Received: 31 October 2023 | Accepted: 12 July 2024

DOI: 10.1111/eea.13500

Entomologia Experimentalis et Applicata

ORIGINAL ARTICLE

Biotremology

VibePy: An open-source tool for conducting high-fidelity vibrational playback experiments

Alana J. Evora[1] | **Reginald B. Cocroft**[2] | **Shyam Madhusudhana**[3] | **Jennifer A. Hamel**[1]

[1]Department of Biology, Elon University, Elon, North Carolina, USA

[2]Division of Biological Sciences, University of Missouri, Columbia, Missouri, USA

[3]Centre for Marine Science and Technology, Curtin University, Perth, Western Australia, Australia

Correspondence
Jennifer A. Hamel, Department of Biology, Elon University, 100 Campus Drive, Elon, NC 27244, USA.
Email: jhamel2@elon.edu

Funding information
Elon University Lumen Program

Abstract

Many insect species communicate about activities central to their survival and reproduction via vibrational signals (i.e., waves that travel through solid substrates). One widely used and effective approach for testing hypotheses about communication is the playback experiment, in which stimuli are played to focal organisms and their responses are documented. Recent technological advances and efforts by vibrational communication researchers have diversified the hardware options available for use in such experiments, but proprietary software is still needed for high-fidelity playbacks. Here, we present an open-source software tool, VibePy, that (1) measures and compensates for undesired filtering and (2) calibrates playback amplitude. Because proprietary software licenses impose economic barriers that can limit access to research, we have developed VibePy in the open-source language Python. The functions provided by VibePy are a stepping stone toward increasing access to vibration research. Because the tool is open-source, we hope that the software will be expanded upon by others in the community of researchers studying vibrational communication and insect behavior.

KEYWORDS
economic barriers lifted, experimental methods, insect behavior, open-source software, Python, signals, substrate-borne vibration, technological advances, vibrational communication

INTRODUCTION

Interest in vibrational communication in insects has grown rapidly in recent decades. Within insects, vibrational signaling is taxonomically widespread (Virant-Doberlet et al., 2023) and used by an estimated 200 000 insect species, either alone or in combination with other signaling modalities (Cocroft & Rodríguez, 2005). Areas of study range from uncovering the functions of vibrational signals associated with mating (Čokl et al., 2000; De Luca & Morris, 1998; Hunt, 1993, 1994) and group-living behaviors (reviewed in Cocroft & Hamel, 2010) to investigating the effects of factors such as increasing temperatures (Conrad et al., 2017; Jocson et al., 2019; Shimizu & Barth, 1996) and vibrational anthropogenic noise on invertebrate signaling (reviewed in Classen-Rodríguez et al., 2021; Raboin & Elias, 2019). Vibrational signaling by agricultural pest insects has also been exploited as pest control via behavioral mechanisms (Mazzoni et al., 2019; Polajnar et al., 2015; Takanashi et al., 2019). This work is being conducted by a diverse community of researchers in widespread geographical locations (for examples, see authors and affiliations in Cocroft, Gogala, et al., 2014; Hill et al., 2019).

Studies of vibrational communication often employ vibrational playback experiments, in which substrate-borne stimuli are recorded and played to focal individuals, whose responses to the stimuli are then documented. The costs of the hardware and software used in vibrational playbacks have historically been barriers to researchers who are new to vibrational communication research. However, a broad array of actuators can play vibrational stimuli, including mini-shakers constructed from small speakers, linear resonant actuators (LRAs), and piezo-electric actuators. Different actuators vary widely in cost as well as in characteristics of their performance, such as frequency range and maximum displacement (Nieri et al., 2022; Wood & O'Connell-Rodwell, 2010). It is possible to conduct a vibrational playback experiment using an inexpensive actuator, but there can be performance gaps between high-cost and low-cost equipment. Furthermore, even a high-cost actuator will produce a stimulus that differs from the original, recorded stimulus in amplitude and its frequency spectrum unless filtering by the actuator and substrate are taken into account (Nieri et al., 2022). However, any of the commonly used vibration actuators, regardless of cost, can produce high-fidelity playbacks using custom-written software program code.

A custom-written script in MATLAB (Cocroft, Hamel, et al., 2014; Michael et al., 2019) addresses two challenges associated with vibrational playback experiments, and in doing so, it also closes the potential performance gaps mentioned above. First, the script measures and compensates for any undesirable filtering of the stimulus by the playback devices and/or substrate, and it then calibrates the stimulus amplitude for playback to the focal organism. Addressing both challenges is important because distortion of a stimulus may impact the responses of the study subject. For example, because mating signals may be species-specific in their spectral parameters (Cocroft et al., 2010), conducting a playback experiment to test individual responses to mating signals requires that any filtering should be measured and compensated for, because a distorted stimulus may not be recognizable to the focal organism as a conspecific signal.

Why is software needed to measure and compensate for filtering? In vibrational playback experiments, each substrate–device combination acts as a unique filter, attenuating some frequencies more than others. For example, if we consider plants as substrates, filtering varies among and within plant species (Cocroft, Hamel, et al., 2014; Elias & Mason, 2014), among parts of individual plants (Čokl et al., 2005; McNett & Cocroft, 2008), and even among locations on the same individual leaf or stem (Čokl et al., 2004; Magal et al., 2000; Michelsen et al., 1982). There can be additional filtering by the playback device: most playback devices—even calibrated ones—amplify some frequencies and attenuate others once they are attached to a load. Finally, the devices used to impart and detect the stimuli may alter the transmission and filtering properties of the substrate by loading mass on to the substrate. These filtering effects cannot be generalized or estimated across substrates and devices, so they must be measured and compensated for on a case-by-case basis. Similarly, although the desired playback amplitude may be typical of what the organism would detect in nature, playback amplitude is specific to each experiment and setup, because amplitude is substrate-dependent (Cocroft, Gogala, et al., 2014).

The custom-written script provided in Michael et al. (2019) enables researchers to perform high-fidelity playbacks. However, accessibility to this tool is limited, because the use of MATLAB requires a proprietary software license, which may present an economic barrier to individuals who want to use this approach. Here, we introduce VibePy, a tool that achieves the same functions as the custom-written script in MATLAB, but which is written using the open-source programming language Python, making the resulting script available to anyone who wants to use or edit it. VibePy measures and compensates for filtering, calibrates playback amplitude, and enables low-cost playback devices to perform comparably to high-cost ones. We chose Python for several reasons. Python libraries such as NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020) provide strong support for scientific computation and digital signal processing, and these libraries can be used to achieve the same basic functions as those in MATLAB. Python does not have complex syntax and prioritizes readability, increasing the likelihood that researchers with little programming experience will be able to expand upon the tool and tailor it to their needs. As a cross-platform language, Python can also be used on different kinds of computers and operating systems (e.g., RaspberryPi OS, MacOS, and Windows). Below, we describe the use of VibePy for vibrational playback experiments and demonstrate its functionality.

MATERIALS AND METHODS

Hardware

We demonstrate VibePy's functionality below using devices commonly employed in vibrational playback experiments. We ran VibePy and played the resulting stimuli from a MacBook Pro 16″ that was running the Ventura v.13.6 OS. We imparted vibrations into a substrate using a LRA (Samsung, 1 cm diameter) attached with accelerometer mounting wax to the stem of a potted *Albizia julibrissin* Durazzini (Fabaceae, Mimosoideae) plant 5 cm from the sensor. We detected the vibrations that were played through the plant stem using a small accelerometer (Model 352A24, weight 0.8 g, frequency range: 0.8–10 kHz ± 10%; PCB Piezotronics, Depew, NY, USA), also affixed to the plant stem with accelerometer wax and powered by a signal conditioner (Vibrametrics 9002). Both the actuator and sensor were connected to an audio interface (IK Multimedia iRig Pro, with input calibrated to 1:1 using a digital oscilloscope), which in turn was connected to the laptop. Our setup also included cables for connecting equipment, and we isolated the setup from unwanted vibrational noise by placing it on a Vibraplane 5704 vibration isolation table (Kinetic Systems, Boston, MA, USA). We note that excellent resources are available to help researchers work through equipment choices and other considerations important for vibrational playback experiments: Michael et al. (2019) provides a detailed guide to constructing a hardware setup for vibrational playbacks; Nieri et al. (2022) provides a comparative guide to actuators and sensors, with examples of how software intervention may make low- and high-cost actuators functionally equivalent; and Cocroft, Gogala, et al. (2014) explores many of the challenges and solutions of vibrational playback experiments.

Software

VibePy is implemented in Python3 and is freely available for download online (doi: 10.5281/zenodo.10059888). It depends on the following freely available open-source packages: sounddevice (Geier, 2023) to play and record signals; SoundFile (Bechtold, 2020) to read and write data to the Waveform Audio Format; NumPy (Harris et al., 2020) and SciPy (Virtanen et al., 2020) for performing data/numerical computations; and Matplotlib (Hunter, 2007) for data visualization. First-time users should consult the "Readme" in the VibePy download for instructions on how to run the application.

Set up VibePy

VibePy should be used after the experimental setup has been assembled and attached to the substrate. Repositioning the playback device, sensor, or substrate may affect signal transmission by changing how the vibrational waves propagate through the substrate and how they are detected by the sensor, which both influence filtering and overall amplitude.

When VibePy is run, the user defines which modules are to be used and enters a list of hardware parameters (Table 1) and signal parameters (Table 2). Alternatively, the user may provide the name of a text file in which these parameters are saved. A template for a parameter file is included in the VibePy download, called "testing_parameters.txt." Providing a custom-named parameter file allows the user to easily document and reuse parameters.

Measure and compensate for filtering

When this module is executed, the user decides whether to compensate again (Step 5), and the other steps will be automatically executed by the software.

1. *Play and record noise*: The signal parameters are used to generate 2 s of *Noise*, which is then played and recorded.
2. *Measure for filtering*: Amplitude spectra of the *Noise* and *Recorded noise* are calculated. The *Noise* has energy distributed equally across all frequencies in the specified range and the spectrum of the *Recorded noise* will likely not match that of the *Noise*, due to filtering by the playback device and substrate. Filtering, or the change in the signal as it travels, can be calculated as the amplitude spectrum of the *Recorded noise* (output) divided by the amplitude spectrum of the *Noise* (input).
3. *Compensate for filtering*: A compensation ratio is calculated by dividing the *Noise* by the *Recorded noise*, narrowed to only include the frequency range of interest defined by the *Low frequency* and *High frequency*, and used to generate a digital filter. This digital filter (i.e., compensation filter) is the inverse of the filtering by the playback device and substrate. When the compensation filter is applied to a stimulus, the compensation filter will amplify frequencies that the playback device and substrate attenuate, and attenuate frequencies that the playback device and substrate amplify. The compensation filter also eliminates frequencies outside of the frequency range of interest.
4. *Play and record compensated noise*: To determine whether the compensation filter accurately compensates for filtering by the playback device and substrate, the compensation filter is applied to the *Noise*, and the resulting *Compensated noise* is played and recorded.
5. *Decide whether to compensate again*: Amplitude spectra of the *Noise* and the *Recorded, compensated noise* are displayed, and the user is asked whether they want to compensate again. If the compensation filter is accurate, the shape of these amplitude spectra should approximately match, and another round of compensation is not necessary. The average and maximum differences (in dB) between the spectra of *Noise* and *Recorded, compensated noise* are also provided by VibePy to help a user make this assessment. If the compensation filter is not accurate, another round of compensation may be necessary. Note that the overall amplitude of the *Recorded, compensated noise* may differ from that of the *Noise* and/or the *Target amplitude*, because amplitude has not been calibrated yet.

TABLE 1 Hardware parameters of instruments used in the experimental setup.

Input device number	The number associated with the audio device or soundcard that the sensor is connected to in the experimental setup
Output device number	The number associated with the audio device or soundcard that the playback device is connected to in the experimental setup. The numbers for input and output can be different or the same (e.g., if the playback device and sensor are both connected to an audio interface)
Sensor channel	The number associated with the sensor channel on the *Input device*
Playback device	The number associated with the playback device channel on the *Output device*
Sensor type	The number associated with the type of sensor used in the experimental setup within the list of sensor types

Note: VibePy will automatically provide lists of available audio devices and soundcards and sensor types.

TABLE 2 Signal parameters of the stimulus that will be played to the focal organism in the playback experiment.

Stimulus file	The name of the file that will be used for the playback experiment. This file should be located in the same folder as VibePy. Note that very large files may require longer times to compensate and calibrate, because of processing time involved with fast Fourier transforms (fft)
Sampling rate (Hz)	The number of samples taken per s to create a digital recording of a continuous signal. The sampling rate should be at least twice the value of the *High frequency* to capture the full information of the signal, otherwise unwanted artifacts may be present in the recording (Sueur, 2018). The most common sampling rate in bioacoustics studies is 44 100 Hz (Sueur, 2018), which is sufficient for the frequency range of all vibrationally signaling animals with which the authors are familiar
fft size	The number of bins used for dividing a selection of sound into equal parts to calculate an amplitude spectrum
Low frequency (Hz)	The lowest frequency of interest in the experiment. This can be the lowest frequency that the focal organism is sensitive to or the lowest frequency of the stimulus used in the playback experiment.
High frequency (Hz)	The highest frequency of interest in the experiment. This can be identified using rationale similar to that for *Low frequency*. Together, *Low frequency* and *High frequency* will determine the frequency range over which the software will compensate for filtering[a]
Target amplitude (m/s^2, mm/s, or mV)	The peak amplitude of the original stimulus in the appropriate units. VibePy automatically indicates the units associated with the *Sensor type*

[a]Users should choose a frequency range that is not much broader than the frequencies of interest when defining *Low frequency* and *High frequency*. If the frequency range is too broad, VibePy will attempt to compensate for extraneous noise. If the frequency range includes frequencies over which the actuator can achieve little output, the intermediate frequencies will be lowered to achieve a flat frequency response. For these reasons, a broad frequency range potentially worsens the compensation performance.

The accuracy of the compensation filter is affected by the recorder's ability to convert the detected signal into a digital recording. Here, two factors are important: the detected amplitude of the signal at each sampling point and the number of units of data used to represent that amplitude digitally (i.e., bit depth and precision). If the amplitude at a given sample is very low, then a greater bit depth would be necessary to fully capture that signal at that sampling point. In acoustics, 32- and 64-bit depths are common; here, we use a 64-bit depth to make the *Recorded noise*. If the *Noise* is detected at very low amplitudes at some frequencies, such that the digital *Recorded noise* does not have enough precision to represent the signal, then the compensation filter will not be accurate. Therefore, multiple rounds of compensation may be necessary to increase the amplitude and create an accurate filter.

If the user elects to continue and compensate again, the process is modified in one way from the original process. The script is still trying to match the amplitude spectrum of the original *Noise* that has energy distributed equally across all frequencies. However, instead of playing this original *Noise* in Step 1, the script plays and records the *Compensated noise* from the previous compensation attempt. Then, the resulting *Recorded, compensated noise* is compared with the original *Noise*. Because the effects of filtering should get smaller each round, this process can be thought of as "fine-tuning." However, repeating compensation more than 2–3 times is unlikely to improve on previous results. Additional compensation may result in adverse effects, such as the signal becoming more dissimilar to the original *Noise* (i.e., distorted).

6. *Apply the final compensation filter to the stimulus*: If the user decides not to compensate again, the final compensation filter is applied to the *Stimulus*, and this *Compensated stimulus* is played and recorded. Amplitude spectra and waveforms of the *Stimulus* and the *Recorded, compensated stimulus* are displayed. The *Compensated stimulus* is saved by VibePy.

Calibrate stimulus amplitude

When this module is executed, VibePy tries to match the peak amplitude of the played vibrational stimulus to the *Target amplitude*. If a compensated stimulus is available, it will be used here. Otherwise, the original *Stimulus* will be used.

1. *Play and record the stimulus to find the peak amplitude*: The *Stimulus* is played and recorded. The peak amplitude (i.e., peak) is located within the *Recorded stimulus*, and the segment of the played *Stimulus* that produced the peak is isolated.
2. *Generate a ladder of modulated peaks*: A *Ladder* of modulated peaks is created by replicating the peak-containing segment 20 times and applying a multiplier to the amplitude of each segment. The multiplier increases the segment's amplitude at each step, such that the amplitude of the first segment has one-tenth of the amplitude of the original played segment and the last segment has a peak amplitude greater than the *Target amplitude*. The *Ladder* and the *Multipliers* are used in the following steps.
3. *Play and record the ladder of modulated peaks*: The *Ladder* is played and recorded. The recorded peak in each segment of the *Recorded ladder* is identified.
4. *Calculate the amplitude multiplier needed to achieve the target amplitude*: To estimate the *Multiplier* needed to achieve the *Target amplitude*, a linear regression is conducted between the *Ladder* amplitudes and the *Recorded peaks*. The results are used to calculate the

peak amplitude of the *Stimulus* that will yield the *Target amplitude*. If the *Target amplitude* is too high for the current equipment settings, the user will need to turn up the gain on the amplifier that is driving the actuator or choose a lower *Target amplitude*.

5. *Apply amplitude multiplier to the stimulus*: The amplitude *Multiplier* calculated in the previous step is applied to the *Stimulus*, and the *Calibrated stimulus* is played. Waveforms of the *Original stimulus* and *Recorded, calibrated stimulus* are displayed with the *Target amplitude* for reference. Then, corresponding amplitude spectra are displayed; note that the overall amplitude of the *Original stimulus* is adjusted to match that of the *Calibrated stimulus* to facilitate comparison of the spectra. The *Calibrated stimulus* is saved by VibePy.

Play vibrational stimulus

When this module is executed, the stimulus is played and recorded. If the compensated, calibrated stimulus is available, it will be used.

RESULTS

The playback stimulus file was a mating signal (Figure 1) recorded from a male *Umbonia crassicornis* (Amyot & Serville) (Hemiptera: Membracidae) on the stem of a potted *A. julibrissin* plant. The signal was recorded using the same make and model of accelerometer as was also used in the VibePy demonstration below, and the accelerometer was positioned 4 cm from the signaling male. Signal parameters used for the VibePy demonstration were a sampling rate of 44 100 Hz, fast Fourier transform (fft) size of 2048, low frequency of 70 Hz, high frequency of 4000 Hz, and target amplitude of 1.2 m/s^2 (within the amplitude range of a male signal recorded near the source).

Recall that VibePy generates, plays, and records 2 s of noise, after which it calculates a ratio between the amplitude spectra of the original *Noise* file and the *Recorded noise*. This ratio is used to construct a compensation filter, which is applied to the original *Noise* file. The resulting file is called *Compensated noise*. To evaluate whether the compensation filter is working, the *Compensated noise* is played through the same setup and substrate and the amplitude spectrum of the resulting recording is compared to that of the original *Noise* file. In this demonstration, after one round of compensation for filtering, the amplitude spectrum of the *Recorded, compensated noise* closely resembled that of the original *Noise* file (Figure 2), suggesting that the digital filter accurately compensated for unwanted filtering. The digital filter was then applied to the *Stimulus* to create the *Compensated stimulus* which was played and recorded. The amplitude spectrum of the *Recorded, compensated stimulus* closely resembled that of the original signal (Figure 3), again showing that the digital filter accurately compensated for unwanted filtering. As an

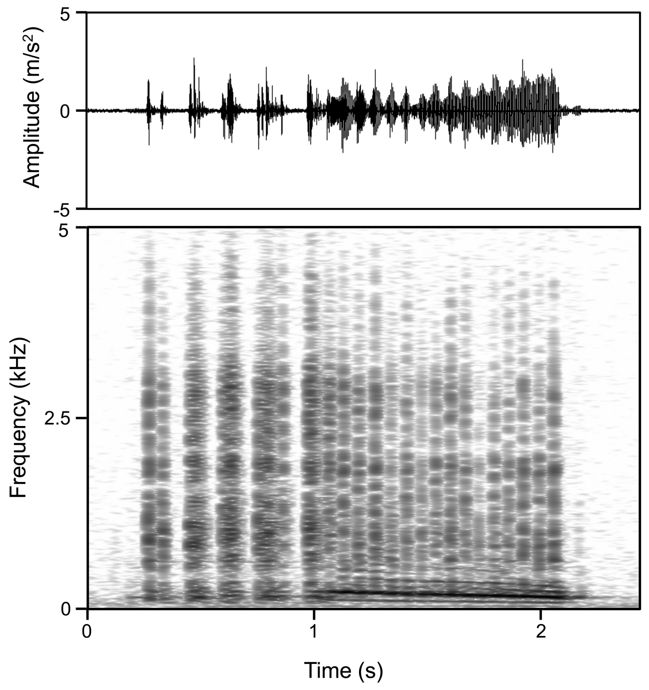

FIGURE 1 Waveform (top) and spectrogram (bottom) of the advertisement signal of a male *Umbonia crassicornis*, used as a playback stimulus for the subsequent figures.

additional test of efficacy, the authors generally record the playback of the compensated stimulus and listen to it along with the original; the two audio files should sound virtually indistinguishable aside from minor differences in background noise.

Note that although a LRA is an uncalibrated device, VibePy's approach of calculating an amplitude multiplier and applying it to the stimulus achieved a close match to the target amplitude (Figure 4).

The result of the compensation and calibration process is a modified stimulus file that will closely reproduce the original stimulus at the desired amplitude, when the experimenter leaves the equipment in place on the substrate where VibePy was run. We note that the “playback” function of VibePy is designed as a means of checking this modified stimulus file, rather than as a way to conduct a playback experiment. For a playback experiment, a user will likely want to use a program such as Audacity to create a file that contains some number of repetitions of the calibrated, compensated stimulus and silent intervals. Alternately, if the user has some Python coding experience, (s)he may construct a playback sequence that can be played directly from Python or used to generate and save a playback file that may be opened in other programs such as Audacity.

DISCUSSION

Playback experiments are a valuable, manipulative approach for testing hypotheses and predictions about

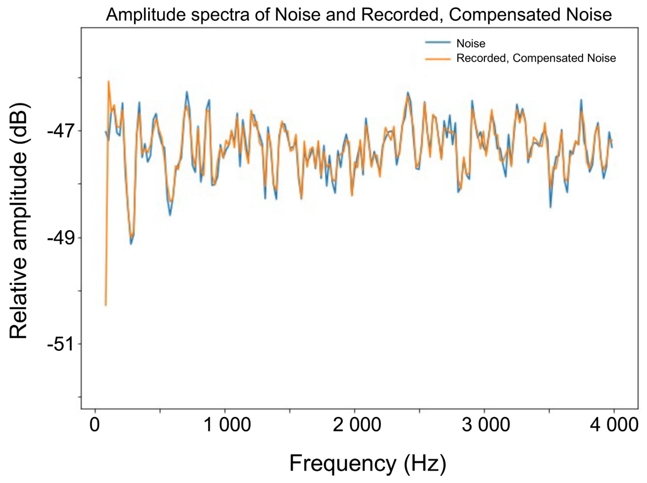

FIGURE 2 Illustration of the output from two rounds of frequency compensation (here, using a linear resonant actuator on a woody stem).

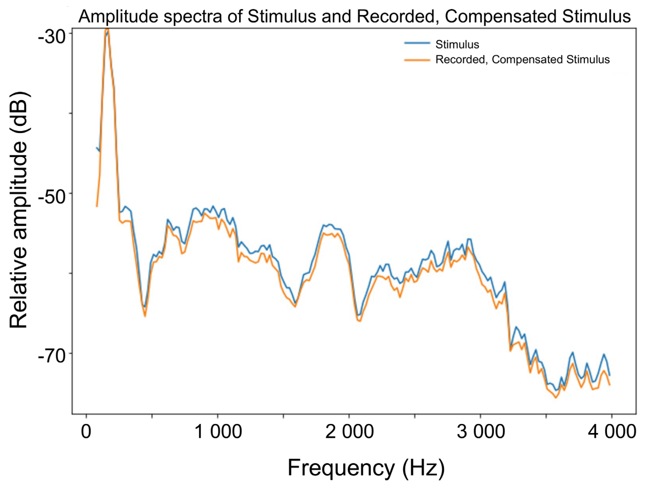

FIGURE 3 Amplitude spectrum of the stimulus (i.e., the advertisement signal of a male *Umbonia crassicornis*) as recorded on a host plant, with the spectrum of the compensated stimulus matching that of the original to within 3 dB across the selected frequency range (70–4000 Hz).

vibrational signal functions and organismal responses to vibrational stimuli. VibePy is an open-source Python tool for conducting high-fidelity vibrational playback experiments. The tool has three modules: (1) measuring and compensating for filtering in playback stimuli, (2) calibrating the amplitude of playback stimuli, and (3) playing the stimuli, incorporating the products of the first and second functions. Although we demonstrated VibePy's functionality using a MacBook Pro, we emphasize that Python is a cross-platform language, and VibePy functions well on many platforms, including a microcomputer (e.g., Raspberry Pi). Because Python emphasizes readability, uses a relatively simple syntax, and can be operated on various types of computers and operating systems, it is our hope that the vibrational communication community will use and expand upon the tool.

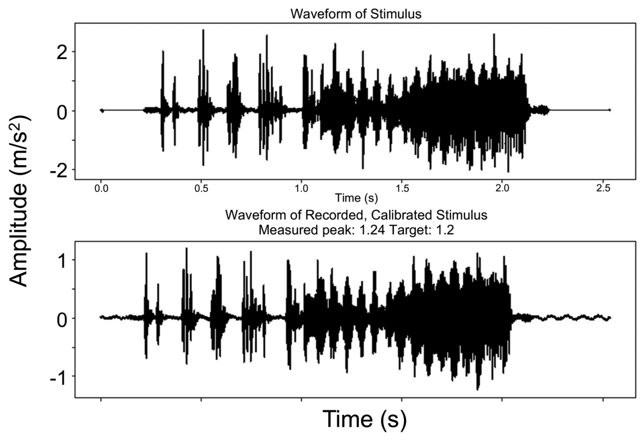

FIGURE 4 Waveforms of the original stimulus (top) and the compensated, amplitude-calibrated stimulus (bottom), as recorded on the host plant stem, showing that the recorded amplitude of the playback stimulus is a close match to the target amplitude. In this example, the amplitude of the original stimulus was greater than the target amplitude.

Before starting to plan and conduct a vibrational playback experiment, it is useful to understand methodological constraints and challenges (Cocroft, Hamel, et al., 2014). VibePy's functionality is ultimately limited by the hardware used in the experiment. For example, it can compensate for unwanted filtering within the frequency ranges of the actuator and sensor, but it cannot function beyond those ranges. It may also be helpful for a user to know that the script will need to be periodically rerun, even if the experimental setup has not been intentionally changed. Shifting in the positioning of an experimental setup is common, for example, either through temperature-related changes in the wax attachment or through changes in plant water content. The user needs to provide VibePy with a target amplitude in appropriate units (i.e., displacement, velocity, and acceleration) for the playback, which may require recording and calculating from focal organisms before beginning the playback experiment. Finally, the user should carefully consider the sequence of operations for their experiment. For instance, for VibePy to measure filtering and amplitude, noise and the stimulus are played through the focal substrate several times. In the example of conducting an experiment with plant-living insects, the researcher would likely want to avoid having the focal animals on the plant while compensation and calibration are taking place.

Some additional current constraints may be addressed through future releases of VibePy. These include that VibePy is currently only written for a single playback stimulus file: it plays the compensated and calibrated stimulus once and does not yet have options for looping or otherwise amending the playback stimulus. VibePy is also currently written to work with a single channel and cannot accommodate a multi-channel experiment. Additionally, VibePy is currently only available as a command-line tool; an updated version with a graphical user interface is planned.

We note that researchers working with airborne sound may find VibePy useful. For example, the script can compensate for the frequency response of a speaker, and this function should be helpful to many users (e.g., if a user wants a flat frequency response). Additionally, if a user is able to specify the mV level of a microphone, (s)he can select "uncalibrated sensor" and thereby also use the script to calibrate amplitude for the microphone. However, amplitude calibration from user-inputted microphone parameters is beyond the current scope of this script, because of the variation among microphone setups used in bioacoustics research. Such setups vary in microphone sensitivity (in different units), whether a microphone is analog or digital, whether a pre-amplifier is used and if so, the gain setting, etc.

Historically, limiting factors for vibration research have included both the economic costs and technical expertise associated with hardware and software needed to perform playback experiments. The study of vibrationally sensitive organisms is growing and includes contributions by researchers across geographic locations and institutional types. As such, a wide array of hardware has been used to perform vibration experiments, ranging from LRAs to modified speakers to calibrated minishakers, with prices also ranging over two orders of magnitude. Without software intervention, high-cost playback devices often outperform low-cost ones within the frequency range over which both can operate. Furthermore, even high-cost actuators may not deliver accurate playbacks without a compensation script, for reasons discussed in Nieri et al. (2022). Such software has been imperative in expanding the array of potential devices capable of performing accurate playback experiments; however, the software license associated with the MATLAB environment remains an economic barrier.

VibePy addresses the same hardware challenges as the custom-written script in MATLAB but also addresses this remaining economic barrier associated with software. Open-source software licenses make research tools available to a wide audience of researchers who can then use them, as well as contribute to expanding the tools' capabilities. Reducing the remaining economic barriers to vibration work via open-source tools can therefore help expand the community of vibrational communication researchers.

AUTHOR CONTRIBUTIONS

Alana J. Evora: Conceptualization (equal); investigation (lead); software (equal); validation (equal); writing – original draft (equal); writing – review and editing (equal). **Reginald B. Cocroft:** Conceptualization (equal); investigation (equal); software (equal); validation (equal); visualization (lead); writing – review and editing (equal). **Shyam Madhusudhana:** Software (equal); validation (supporting); writing – review and editing (supporting). **Jennifer A. Hamel:** Conceptualization (equal); investigation (supporting); writing – original draft (equal); writing – review and editing (lead).

ACKNOWLEDGMENTS

The authors thank Tim Forrest for motivating the development of the custom-written script in MATLAB and Sabrina Michael for inspiring the creation of a Python version of this tool. The authors also thank Brennan Copp and Ryan Mattfeld for insightful conversations as this project developed. This work was supported by the Elon University Lumen Program to AJE.

CONFLICT OF INTEREST STATEMENT

The authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available in GitHub at http://doi.org/10.5281/zenodo.10059888.

ORCID

Alana J. Evora https://orcid.org/0009-0002-3174-7839
Reginald B. Cocroft https://orcid.org/0000-0002-6991-5757
Shyam Madhusudhana https://orcid.org/0000-0002-4142-3881
Jennifer A. Hamel https://orcid.org/0000-0001-8505-3520

REFERENCES

Bechtold, B. (2020) SoundFile: an audio library based on libsndfile, CFFI and NumPy. Available from: https://github.com/bastibe/python-soundfile [Accessed 19th December 2023].

Classen-Rodríguez, L., Tinghitella, R. & Fowler-Finn, K. (2021) Anthropogenic noise affects insect and arachnid behavior, thus changing interactions within and between species. *Current Opinion in Insect Science*, *47*, 142–153. Available from: https://doi.org/10.1016/j.cois.2021.06.005

Cocroft, R.B., Gogala, M., Hill, P.S.M. & Wessel, A. (Eds.). (2014) *Studying vibrational communication, animal signals and communication*. Berlin, Heidelberg: Springer-Verlag. Available from: https://doi.org/10.1007/978-3-662-43607-3

Cocroft, R.B., Hamel, J., Su, Q. & Gibson, J. (2014) Vibrational playback experiments: challenges and solutions. In: Cocroft, R.B., Gogala, M., Hill, P.S.M. & Wessel, A. (Eds.) *Studying vibrational communication, animal signals and communication*. Berlin, Heidelberg: Springer, pp. 249–274.

Cocroft, R.B. & Hamel, J.A. (2010) Vibrational communication in the "other" social insects: a diversity of ecology, signals, and signal function. In: O'Connell-Rodwell, C. (Ed.) *Vibrational communication in animals*. Trivandrum, India: Research Signposts, pp. 47–68.

Cocroft, R.B. & Rodríguez, R.L. (2005) The behavioral ecology of insect vibrational communication. *BioScience*, *55*, 323–334. Available from: https://doi.org/10.1641/0006-3568(2005)055[0323:TBEOIV]2.0.CO;2

Cocroft, R.B., Rodríguez, R.L. & Hunt, R.E. (2010) Host shifts and signal divergence: mating signals covary with host use in a complex of specialized plant-feeding insects. *Biological Journal of the Linnean Society*, *99*, 60–72. Available from: https://doi.org/10.1111/j.1095-8312.2009.01345.x

Čokl, A., Prešern, J., Virant-Doberlet, M., Bagwell, G.J. & Millar, J.G. (2004) Vibratory signals of the harlequin bug and their transmission through plants. *Physiological Entomology*, *29*, 372–380. Available from: https://doi.org/10.1111/j.0307-6962.2004.00395.x

Čokl, A., Virant-Doberlet, M. & Stritih, N. (2000) The structure and function of songs emitted by southern green stink bugs from Brazil, Florida, Italy and Slovenia. *Physiological Entomology*, *25*, 196–205. Available from: https://doi.org/10.1046/j.1365-3032.2000.00187.x

Čokl, A., Zorović, M., Žunič, A. & Virant-Doberlet, M. (2005) Tuning of host plants with vibratory songs of *Nezara viridula* L. (Heteroptera: Pentatomidae). *Journal of Experimental Biology*, *208*, 1481–1488. Available from: https://doi.org/10.1242/jeb.01557

Conrad, T., Stöcker, C. & Ayasse, M. (2017) The effect of temperature on male mating signals and female choice in the red mason bee, *Osmia bicornis* (L.). *Ecology and Evolution*, *7*, 8966–8975. Available from: https://doi.org/10.1002/ece3.3331

De Luca, P.A. & Morris, G.K. (1998) Courtship communication in meadow katydids: female preference for large male vibrations. *Behaviour*, *135*, 777–794. Available from: https://doi.org/10.1163/156853998792640422

Elias, D.O. & Mason, A.C. (2014) The role of wave and substrate heterogeneity in vibratory communication: practical issues in studying the effect of vibratory environments in communication. In: Cocroft, R.B., Gogala, M., Hill, P.S.M. & Wessel, A. (Eds.) *Studying vibrational communication, animal signals and communication*. Berlin, Heidelberg: Springer, pp. 215–247. Available from: https://doi.org/10.1007/978-3-662-43607-3_12

Geier, M. (2023) Sounddevice: a Python module that provides bindings for the PortAudio library. Available from: https://github.com/spatialaudio/python-sounddevice/blob/master/sounddevice.py [Accessed 19th December 2023].

Harris, C.R., Millman, K.J., van der Walt, S.J., Gommers, R., Virtanen, P., Cournapeau, D. et al. (2020) Array programming with NumPy. *Nature*, *585*, 357–362. Available from: https://doi.org/10.1038/s41586-020-2649-2

Hill, P., Lakes-Harlan, R., Mazzoni, V., Narins, P.M., Virant-Doberlet, M. & Wessel, A. (Eds.). (2019) *Biotremology: studying vibrational behavior, animal signals and communication*. Cham: Springer International Publishing. Available from: https://doi.org/10.1007/978-3-030-22293-2

Hunt, R.E. (1993) Role of vibrational signals in mating behavior of *Spissistilus festinus* (Homoptera: Membracidae). *Annals of the Entomological Society of America*, *86*, 356–361. Available from: https://doi.org/10.1093/aesa/86.3.356

Hunt, R.E. (1994) Vibrational signals associated with mating behavior in the treehopper, *Enchenopa binotata* Say (Hemiptera: Homoptera: Membracidae). *Journal of the New York Entomological Society*, *102*, 266–270.

Hunter, J.D. (2007) Matplotlib: a 2D graphics environment. *Computing in Science & Engineering*, *9*, 90–95. Available from: https://doi.org/10.1109/MCSE.2007.55

Jocson, D.M.I., Smeester, M.E., Leith, N.T., Macchiano, A. & Fowler-Finn, K.D. (2019) Temperature coupling of mate attraction signals and female mate preferences in four populations of *Enchenopa* treehopper (Hemiptera: Membracidae). *Journal of Evolutionary Biology*, *32*, 1046–1056. Available from: https://doi.org/10.1111/jeb.13506

Magal, C., Schöller, M., Tautz, J. & Casas, J. (2000) The role of leaf structure in vibration propagation. *The Journal of the Acoustical Society of America*, *108*, 2412–2418. Available from: https://doi.org/10.1121/1.1286098

Mazzoni, V., Nieri, R., Eriksson, A., Virant-Doberlet, M., Polajnar, J., Anfora, G. et al. (2019) Mating disruption by vibrational signals: state of the field and perspectives. In: Hill, P.S.M., Lakes-Harlan, R., Mazzoni, V., Narins, P.M., Virant-Doberlet, M. & Wessel, A. (Eds.) *Biotremology: studying vibrational behavior, animal signals and communication*. Cham: Springer International Publishing, pp. 331–354. Available from: https://doi.org/10.1007/978-3-030-22293-2_17

McNett, G.D. & Cocroft, R.B. (2008) Host shifts favor vibrational signal divergence in *Enchenopa binotata* treehoppers. *Behavioral Ecology*, *19*, 650–656. Available from: https://doi.org/10.1093/beheco/arn017

Michael, S.C.J., Appel, H.A. & Cocroft, R.B. (2019) Methods for replicating leaf vibrations induced by insect herbivores. *Methods in Molecular Biology*, *1991*, 141–157. Available from: https://doi.org/10.1007/978-1-4939-9458-8_15

Michelsen, A., Fink, F., Gogala, M. & Traue, D. (1982) Plants as transmission channels for insect vibrational songs. *Behavioral Ecology and Sociobiology*, *11*, 269–281. Available from: https://doi.org/10.1007/BF00299304

Nieri, R., Michael, S.C.J., Pinto, C.F., Urquizo, O.N., Appel, H.M. & Cocroft, R.B. (2022) Inexpensive methods for detecting and reproducing substrate-borne vibrations: advantages and limitations. In: Hill, P.S.M., Mazzoni, V., Stritih-Peljhan, N., Virant-Doberlet, M. & Wessel, A. (Eds.) *Biotremology: physiology, ecology, and evolution, animal signals and communication*. Cham: Springer International Publishing, pp. 203–218. Available from: https://doi.org/10.1007/978-3-030-97419-0_8

Polajnar, J., Eriksson, A., Lucchi, A., Anfora, G., Virant-Doberlet, M. & Mazzoni, V. (2015) Manipulating behaviour with substrate-borne vibrations – potential for insect pest control. *Pest Management Science*, *71*, 15–23. Available from: https://doi.org/10.1002/ps.3848

Raboin, M. & Elias, D.O. (2019) Anthropogenic noise and the bioacoustics of terrestrial invertebrates. *Journal of Experimental Biology*, *222*, 178749. Available from: https://doi.org/10.1242/jeb.178749

Shimizu, I. & Barth, F.G. (1996) The effect of temperature on the temporal structure of the vibratory courtship signals of a spider (*Cupiennius salei* Keys.). *Journal of Comparative Physiology. A*, *179*, 363–370. Available from: https://doi.org/10.1007/BF00194990

Sueur, J. (2018) *Sound analysis and synthesis with R*. Cham: Springer International Publishing. Available from: https://doi.org/10.1007/978-3-319-77647-7

Takanashi, T., Uechi, N. & Tatsuta, H. (2019) Vibrations in hemipteran and coleopteran insects: behaviors and application in pest management. *Applied Entomology and Zoology*, *54*, 21–29. Available from: https://doi.org/10.1007/s13355-018-00603-z

Virant-Doberlet, M., Stritih-Peljhan, N., Žunič-Kosi, A. & Polajnar, J. (2023) Functional diversity of vibrational signaling systems in insects. *Annual Review of Entomology*, *68*, 191–210.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D. et al. (2020) SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*, 261–272. Available from: https://doi.org/10.1038/s41592-019-0686-2

Wood, J. & O'Connell-Rodwell, C. (2010) Studying vibrational communication: equipment options, recording, playback and analysis techniques. In: Wood, J. & O'Connell-Rodwell, C. (Eds.) *The use of vibrations in communication: properties, mechanisms and function across taxa*. Trivandrum, India: Transworld Research Network, pp. 249–274.

How to cite this article: Evora AJ, Cocroft RB, Madhusudhana S & Hamel JA (2024) VibePy: An open-source tool for conducting high-fidelity vibrational playback experiments. *Entomologia Experimentalis et Applicata 00*: 1–8. https://doi.org/10.1111/eea.13500