relative positions may lead to a better match of the observed data.

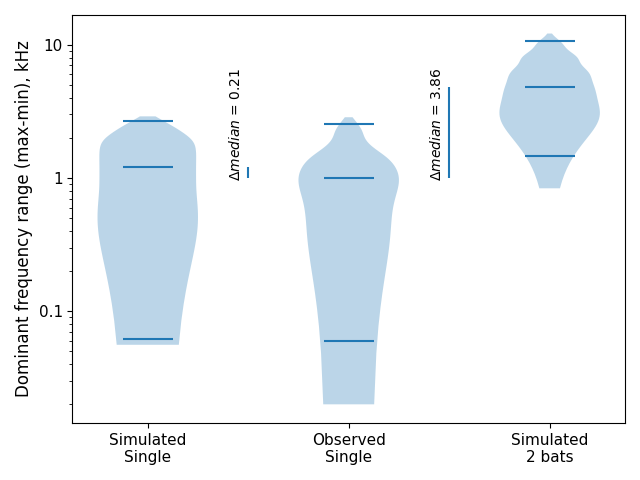


Figure 6.3: The distribution of dominant frequency ranges expected when a single bat echolocates (left), observed when a single bat (middle), and calculated when two bats fly. Violinplots indicate data distribution, horizontal lines indicate the 25, 50, and 75th percentiles. The y-axis is log10 scaled.

The code to implement this calculation is in the Combined analysis notebook.ipynb and its HTML version.

## 6.2 tFM echo-call overlap probabilities

The probability of a tFM echo overlapping with the tFM portion of another bat’s call was derived through simulation. The echo/call duration was fixed at 3.4ms and the inter-tFM duration was set to 40 and 50ms. A tFM echo was placed randomly in a time-span between 0-(echo + inter-tFM duration). A tFM call was also randomly placed in the same time-span, and a temporal overlap was checked. The random placement and overlap checking was done 20,000 times to derive a probability of echo-call overlap at the two inter-tFM intervals.

For *3* bats, an echo may be overlapped by two calls. The probability of echo-call overlap here is between 1.6 to 2.1%. Further details are in the Jupyter notebook titled tFM-overlaps.ipynb.

# References

1. Ivanova T. Important bat underground habitats (IBuH) in Bulgaria. Acta Zool Bulg. 2005;57(2):197–206.

2. Laurijssen D, Verreycken E, Geipel I, Daems W, Peremans H, Steckel J. Low-cost synchronization of high-speed audio and video recordings in bio-acoustic experiments. Journal of Experimental Biology. 2018;221(4).

3. Meltytech L. Shotcut. 2019. Available from: <https://shotcut.org/>

4. Fawcett K, Jacobs DS, Surlykke A, Ratcliffe JM. Echolocation in the bat, rhinolophus capensis: The influence of clutter, conspecifics and prey on call design and intensity. Biology Open. 2015;4(6):693–701.