Iracema: a Python library for audio content analysis

Tairone N. Magalhães[1][*], **Felippe B. Barros**[1][†], **Mauricio A. Loureiro**[1]

[1]CEGeME - Center for Research on Musical Gesture and Expression – UFMG
Av. Antônio Carlos, 6627 – 31270-010, Belo Horizonte, MG

tairone@ufmg.br, felippebb@ufmg.br, mauricioloureiro@ufmg.br

Abstract. *This paper introduces the alpha version of a Python library called Iracema, which aims to provide models for the extraction of meaningful information from recordings of monophonic pieces of music, for purposes of research in music performance. With this objective in mind, we propose an architecture that will provide to users an abstraction level that simplifies the manipulation of different kinds of time series, as well as the extraction of segments from them. In this paper we: (1) introduce some key concepts at the core of the proposed architecture; (2) list the current functionalities of the package; (3) give some examples of the application programming interface; and (4) give some brief examples of audio analysis using the system.*

1 Introduction

Despite the fact that music performance is a central element to nearly every culture, its empirical study is relatively recent, with the seminal works dating back to the turn of the twentieth century. As stated by Clarke in [1], "only once methods had been developed to record either the sounds of performance, or the actions of instruments, was any kind of detailed [empirical] study possible". Over the last few decades, we have witnessed considerable growth in this field of study [2, 3], and the availability of new tools and technologies for extracting information from performances have played a pivotal role in this surge. We believe that the continuous development of more specialized tools to extract information from music performance, as well as better techniques for obtaining more meaningful representations of musical content, will be of crucial importance to continually support empirical research in music performance.

In this scenario, we introduce the alpha version of *Iracema*, a Python package for audio content analysis aimed at the empirical research on music performance. It provides functionalities for extracting patterns of manipulation of duration, energy, and spectral content from monophonic audio, specially for instruments such as clarinet, flute, trumpet, and trombone. Its development was motivated by research projects conducted at CEGeME[1], and was strongly inspired by a previous Matlab tool developed by the group, called Expan [4], which has not been released for public use.

In contrast to instruments like guitar or piano, in which the excitation that produces sound only happens at the beginning of a note[2], in woodwind and brass instruments, the player continuously feeds energy into the system, by means of high pressure air from his lungs. Therefore, due to the dynamic control that the player has over the acoustic properties of the sound, a single note might contain a lot of important expressive information, e.g., timbral manipulations, or dynamic intensity variations. It is harder to extract this kind of information from polyphonic musical signals, such as a full orchestral recording, than from signals of a single source. So a reasonable approach is to use monophonic recordings to better understand them. Another characteristic of the instruments of our interest, is that they can produce very soft attacks, which makes the precise detection of note onsets tricky, especially avoiding the occurrence of false positives. Thus, the techniques implemented on Iracema focus mainly on the extraction of information from monophonic sounds with soft note attack[3].

2 Iracema

Iracema is licensed under the GNU General Public License v3.0, and its source code can be freely obtained at `https://github.com/cegeme/iracema`. To obtain more detailed information about the library, like usage examples, more information about the feature extractors available, library modules, and extensive documentation of the API, check the online documentation, which is available at `https://cegeme.github.io/iracema`.

Iracema uses NumPy arrays for storing and manipulating data, providing a new level of abstraction on top of such objects [5]. It also wraps some functionalities from SciPy [6] to provide methods with a more natural interface for audio content extraction operations, as well as compatibility with Iracema's objects.

2.1 Architecture

Software architecture refers to the set of structures needed to reason about a system. These structures are comprised of software elements, relations among them, and properties of both elements and relations [7]. This section will discuss some import aspects of Iracema's architecture and offer an overview of the elements that compose the core functionalities of the library.

Audio content analysis systems rely on the manipulation of dynamic data, i.e., data that represent an attribute's changes over time. Thus, *time series* is a fundamental element in Iracema's architecture. The starting

[*]Supported by CAPES.
[†]Supported by CNPq.
[1]http://musica.ufmg.br/cegeme/

[2]I.e., the plucking of strings in a guitar or a hammer hitting the strings of a piano.
[3]Monophonic sounds with soft note attacks motivated the development of the system, but the reader should be aware that some functionalities of Iracema could also be applied to polyphonic or percussive sounds.

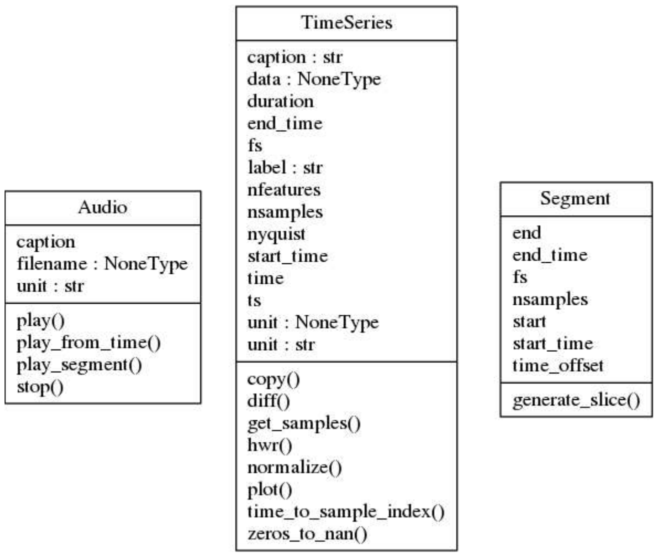

Figure 1: Diagram showing the core classes of Iracema.

point for any task performed by the system is the *audio time series*, from which other kinds of time-related data will be extracted. The transformation of time series into other time series, to obtain more meaningful representations of the underlying audio, is a common behavior of audio content analysis systems, usually called *feature extraction*. The implementation of such extractors usually depends on some recurrent types of operations, like applying sliding windows to a series of data, for example. In Iracema, these operations are called *aggregation* methods.

Sometimes it will be necessary to deal with a specific excerpt of a time series, such as a musical phrase or a note. There is another important element in the architecture, called *segment*, that can be used to delimit such excerpts. A user may sometimes specify the limits for a segment, within the time series, if he is already aware of its beginning and end; however, most of the time, users will expect the system to identify such limits by itself, a common kind of task in audio content extraction, known as *segmentation*.

Some of the aforementioned elements, like audio, time series, and segments have been implemented as classes, since they have intrinsic data (e.g., the samples of the time series, and the start/end of the segments) and behaviour (e.g., generating time vectors in time series or calculating indexes in segments). Figure 1 shows those classes in a diagram. The `Audio` class inherits the functionalities from `TimeSeries`, and add some specific behaviours (such as loading wave files). `Segments` provide a handy way to extract corresponding excerpts from time series of different sampling rates, since it performs all the necessary index conversion operations to extract data that coincide with the same time interval.

Other elements have been implemented as methods that take objects of those classes as input and output another object. For example, the method `fft` takes as input an `audio` object, a `window_size`, and a

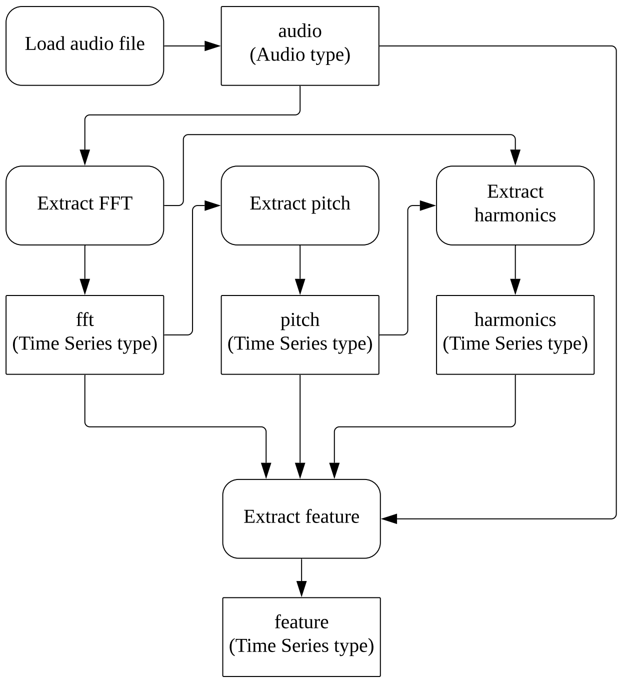

Figure 2: Extracting features from an audio file.

`hop_size`, and generates a time series in which each sample contains all the bins of the FFT for the interval corresponding to `hop_size`. Another example, the method `spectral_flux` will take a time series containing the result of an FFT operation as input and generate another time series containing the calculated spectral flux. Figure 2 shows a diagram that illustrates the typical workflow for performing basic feature extraction from audio files.

Segmentation methods will usually take `time_series` objects as input to output a list of segments (Figure 3). Then, these segments can be used to easily extract excerpts from time series objects (Figure 4), using square brackets (the same operator used in Python to perform indexing/slicing operations).

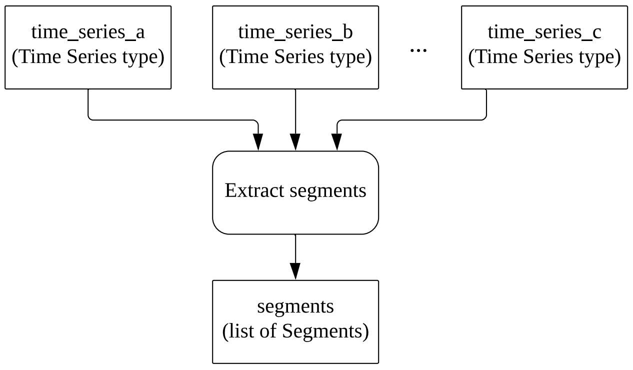

Figure 3: Extracting segments from time series.

2.2 Modules and functionalities

These are the modules that compose Iracema, and their respective functionalities:

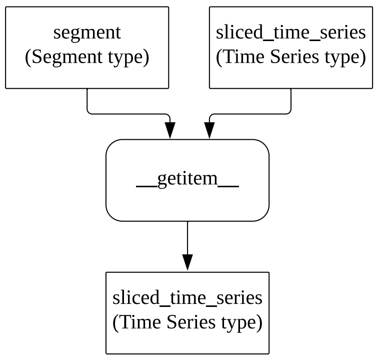

Figure 4: Using a segment to slice a time series.

- `timeseries`: contains the definition of the classes `TimeSeries` and `Audio`.
- `segment`: contains the definition of the classes `Segment` and `SegmentList`.
- `spectral`: contains methods for frequency domain analysis (currently the FFT);
- `pitch`: a few different models for pitch detection.
- `harmonics`: a model for extracting harmonic components from audio.
- `features`: contains methods with the implementation of several classic feature extractors.
- `segmentation`: methods for automatic audio segmentation.
- `plot`: contains several different methods for plotting time series data.
- `aggregation`: contains some common aggregation methods that can be useful for implementing feature extractors.
- `io`: subpackage containing IO methods, for loading/writing files, playing audio, etc.
- `util`: subpackage containing some useful modules for unit conversion, DSP, windowing operations,etc.

2.3 Pitch detection models

The module `pitch` contains models for pitch detection. At the time this paper was finished, two methods had been implemented, as well as an extra method that wraps a model from an external library. The following list shows the pitch methods available:

Harmonic Product Spectrum Measures the maximum coincidence for harmonics, based on successive downsampling operations on the frequency spectrum of the signal [8].

Expan Pitch Based on the algorithm implemented in Expan [4]. It chooses the highest peaks in the frequency spectrum of a signal as potential candidates, and then extract their theoretical harmonics. The candidate with the higher harmonic energy is chosen as the fundamental frequency.

CREPE Based on a deep convolutional neural network operating directly on the time-domain waveform input [9]. This is a wrapper that uses an external library.

2.4 Feature extractors

These are the methods available in the module `features`:

Peak Envelope Extracts the envelope of the waveform by extracting the peaks in the amplitude for each analysis window.

RMS Calculate the root mean square of a time-series. This is usually a good better choice for extracting the envelope of an audio signal, since it is more closely related to our perception of intensity that the peak envelop.

Zero-crossing The zero crossing is a measure of how many time-series a signal crosses the zero axis in one second. It gives some insight on the noisiness character of a sound.

Spectral Flatness Gives an estimation of the noisiness/sinusoidality of an audio signal. It might be used to determine voiced/unvoiced parts of a signal [10].

HFC Measures of the amount of high frequency content of a time-series spectrum. It produces sharp peaks during attacks transients [11] and might be a good choice for detecting onsets in percussive sounds.

Spectral Centroid The spectral centroid is a well known timbral feature that is used to describe the brightness of a sound. It represents the center of gravity of the frequency components of a signal [12].

Spectral Spread Gives an estimation of the spread of the spectral energy around the spectral centroid [10].

Spectral Flux : Measures the amount of change between adjacent spectral frames [13].

Spectral Skewness Measures how symmetric is the distribution of the values for the spectral magnitudes around their arithmetic mean [14].

Spectral Kurtosis Measures if the distribution of the spectral magnitude values is shaped like a Gaussian distribution or not [14].

Spectral Rolloff The spectral rolloff is a measure of the bandwidth of the spectrum [14]. It is defined as the point in the spectrum bellow which a percentage k of the spectral energy is contained.

Spectral Entropy Measures the unpredictability or disorder in the distribution of the spectral energy [15].

Spectral Energy The total energy of a frame of the spectrum.

Harmonic Energy The total energy of the harmonic partials of a time-series.

Inharmonicity Determines the divergence of the time-series spectral components from an ideal harmonic spectrum.

Noisiness It is the ratio of the noise energy to the total energy of a signal. Represents how noisy a signal is (values closer to 1), as oposed to harmonic (values close to 0) [10].

Odd-to-Even Ratio It is the ratio between the energy of the odd and even energy harmonics of a signal.

3 Examples

This section shows some basic code examples for the library.

```
 1 import iracema
 2
 3 # loading audio file
 4 audio = iracema.Audio(
 5   "audio/03 - Clarinet - Fast Excerpt.wav")
 6
 7 # plotting waveform
 8 audio.plot()
 9
10 # playing audio
11 audio.play()
```

Listing 1: Loading audio and plotting waveform.

Loading audio files in Iracema is pretty straightforward, and the only thing that must be specified is a string containing the path to the audio file that should be loaded[4]. In code listed above, the initializer method for the class `iracema.Audio` shown in line 4 will load the content of the wave file into an `audio` object. Then, the object's `plot()` method (line 8) will display its waveform (shown in Figure 5), automatically setting some basic plot parameters, such as axis labels and title, by using metadata from the audio time series. In line 11, the method `play()` will reproduce the corresponding audio.

```
12 # calculate FFT
13 window_size, hop_size = 2048, 1024
14 fft = iracema.spectral.fft(
15     audio, window_size, hop_size)
16
17 # plot spectrogram
18 iracema.plot.plot_spectrogram(
19     fft, fftlim=(0,10000))
```

Listing 2: Calculating FFT and plotting spectrogram.

The method `iracema.spectral.fft()` shown in line 14 will calculate the FFT for the audio file, using a sliding window of size 2048, with 1024 samples of overlap. It will generate another time series object as output, which will contain multiple values per sample, one corresponding to each bin of the FFT. It will then be passed to the method `iracema.plot.plot_spectrogram()` in line 18, to obtain the visualization shown in Figure 6.

```
20 # extract the pitch and then the harmonics
21 pitch = iracema.pitch.hps(fft)
22 harmonics = iracema.harmonics.extract(fft, pitch)
23
24 # plot the harmonics over the spectrogram
25 iracema.plot.plot_spectrogram_harmonics(
26     fft, pitch, harmonics['frequency'],
27     fftlim=(0,12000))
```

Listing 3: Extracting pitch and harmonics.

The pitch will be extracted from the signal using the method `iracema.pitch.hps()` in line

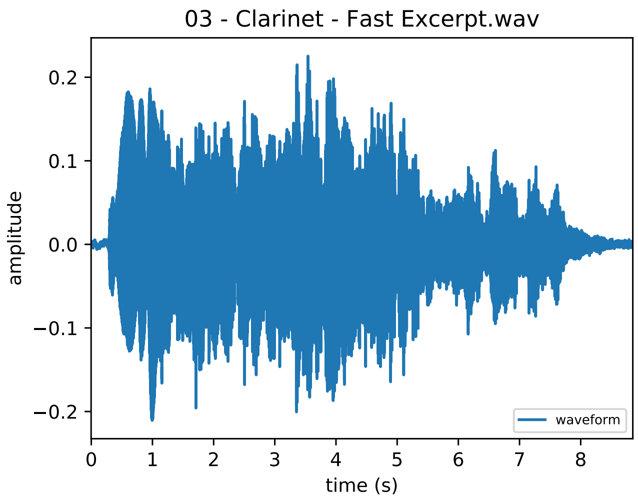

Figure 5: Waveform for the audio file loaded.

21, and then, in the next line, passed as a parameter to the method `iracema.harmonics.extract()`, along with the previously calculated FFT. The method for extraction of harmonics will extract 16 harmonics by default, but a different number could be specified, using the optional argument `nharm`. In line 25, both objects will be plotted over a spectrogram using the method `iracema.plot.plot_spectrogram_harmonics()`, resulting in the plot show in Figure 7.

```
28 # extract some features
29 sflux = iracema.features.spectral_flux(fft)
30 sflat = iracema.features.spectral_flatness(fft)
31 sc = iracema.features.spectral_centroid(fft)
32 no = iracema.features.noisiness(fft,
33     harmonics['magnitude'])
34 hfc = iracema.features.hfc(fft)
35
36 # plot waveform and other features
37 iracema.plot.plot_waveform_trio_and_features(
38     audio, features=(sflux, sflat, sc, no, hfc))
```

Listing 4: Feature extraction.

In lines 29-34, five different features will be calculated for the whole audio signal. Then, they will all be plotted (line 37), along with a visualization of the waveform, using the method `iracema.plot.plot_waveform_trio_and_features()` (Figure 8).

4 Future perspectives

The functionalities of the library will move towards feature extractors that can provide more meaningful representations of the information from music performance, from a musical point of view. The architecture proposed for Iracema and the feature extractors mentioned in this article form the basis for the development of such representations, which will be included in the future stages of development of the tool.

Good models for note segmentation are essential for audio content extraction, so this is our major concern

[4]Iracema uses the library audioread [16] to load audio files, and can handle different audio file formats.

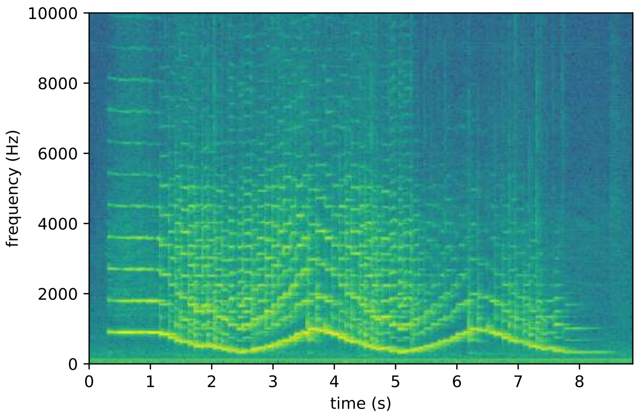

Figure 6: Spectrogram for the audio file loaded.

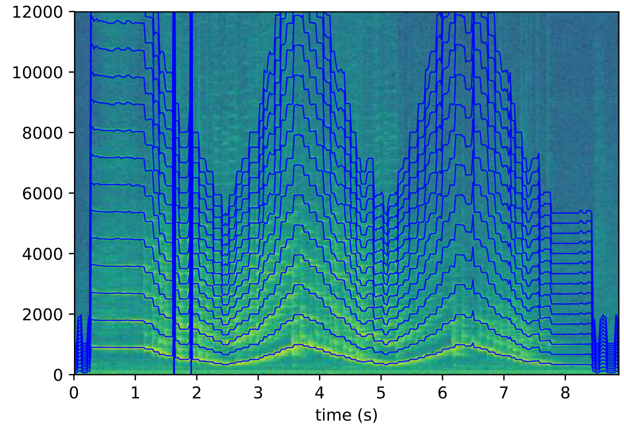

Figure 7: Frequency for the harmonics components extracted (the plot has been limited to the range 0-12 KHz using the parameter *fftlim*).

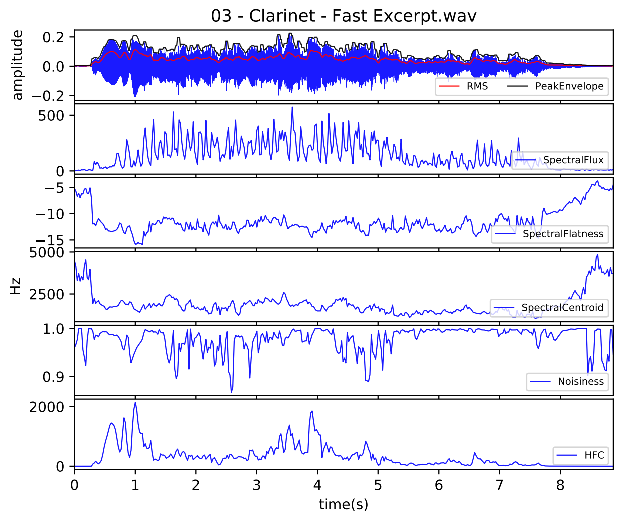

Figure 8: Five different features exctracted for the audio file loaded.

at the time. Although we have already implemented some basic models that use pitch and energy information to detect note onsets, sometimes they produce some false positives, therefore, we are working on a better model, using machine learning techniques, which should be included in a future release. Such robust note segmentation is essential for obtaining good articulation descriptors, and we have already developed a legato index descriptor that relies on such robustness. We also plan to include, in a future version of Iracema, a vibrato descriptor, which was previously proposed and described in [17].

Acknowledgement

The development of this work has been supported by CAPES/Brazil (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior) and CNPq/Brazil (Conselho Nacional de Desenvolvimento Científico e Tecnológico).

References

[1] Eric Clarke. Empirical methods in the study of performance. In *Empirical musicology: Aims, methods, prospects*, pages 77–102. 2004.

[2] Alf Gabrielsson. Music performance research at the millennium. *Psychology of Music*, 31(3):221–272, 2003.

[3] C Palmer. Music performance. *Annual review of psychology*, 48:115–38, 1997.

[4] Campolina, Thiago A. M. and Mota, Davi A. and Loureiro, Mauricio A. Expan: a tool for musical expressiveness analysis. In *Proceedings of the 2nd International Conference of Students of Systematic Musicology*, pages 24–27, 2009.

[5] Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[6] Eric Jones, Travis Oliphant, Pearu Peterson, and Others. SciPy: Open source scientific tools for Python, 2001.

[7] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Publishing Company, third edit edition, 2013.

[8] P De La Cuadra. Efficient pitch detection techniques for interactive music. In *ICMC*, pages 403–406, 2001.

[9] Jong Wook Kim, Justin Salamon, Peter Li, and Juan Pablo Bello. CREPE: A Convolutional Representation for Pitch Estimation. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), 2018*, 2018.

[10] G. Peeters, B.L. Giordano, P. Susini, N. Misdariis, and S. McAdams. The timbre toolbox: extracting audio descriptors from musical signals. 130(5), 2011.

[11] Juan Pablo Bello, Laurent Daudet, Samer Abdallah, Chris Duxbury, Mike Davies, and Mark B. Sandler. A tutorial on onset detection in music signals. *IEEE Transactions on Speech and Audio Processing*, 13(5):1035–1046, 2005.

[12] Tae Hong Park. *Introduction to digital signal processing: Computer musically speaking*. World Scientific Publishing Co. Pte. Ltd., 2010.

[13] Simon Dixon. Onset Detection Revisited. In *9th International Conference on Digital Audio Effects*, pages 133–137, Montreal, Canada, 2006.

[14] Alexander Lerch. *An introduction to audio content analysis: Applications in signal processing and music informatics*. 2012.

[15] Aik Ming Toh, Roberto Togneri, and Sven Nordholm. Spectral entropy as speech features for speech recognition. *Computer Engineering*, (1):22–25, 2005.

[16] Adrian Sampson. Audioread. `https://github.com/beetbox/audioread`, 2011.

[17] Magalhães, Tairone N. and Mota, Davi A. and Neto, Aluizio B. O. and Loureiro, Mauricio A. Análise do vibrato e bending na guitarra elétrica. In *Anais do XV Simpósio Brasileiro de Computação Musical*, pages 36–47, 2015.