guitarsounds : A Python package to visualize harmonic sounds for musical instrument design

Olivier Chabot ⓘ [1] **and Louis Brillon**[1]

1 École de lutherie Bruand, Montréal, QC, Canada

Summary

The `guitarsounds` Python package can be used to efficiently visualize relevant features of harmonic sounds and is mainly developed to compare musical instrument design iterations. `guitarsounds` is wrapped around the implementation of common signal processing features, which are manly taken from `numpy` (Harris et al., 2020) and `scipy` (Virtanen et al., 2020). The signal processing features are augmented to perform the comparative analysis of transient harmonic sounds. Such sounds are defined as having a clear onset, and a frequency-amplitude distribution clearly concentrated around partials. Consequently, while the package is named `guitarsounds`, its analysis framework can be used with any harmonic sound, such as those produced by a piano, or a percussion instrument.

The `guitarsounds` package is divided in two main components. First, the package is developed around a convenient object-oriented Application Programming Interface (API) which can be used to extract features from sounds and visualize them according to the user's needs. Then, a graphical user interface (GUI) makes most of the features of `guitarsounds` available to users less knowledgeable in programming. `guitarsounds` is meant to be used with the Jupyter Notebook interface as to allow interactively exploring the sound data, either with the API or the GUI.

The main features of guitarsounds are:

- Automate the loading, conditioning and normalization of multiple sound files to meaningfully compare their features.
- Visualize sounds features relevant to musical instrument design, such as:
  - Linear and logarithmic time envelope
  - Octave bands Fourier transform
  - Time-dependent damping
- Divide sounds in frequency bands to analyze variations in temporal behaviour for different frequency ranges.
- Extract the Fourier transform peaks of a harmonic signal using a custom peak finding algorithm.
- Extract numerical values for certain features such as the Helmholtz cavity frequency of a guitar.
- Provide an easy-to-use signal processing API to compute new features meeting specific needs by providing acess to lower level features and handling the differences between sound files, such as the file sample rate.

Specifically, the API provides 5 classes nested together : `SoundPack`, `Sound`, `Signal` and `Plot`. Starting from the innermost class, the `Plot` class handles the low level plotting of specific features, such as plotting the Fast Fourier Transform (FFT) of a sound file. The `Signal` class is used to store the data of an array corresponding to a single signal. For example, if a sound file is read and filtered, the array resulting from the filtering operation will be stored in a new instance of the `Signal` class. An instance of the `Plot` class is constructed for each `Signal`

44 class instance and stored as an attribute of the `Signal` class. The `Signal` class contains all
45 the features relying only on the data of a single sound signal, such as the computation of the
46 envelope. The `Sound` class is used to store all the information corresponding to a single sound
47 file. When a `.wav` file is read using `guitarsounds`, all the processing is handled by the `Sound`
48 class, such as truncating, filtering or normalizing the sound signal. The `Sound` class provides
49 the features relying on more than one `Signal` instance, but still using the information from a
50 single sound file, such as the power distribution of a sound across different frequency bands.
51 Finally, the `SoundPack` class contains multiple `Sound` class instances and provides the features
52 used to compare the data between different sound files. The `SoundPack` methods are divided
53 between methods developed to compare two sounds and methods developed to compare an
54 arbitrary amount of sounds. As an example, the method plotting the FFT of two sounds in a
55 mirror configuration can only be called if the `SoundPack` was constructed using exactly two
56 sounds, whereas the method showing a table of the different sound fundamental frequencies
57 can be called for a `SoundPack` instance created using an arbitrary number of `Sounds`.

58 As an example, the log-time envelope, a feature representing the amplitude of a sound with a
59 higher definition in time at the start of the sound, can be computed using `guitarsounds`. By
60 plotting this feature for the same note played on two instruments, the dynamic response of the
61 instruments can be compared for a specific excitation frequency. A code snippet comparing
62 the log-time envelope of two sounds is presented below with the associated output in Figure 1.
63 In the following code, the `SoundPack` object is first instantiated from the specified sound files.
64 For each file, a `Sound` class instance is created and conditioned. In the conditioning procedure,
65 the signal is first resampled to have a sample rate equal to 22050 Hz, this is important to
66 ensure all the features compared between sounds are computed using the same sample rate.
67 The sound is then trimmed so that the beginning of the onset is at 100 ms, as can be seen
68 on Figure 1. To ensure compared sounds have the same length, the ends of the sounds are
69 trimmed so that each sound has the same number of samples as the shortest sound. The
70 `guitarsounds` package relies on `matplotlib` (Hunter, 2007) for all its vizualisation features.
71 Thus, users familiar with `matplotlib` objects can tune the figures created by `guitarsounds`
72 to their needs.

```python
import guitarsounds
import matplotlib.pyplot as plt

# Use guitarsound to compare the log-time envelope of two sounds
soundfile1 = "example_sounds/Wood_Guitar/Wood_A5.wav"
soundfile2 = "example_sounds/Carbon_Guitar/Carbon_A5.wav"
# Specify names to be used in the legend of the plot
mysounds = guitarsounds.SoundPack(soundfile1, soundfile2, names=["wood", "carbon"])
mysounds.plot("log envelop")

# Access the matplotlib Figure and Axes objects created by guitarsounds
plt.gca().set_title("My sound comparison") # To change the title
plt.gcd().savefig("log_envelope_compare") # To save the figure
```
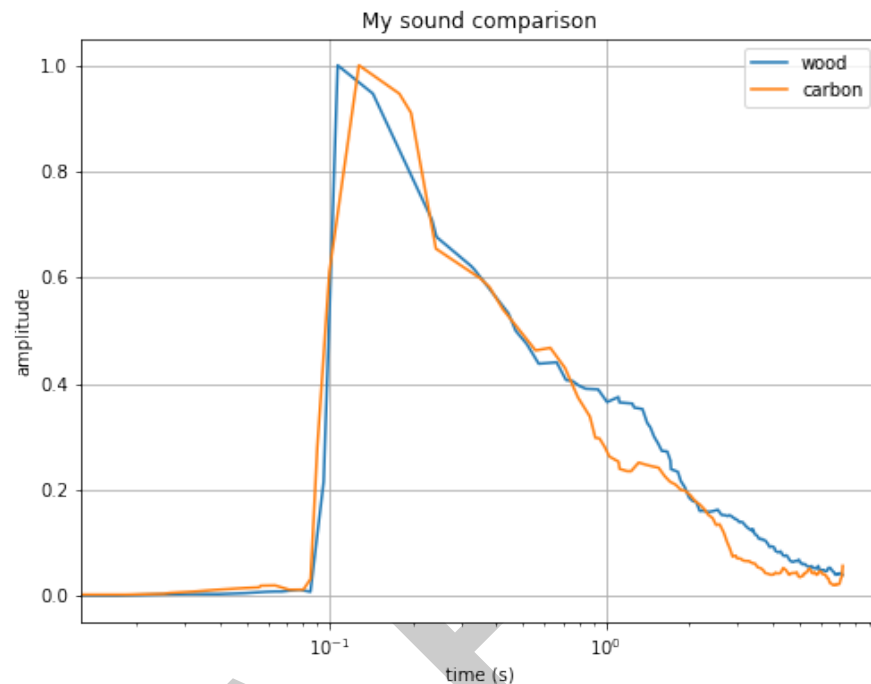
Figure 1: Output of the code snippet comparing the log-time envelope of two sounds.

Statement of need

guitarsounds was developped to meet the needs of the Bruand lutherie school, more precisely as a tool to visualize and compare the sounds of different guitar designs using custom sound features. The guitarsounds package was used in previous academic work (in press) to investigate the difference between two guitar designed using an innovative numerical prototyping method based on topological optimization. In the scope of this research, guitarsounds allowed the measurement of specific sound features such as the slope of the peaks in the signal Fourier transform, computed using a linear regression. This feature is related to the instrument's tone (Sumi & Ono, 2008), and was used to study the differences between the two guitar designs. guitarsounds is also used in ongoing research at the Bruand lutherie school to manage sound data in a project where guitar sounds are generated with random values for specific features, to provide data for a psycho-acoustic study. The guitarsounds API is also used to give an introduction to programming for data analysis to the school's students and in the teaching activities, as a tool to visualize the physical phenomena involved in the sounds produced by guitars, such as the Hemlotz cavity frequency of an instrument. The GUI was included in the package as knowledge or interest in programming isn't expected in the luthier's training. A screen capture of the GUI is shown in Figure 2.

The features of guitarsounds differs from existing packages in its ability to be both used alone to produce decent figures with a minimal number of lines of code and as a tool in a Python data visualization stack where the sound specific needs can be handled by guitarsounds. There exists an overlap between guitarsounds, and the librosa (McFee et al., 2015) Python package for music analysis, however librosa is not a dependency of guitarsounds, and the latter is more focused on feature extraction for machine learning application and lacks features tailored to harmonic sound analysis and integrated comparison of sounds. The librosa package is also aimed at an audience with a more developed knowledge of Python programming.
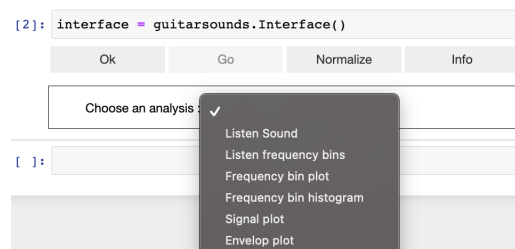
Figure 2: Graphical user interface in the Jupyter Notebook environment.

References

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

McFee, B., Raffel, C., Liang, D., Ellis, D., McVicar, M., Battenberg, E., & Nieto, O. (2015). *Librosa: Audio and Music Signal Analysis in Python*. 18–24. https://doi.org/10.25080/Majora-7b98e3ed-003

Sumi, T., & Ono, T. (2008). Classical guitar top board design by finite element method modal analysis based on acoustic measurements of guitars of different quality. *Acoustical Science and Technology*, *29*(6), 381–383. https://doi.org/10.1250/ast.29.381

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., Walt, S. J. van der, Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … Mulbregt, P. van. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, *17*(3), 261–272. https://doi.org/10.1038/s41592-019-0686-2