audiotools

Jörg Encke

Sep 01, 2023

CONTENTS:

audiotools is a python package designed to generate and anlyze acoustic stimuli for use in auditory research. It aims to provide an easy to use and intuitive interface.

CHAPTER

ONE

FLUENT INTERFACE

The main API of audiotools provides a fluent interface for generating and analyzing signals. In a fluent interface, methods are applied in-place and the object itself is returend which allowes methods to be stacked.

CHAPTER

TWO

THE SIGNAL CLASS

The audiotools.Signal class is used to work with signals in the time domain. Like all other classes that are used, the Signal class is inherited from the numpy.ndarray class and thus also inherits all its methods. It is also directly compatible with most of the packages in scientific stack such as scipy and matplotlib.

To create a empty signal, the class is called providing the number of channels, the duration of the stimulus and the sampling rate.

```
>>> sig = audio.Signal(n_channels=2, duration=1, fs=48000)
>>> print(sig.shape)
(48000, 2)
```

Basic properties of the signal such as the number of channels, samples or the duration are availible as properties:

```
>>> print(sig.n_channels, sig.duration, sig.n_samples, sig.fs)
2 1.0 48000 48000
```

Signals can have several dimensions:

```
>>> sig = audio.Signal(n_channels=(2, 3), duration=1, fs=48000)
>>> print(sig.shape)
(48000, 2, 3)
```

to directly index individual channels, the objects provides the *ch* property which also supports channel slicing

```
>>> sig = audio.Signal(n_channels=(2, 3), duration=1, fs=48000)
>>> slice = sig.ch[0, :]
>>> print(sig.shape, slice.shape)
(48000, 2, 3) (48000, 3)
```

Methods are allways applied to all channels.

```
>>> sig = audio.Signal(n_channels=2, duration=1, fs=48000)
>>> sig.add_noise()
>>> np.all(sig.ch[0] == sig.ch[1])
True
```

thus adds the same noise to both channels of the signal. The `ch` indexer if methods should be applied to one individual signal.

```
>>> sig = audio.Signal(n_channels=2, duration=1, fs=48000)
>>> sig.ch[0].add_noise()
>>> sig.ch[1].add_noise()
```

(continues on next page)

(continued from previous page)

```
>>> np.all(sig.ch[0] == sig.ch[1])
False
```

Using the ch indexer is equivalent to direclty indexing the signal

```
>>> sig = audio.Signal(n_channels=2, duration=1, fs=48000)
>>> sig.ch[0].add_tone(500)
>>> sig[:, 1].add_tone(500)
>>> np.all(sig.ch[0] == sig.ch[1])
True
```

CHAPTER

THREE

THE FREQUENCYDOMAINSIGNAL CLASS

Audiotools provides a simple mechanism of switching between time-domain and frequency-domain representation of a signal.

```
>>> sig = audio.Signal(2, 1, 48000).add_noise()
>>> print(type(sig))
<class 'audiotools.oaudio.signal.Signal'>
>>> fdomain_sig = sig.to_freqdomain()
>>> print(type(fdomain_sig))
<class 'audiotools.oaudio.freqdomain_signal.FrequencyDomainSignal'>
```

calling the method `audiotools.Signal.to_freqdomain()` returns a FrequencyDomainSignal object which contains the FFT transformed signal. It is important to note that the object does not directly contain the FFT transformed but that all frequency components where normalized by dividing them by the number of samples.

Like the Signal class, the FrequencyDomainSignal is inherits from `numpy.ndarray` an empty object can be created using an syntax identical to creating a Signal object

```
>>> sig = audio.FrequencyDomainSignal(n_channels=2, duration=1, fs=48000)
>>> print(sig.shape)
(48000, 2)
```

CHAPTER

FOUR

API REFERENCE

4.1 Fluent Interface API

4.1.1 Signals in the time domain (audiotools.Signal)

The *Signal* Class inherits from *numpy.ndarray* via the *audiotools.BaseSignal* class:

As a consequence, *numpy.ndarray* methods such as *x.min()*, *x.max()*, *x.sum()*, *x.var()* and others can also be used on auditools.Signal objects. For more informations check the numpy docs.

4.1.2 Signals in the frequency domain (audiotools.FrequencyDomainSignal)

The *FrequencyDomainSignal* class also inherits from *numpy.ndarray* via the *audiotools.BaseSignal* class:

4.2 Function Based API

The function based interface provides most of the functions that are availible as methods of the `Signal()` and also some that are not directly availible through the `Signal()` class.

4.2.1 audiotools.filter

Individual filter can either be applied by directly calling the respective filter functions such as `filter.gammatone()` or by using the unified interfaces for `filter.bandpass()`, `filter.lowpass()` and `filter.highpass()` filters. When using the unified interface, all additional arguments are passed to the respective filter functions.

Filterbanks are created using the `create_filterbank()` command

CHAPTER

FIVE

WHAT IS AUDIOTOOLS ?

audiotools is a python package designed to generate and analyze acoustic stimuli for use in auditory research. It aims to provide an easy to use and intuitive interface.

auditools provides the powerfull *Signal* class which extends the standard *numpy* array class with a fluent interface that provides methods and attributes often used in auditory signal processing.

The commands:

```
>>> import audiotools as audio
>>> sig = audio.Signal(n_channels=1, duration=1, fs=48000)
>>> sig.add_tone(500).set_dbspl(60).add_fade_window(10e-3, 'cos')
```

create a 1 second long signal with 1 channel at a sampling rate of 48kHz. A 500 Hz tone is then added to this signal, the level is set to 60dB SPL and a 10ms raised cosine fade-in and fade-out is added.

The Signal class also provides method to quickly switch between the frequency and time-domain representation of the same signal:

```
>>> sig.add_noise()
>>> f_sig = sig.to_freqdomain()
>>> f_sig[f_sig.freq.abs() > 1000] = 0
>>> sig = f_sig.to_timedomain()
```

first adds gaussian white noise to the signal and then sets all spectral components above 1kHz to zero.

All Signal classes are extensions of the standard *numpy* array, they can be used as drop-in replacements. As a consequence, the Signal class also inherits all methods of *numpy.ndarray*:

```
>>> import numpy as np
>>> sig = audio.Signal(n_channels=3, duration=1, fs=48000)
>>> sig.add_uncorr_noise(0.5)
>>> sig.var(axis=0)
Signal([1., 1., 1.])
```

More information and a detailed documentation of the methods and functions provided by audiotools can be found in the *api* and *introduction* sections.

5.1 Installation

5.1.1 Using pip

You can use pip to install audiotools

```
pip install audiotoolbox
```

5.1.2 From GitHub

Or directly from GitHub

1. Clone the repository: *git clone https://github.com/Jencke/audiotools.git*
2. Install the package: *pip install ./*
3. Optionally run the tests: *pytest*

5.2 Indices and tables

- genindex
- modindex
- search