aubio Documentation

Release 0.4.8

Paul Brossier

Nov 26, 2018

Contents

aubio is a collection of algorithms and tools to label and transform music and sounds. It scans or *listens* to audio signals and attempts to detect musical events. For instance, when a drum is hit, at which frequency is a note, or at what tempo is a rhythmic melody.

aubio features include segmenting a sound file before each of its attacks, performing pitch detection, tapping the beat and producing midi streams from live audio.

CHAPTER 1

Quick links

- *Python documentation*
- *Command line tools*
- *Developing with aubio*
- *Building aubio*

CHAPTER 2

Project pages

- Project homepage: https://aubio.org
- aubio on github: https://github.com/aubio/aubio
- aubio on pypi: https://pypi.python.org/pypi/aubio
- Doxygen documentation: https://aubio.org/doc/latest/
- Mailing lists: https://lists.aubio.org
- Travis Continuous integration page
- Appveyor Continuous integration page
- Landscape python code validation
- ReadTheDocs documentation

CHAPTER 3

Features

aubio provides several algorithms and routines, including:

- several onset detection methods
- different pitch detection methods
- tempo tracking and beat detection
- MFCC (mel-frequency cepstrum coefficients)
- FFT and phase vocoder
- up/down-sampling
- digital filters (low pass, high pass, and more)
- spectral filtering
- transient/steady-state separation
- sound file read and write access
- various mathematics utilities for music applications

The name aubio comes from *audio* with a typo: some errors are likely to be found in the results.

CHAPTER 4

Content

4.1 Installing aubio

aubio runs on Linux, Windows, macOS, iOS, Android, and probably a few others operating systems.

Aubio is available as a C library and as a python module.

4.1.1 Cheat sheet

- *get aubio latest source code*:

```
# official repo
git clone https://git.aubio.org/aubio/aubio
# mirror
git clone https://github.com/aubio/aubio
# latest release
wget https://aubio.org/pub/aubio-<version>.tar.gz
```

- *build aubio from source*:

```
# 1. simple
cd aubio
make

# 2. step by step
./scripts/get_waf.sh
./waf configure
./waf build
sudo ./waf install
```

- *install python-aubio from source*:

```
# from git
pip install git+https://git.aubio.org/aubio/aubio/
# mirror
pip install git+https://github.com/aubio/aubio/
# from latest release
pip install https://aubio.org/pub/aubio-latest.tar.bz2
# from pypi
pip install aubio
# from source directory
cd aubio
pip install -v .
```

- *install python-aubio from a pre-compiled binary*:

```
# conda [osx, linux, win]
conda install -c conda-forge aubio
# .deb (debian, ubuntu) [linux]
sudo apt-get install python3-aubio python-aubio aubio-tools
# brew [osx]
brew install aubio --with-python
```

- *get a pre-compiled version of libaubio*:

```
# .deb (linux) WARNING: old version
sudo apt-get install aubio-tools

# python module
./setup.py install
# using pip
pip install .
```

- *check the list of optional dependencies*:

```
# debian / ubuntu
dpkg -l libavcodec-dev libavutil-dev libavformat-dev \
        libswresample-dev libavresample-dev \
        libsamplerate-dev libsndfile-dev \
        txt2man doxygen
```

4.2 Downloading aubio

A number of distributions already include aubio. Check your favorite package management system, or have a look at the aubio download page for more options.

To use aubio in an android project, see *Android build*.

To compile aubio from source, read *Building aubio*.

4.2.1 Pre-compiled binaries

Pre-compiled binaries are available for macOS, iOS, and windows

To use aubio in a macOS or iOS application, see *Frameworks for Xcode*.

4.2.2 Debian/Ubuntu packages

For the latest Debian packages, see https://packages.debian.org/src:aubio.

For the latest Ubuntu packages, see http://packages.ubuntu.com/src:aubio.

For the latest version of the packages, see https://anonscm.debian.org/cgit/collab-maint/aubio.git/. Use `git-buildpackage` to build from the git repository. For instance:

```
$ git clone git://anonscm.debian.org/collab-maint/aubio.git
$ cd aubio
$ git buildpackage
```

4.3 Building aubio

Note: To download a prebuilt version of aubio, see *Downloading aubio*.

aubio uses waf to configure, compile, and test the source. A copy of waf is included in aubio tarball, so all you need is a terminal, a compiler, and a recent version of python installed.

Note: Make sure you have all the *Build options* you want before building.

4.3.1 Latest release

The **latest stable release** can be downloaded from https://aubio.org/download:

```
$ curl -O http://aubio.org/pub/aubio-<version>.tar.bz2
$ tar xf aubio-<version>.tar.bz2
$ cd aubio-<version>/
```

4.3.2 Git repository

The **latest git branch** can be obtained with:

```
$ git clone git://git.aubio.org/git/aubio
$ cd aubio/
```

The following command will fetch the correct waf version (not included in aubio's git):

```
$ ./scripts/get_waf.sh
```

Note: Windows users without Git Bash installed will want to use the following commands instead:

```
$ curl -fsS -o waf https://waf.io/waf-1.8.22
$ curl -fsS -o waf.bat https://raw.githubusercontent.com/waf-project/waf/master/utils/
↪waf.bat
```

4.3.3 Compiling

To compile the C library, examples programs, and tests, run:

```
$ ./waf configure
```

Check out the available options using `./waf configure --help`. Once you are done with configuration, you can start building:

```
$ ./waf build
```

To install the freshly built C library and tools, simply run the following command:

```
$ sudo ./waf install
```

Note: Windows users should simply run `waf`, without the leading `./`. For instance:

```
$ waf configure build
```

4.3.4 Running as a user

To use aubio without actually installing, for instance if you don't have root access to install libaubio on your system,

On Linux or macOS, sourcing the script `scripts/setenv_local.sh` should help:

```
$ source ./scripts/setenv_local.sh
```

This script sets `LD_LIBRARY_PATH`, for libaubio, and `PYTHONPATH` for the python module.

On Linux, you should be able to set `LD_LIBRARY_PATH` with:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$PWD/build/src
```

On Mac OS X, a copy or a symlink can be made in `~/lib`:

```
$ mkdir -p ~/lib
$ ln -sf $PWD/build/src/libaubio*.dylib ~/lib/
```

Note on Mac OS X systems older than El Capitan (10.11), the `DYLD_LIBRARY_PATH` variable can be set as follows:

```
$ export DYLD_LIBRARY_PATH=$DYLD_LIBRARY_PATH:$PWD/build/src
```

4.3.5 Cleaning

If you wish to uninstall the files installed by the `install` command, use `uninstall`:

```
$ sudo ./waf uninstall
```

To clean the source directory, use the `clean` command:

```
$ ./waf clean
```

To also forget the options previously passed to the last `./waf configure` invocation, use the `distclean` command:

```
$ ./waf distclean
```

4.3.6 Frameworks for Xcode

Binary frameworks are available and ready to use in your XCode project, for iOS and macOS.

1. Download and extract the corresponding `framework.zip` file from the *Download* page
2. Select **Build Phases** in your project setting and unfold **Link Binary with Libraries**
3. Add *AudioToolbox* and *Accelerate* system frameworks (or make sure they are listed)
4. Add `aubio.framework` from the unzipped `framework.zip`
5. Include the aubio header in your code:
   - in C/C++:

```
#include <aubio/aubio.h>
```

   - in Obj-C:

```
#import <aubio/aubio.h>
```

   - in Swift:

```
import aubio
```

4.3.7 Using aubio from swift

Once you have downloaded and installed *aubio.framework*, you sould be able to use aubio from C, Obj-C, and Swift source files.

Here is a short example showing how to read a sound file in swift:

```
import aubio

let path = Bundle.main.path(forResource: "example", ofType: "mp4")
if (path != nil) {
    let hop_size : uint_t = 512
    let a = new_fvec(hop_size)
    let b = new_aubio_source(path, 0, hop_size)
    var read: uint_t = 0
    var total_frames : uint_t = 0
    while (true) {
        aubio_source_do(b, a, &read)
        total_frames += read
        if (read < hop_size) { break }
    }
    print("read", total_frames, "frames at", aubio_source_get_samplerate(b),
↪"Hz")
    del_aubio_source(b)
    del_fvec(a)
} else {
```

(continues on next page)

(continued from previous page)

```
    print("could not find file")
}
```

4.3.8 Android build

To compile aubio for android, you will need to get the Android Native Development Toolkit (NDK), prepare a standalone toolchain, and tell waf to use the NDK toolchain. An example script to complete these tasks is available in `scripts/build_android`.

4.4 Build options

If built without any external dependencies aubio can be somewhat useful, for instance to read, process, and write simple wav files.

To support more media input formats and add more features to aubio, you can use one or all of the following *external libraries*.

You may also want to know more about the *other options* and the *platform notes*

The configure script will automatically for these extra libraries. To make sure the library or feature is used, pass the *–enable-flag* to waf. To disable this feature, use *–disable-feature*.

To find out more about the build commands, use the *–verbose* option.

4.4.1 External libraries

External libraries are checked for using `pkg-config`. Set the `PKG_CONFIG_PATH` environment variable if you have them installed in an unusual location.

Note: If `pkg-config` is not found in `PATH`, the configure step will succeed, but none of the external libraries will be used.

4.4.2 Media libraries

libav

libav.org, open source audio and video processing tools.

If all of the following libraries are found, they will be used to compile `aubio_source_avcodec`. so that `aubio_source` will be able to decode audio from all formats supported by libav.

- libavcodec
- libavformat
- libavutil
- libavresample

To enable this option, configure with `--enable-avcodec`. The build will then failed if the required libraries are not found. To disable this option, configure with `--disable-avcodec`

libsndfile

> libsndfile, a C library for reading and writing sampled sound files.

With libsndfile built in, `aubio_source_sndfile` will be built in and used by `aubio_source`.

To enable this option, configure with `--enable-sndfile`. The build will then fail if the required library is not found. To disable this option, configure with `--disable-sndfile`

libsamplerate

> libsamplerate, a sample rate converter for audio.

With libsamplerate built in, `aubio_source_sndfile` will support resampling, and `aubio_resample` will be fully functional.

To enable this option, configure with `--enable-samplerate`. The build will then fail if the required library is not found. To disable this option, configure with `--disable-samplerate`

4.4.3 Optimisation libraries

libfftw3

> FFTW, a C subroutine for computing the discrete Fourier transform

With libfftw3 built in, `aubio_fft` will use FFTW to compute Fast Fourier Transform (FFT), allowing aubio to compute FFT on length that are not a power of 2.

To enable this option, configure with `--enable-fftw3`. The build will then fail if the required library is not found. To disable this option, configure with `--disable-fftw3`

blas

On macOs/iOS, blas are made available through the Accelerate framework.

On Linux, they can be enabled with `--enable-blas`. On Debian (etch), *atlas*, *openblas*, and *libblas* have been successfully tested.

When enabled, `waf` will check for the current blas configuration by running `pkg-config --libs blas`. Depending of the library path returned by `pkg-config`, different headers will be searched for.

Note: On Debian systems, multiple versions of BLAS and LAPACK can be installed. To configure which libblas is being used:

```
$ sudo update-alternatives --config libblas.so
```

atlas

ATLAS BLAS APIs will be used the path returned by `pkg-config --libs blas` contains `atlas`.

Example:

```
$ pkg-config --libs blas
-L/usr/lib/atlas-base/atlas -lblas
$ ./waf configure --enable-atlas
[...]
Checking for 'blas'                         : yes
Checking for header atlas/cblas.h           : yes
```

openblas

OpenBlas libraries will be used when the output of `pkg-config --libs blas` contains 'openblas',

Example:

```
$ pkg-config --libs blas
-L/usr/lib/openblas-base -lblas
$ ./waf configure --enable-atlas
[...]
Checking for 'blas'                         : yes
Checking for header openblas/cblas.h        : yes
```

libblas

Netlib's libblas (LAPACK) will be used if no specific library path is specified by `pkg-config`

Example:

```
$ pkg-config --libs blas
-lblas
$ ./waf configure --enable-atlas
[...]
Checking for 'blas'                         : yes
Checking for header cblas.h                 : yes
```

4.4.4 Platform notes

On all platforms, you will need to have installed:

- a compiler (gcc, clang, msvc, . . .)
- python (any version >= 2.7, including 3.x)
- a terminal to run command lines in

Linux

The following *External libraries* will be used if found: libav, libsamplerate, libsndfile, *libfftw3*.

macOS

The following system frameworks will be used on Mac OS X systems:

- Accelerate to compute FFTs and other vectorized operations optimally.
- CoreAudio and AudioToolbox to decode audio from files and network streams.

Note: To build a fat binary for both `i386` and `x86_64`, use `./waf configure --enable-fat`.

The following *External libraries* will also be checked: libav, libsamplerate, libsndfile, *libfftw3*.

To build a fat binary on a darwin like system (macOS, tvOS, appleOS, ...) platforms, configure with `--enable-fat`.

Windows

To use a specific version of the compiler, `--msvc_version`. To build for a specific architecture, use `--msvc_target`. For instance, to build aubio for `x86` using `msvc 12.0`, use:

```
waf configure --msvc_version='msvc 12.0' --msvc_target='x86'
```

The following *External libraries* will be used if found: libav, libsamplerate, libsndfile, *libfftw3*.

iOS

The following system frameworks will be used on iOS and iOS Simulator.

- Accelerate to compute FFTs and other vectorized operations optimally.
- CoreAudio and AudioToolbox to decode audio from files and network streams.

To build aubio for iOS, configure with `--with-target-platform=ios`. For the iOS Simulator, use `--with-target-platform=iosimulator` instead.

By default, aubio is built with the following flags on iOS:

```
CFLAGS="-fembed-bitcode -arch arm64 -arch armv7 -arch armv7s -miphoneos-version-min=6.1"
```

and on iOS Simulator:

```
CFLAGS="-arch i386 -arch x86_64 -mios-simulator-version-min=6.1"
```

Set `CFLAGS` and `LINKFLAGS` to change these default values, or edit `wscript` directly.

4.4.5 Other options

Some additional options can be passed to the configure step. For the complete list of options, run:

```
$ ./waf --help
```

Here is an example of a custom command:

```
$ ./waf --verbose configure build install \
            --enable-avcodec --enable-wavread --disable-wavwrite \
            --enable-sndfile --enable-samplerate --enable-docs \
            --destdir $PWD/build/destdir --testcmd="echo %s" \
            --prefix=/opt --libdir=/opt/lib/multiarch \
            --manpagesdir=/opt/share/man  \
            uninstall clean distclean dist distcheck
```

Double precision

The datatype used to store real numbers in aubio is named *smpl_t*. By default, *smpl_t* is defined as *float*, a single-precision format (32-bit). Some algorithms require a floating point representation with a higher precision, for instance to prevent arithmetic underflow in recursive filters. In aubio, these special samples are named *lsmp_t* and defined as *double* by default (64-bit).

Sometimes it may be useful to compile aubio in *double-precision*, for instance to reproduce numerical results obtained with 64-bit routines. In this case, *smpl_t* will be defined as *double*.

The following table shows how *smpl_t* and *lsmp_t* are defined in single- and double-precision modes:

Table 1: Single and double-precision modes

	single	double
smpl_t	`float`	`double`
lsmp_t	`double`	`long double`

To compile aubio in double precision mode, configure with `--enable-double`.

To compile in single-precision mode (default), use `--disable-double` (or simply none of these two options).

Disabling the tests

In some case, for instance when cross-compiling, unit tests should not be run. Option `--notests` can be used for this purpose. The tests will not be executed, but the binaries will be compiled, ensuring that linking against libaubio works as expected.

Note: The `--notests` option should be passed to both `build` and `install` targets, otherwise waf will try to run them.

Edit wscript

Many of the options are gathered in the file *wscript*. a good starting point when looking for additional options.

4.4.6 Building the docs

If the following command line tools are found, the documentation will be built built:

- doxygen to build the *Doxygen documentation*.
- txt2man to build the *Command line tools*
- sphinx to build this document

These tools are searched for in the current `PATH` environment variable. By default, the documentation is built only if the tools are found.

To disable the documentation, configure with `--disable-docs`. To build with the documentation, configure with `--enable-docs`.

4.5 Installing aubio for Python

aubio is available as a package for Python 2.7 and Python 3. The aubio extension is written C using the Python/C and the Numpy/C APIs.

For general documentation on how to install Python packages, see Installing Packages.

4.5.1 Installing aubio with pip

aubio can be installed from PyPI using `pip`:

```
$ pip install aubio
```

See also Installing from PyPI for general documentation.

Note: aubio is currently a source only package, so you will need a compiler to install it from PyPI. See also *Installing aubio with conda* for pre-compiled binaries.

4.5.2 Installing aubio with conda

Conda packages are available through the conda-forge channel for Linux, macOS, and Windows:

```
$ conda config --add channels conda-forge
$ conda install -c conda-forge aubio
```

4.5.3 Double precision

This module can be compiled in double-precision mode, in which case the default type for floating-point samples will be 64-bit. The default is single precision mode (32-bit, recommended).

To build the aubio module with double precision, use the option *–enable-double* of the *build_ext* subcommand:

```
$ ./setup.py clean
$ ./setup.py build_ext --enable-double
$ pip install -v .
```

Note: If linking against *libaubio*, make sure the library was also compiled in *Double precision* mode.

4.5.4 Checking your installation

Once the python module is installed, its version can be checked with:

```
$ python -c "import aubio; print(aubio.version, aubio.float_type)"
```

The command line *aubio* is also installed:

```
$ aubio -h
```

4.5.5 Python tests

A number of Python tests are provided in the python tests. To run them, install nose2 and run the script `python/tests/run_all_tests`:

```
$ pip install nose2
$ ./python/tests/run_all_tests
```

4.6 Python documentation

This module provides a number of classes and functions for the analysis of music and audio signals.

4.6.1 Contents

Data-types

This section contains the documentation for *`float_type`*, *`fvec`*, and *`cvec`*.

`aubio.`**`float_type`**

A string constant describing the floating-point representation used in *`fvec`*, *`cvec`*, and elsewhere in this module.

Defaults to *"float32"*.

If *aubio* was built specifically with the option *–enable-double*, this string will be defined to *"float64"*. See *Double precision* in *Installing aubio for Python* for more details on building aubio in double precision mode.

Examples

```
>>> aubio.float_type
'float32'
>>> numpy.zeros(10).dtype
'float64'
>>> aubio.fvec(10).dtype
'float32'
>>> np.arange(10, dtype=aubio.float_type).dtype
'float32'
```

`class` `aubio.`**`fvec`**(*input_arg=1024*)

A vector holding float samples.

If *input_arg* is an *int*, a 1-dimensional vector of length *input_arg* will be created and filled with zeros. Otherwise, if *input_arg* is an *array_like* object, it will be converted to a 1-dimensional vector of type *`float_type`*.

Parameters **`input_arg`** (*int* or *array_like*) – Can be a positive integer, or any object that can be converted to a numpy array with `numpy.array()`.

Examples

```
>>> aubio.fvec(10)
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
>>> aubio.fvec([0,1,2])
array([0., 1., 2.], dtype=float32)
>>> a = np.arange(10); type(a), type(aubio.fvec(a))
(<class 'numpy.ndarray'>, <class 'numpy.ndarray'>)
>>> a.dtype, aubio.fvec(a).dtype
(dtype('int64'), dtype('float32'))
```

Notes

In the Python world, *fvec* is simply a subclass of `numpy.ndarray`. In practice, any 1-dimensional *numpy.ndarray* of *dtype* `float_type` may be passed to methods accepting *fvec* as parameter. For instance, *sink()* or *pvoc()*.

See also:

`cvec` a container holding spectral data

`numpy.ndarray` parent class of `fvec`

`numpy.zeros` create a numpy array filled with zeros

`numpy.array` create a numpy array from an existing object

class aubio.**cvec**(*size*)

A container holding spectral data.

Create one *cvec* to store the spectral information of a window of *size* points. The data will be stored in two vectors, `phas` and `norm`, each of shape (`length`,), with *length = size // 2 + 1*.

Parameters **size** (*int*) – Size of spectrum to create.

Examples

```
>>> c = aubio.cvec(1024)
>>> c
aubio cvec of 513 elements
>>> c.length
513
>>> c.norm.dtype, c.phas.dtype
(dtype('float32'), dtype('float32'))
>>> c.norm.shape, c.phas.shape
((513,), (513,))
```

See also:

`fvec`, `fft`, `pvoc`

length

Length of *norm* and *phas* vectors.

Type int

norm

Vector of shape *(length,)* containing the magnitude.

Type numpy.ndarray

phas
Vector of shape *(length,)* containing the phase.

Type numpy.ndarray

Input/Output

This section contains the documentation for two classes: `source`, to read audio samples from files, and `sink`, to write audio samples to disk.

class aubio.**source**(*path*, *samplerate=0*, *hop_size=512*, *channels=0*)
Read audio samples from a media file.

source open the file specified in *path* and creates a callable returning *hop_size* new audio samples at each invocation.

If *samplerate=0* (default), the original sampling rate of *path* will be used. Otherwise, the output audio samples will be resampled at the desired sampling-rate.

If *channels=0* (default), the original number of channels in *path* will be used. Otherwise, the output audio samples will be down-mixed or up-mixed to the desired number of channels.

If *path* is a URL, a remote connection will be attempted to open the resource and stream data from it.

The parameter *hop_size* determines how many samples should be read at each consecutive calls.

Parameters

- **path** (*str*) – pathname (or URL) of the file to be opened for reading
- **samplerate** (*int, optional*) – sampling rate of the file
- **hop_size** (*int, optional*) – number of samples to be read per iteration
- **channels** (*int, optional*) – number of channels of the file

Examples

By default, when only *path* is given, the file will be opened with its original sampling rate and channel:

```
>>> src = aubio.source('stereo.wav')
>>> src.uri, src.samplerate, src.channels, src.duration
('stereo.wav', 48000, 2, 86833)
```

A typical loop to read all samples from a local file could look like this:

```
>>> src = aubio.source('stereo.wav')
>>> total_read = 0
>>> while True:
...     samples, read = src()
...     # do something with samples
...     total_read += read
...     if read < src.hop_size:
...         break
...
```

In a more Pythonic way, it can also look like this:

```
>>> total_read = 0
>>> with aubio.source('stereo.wav') as src:
...     for frames in src:
...         total_read += samples.shape[-1]
...
```

Basic interface

source is a **callable**; its `__call__()` method returns a tuple containing:

- a vector of shape *(hop_size,)*, filled with the *read* next samples available, zero-padded if *read* < *hop_size*
- *read*, an integer indicating the number of samples read

To read the first *hop_size* samples from the source, simply call the instance itself, with no argument:

```
>>> src = aubio.source('song.ogg')
>>> samples, read = src()
>>> samples.shape, read, src.hop_size
((512,), 512, 512)
```

The first call returned the slice of samples *[0 : hop_size]*. The next call will return samples *[hop_size: 2*hop_size]*.

After several invocations of `__call__()`, when reaching the end of the opened stream, *read* might become less than *hop_size*:

```
>>> samples, read = src()
>>> samples.shape, read
((512,), 354)
```

The end of the vector *samples* is filled with zeros.

After the end of the stream, *read* will be *0* since no more samples are available:

```
>>> samples, read = src()
>>> samples.shape, read
((512,), 0)
```

Note: when the source has more than one channels, they are be down-mixed to mono when invoking `__call__()`. To read from each individual channel, see `__next__()`.

`for` statements

The *source* objects are **iterables**. This allows using them directly in a `for` loop, which calls `__next__()` until the end of the stream is reached:

```
>>> src = aubio.source('stereo.wav')
>>> for frames in src:
>>>     print (frames.shape)
...
(2, 512)
(2, 512)
(2, 230)
```

Note: When *next(self)* is called on a source with multiple channels, an array of shape *(channels, read)* is returned, unlike with `__call__()` which always returns the down-mixed channels.

If the file is opened with a single channel, *next(self)* returns an array of shape *(read,)*:

```
>>> src = aubio.source('stereo.wav', channels=1)
>>> next(src).shape
(512,)
```

`with` statements

The *source* objects are **context managers**, which allows using them in `with` statements:

```
>>> with aubio.source('audiotrack.wav') as source:
...     n_frames=0
...     for samples in source:
...         n_frames += len(samples)
...     print('read', n_frames, 'samples in', samples.shape[0], 'channels',
...         'from file "%%s"' %% source.uri)
...
read 239334 samples in 2 channels from file "audiotrack.wav"
```

The file will be closed before exiting the statement.

See also the methods implementing the context manager, `__enter__()` and `__exit__()`.

Seeking and closing

At any time, `seek()` can be used to move to any position in the file. For instance, to rewind to the start of the stream:

```
>>> src.seek(0)
```

The opened file will be automatically closed when the object falls out of scope and is scheduled for garbage collection.

In some cases, it is useful to manually `close()` a given source, for instance to limit the number of simultaneously opened files:

```
>>> src.close()
```

Input formats

Depending on how aubio was compiled, `source` may or may not open certain **files format**. Below are some examples that assume support for compressed files and remote urls was compiled in:

- open a local file using its original sampling rate and channels, and with the default hop size:

```
>>> s = aubio.source('sample.wav')
>>> s.uri, s.samplerate, s.channels, s.hop_size
('sample.wav', 44100, 2, 512)
```

- open a local compressed audio file, resampling to 32000Hz if needed:

```
>>> s = aubio.source('song.mp3', samplerate=32000)
>>> s.uri, s.samplerate, s.channels, s.hop_size
('song.mp3', 32000, 2, 512)
```

- open a local video file, down-mixing and resampling it to 16kHz:

```
>>> s = aubio.source('movie.mp4', samplerate=16000, channels=1)
>>> s.uri, s.samplerate, s.channels, s.hop_size
('movie.mp4', 16000, 1, 512)
```

- open a remote resource, with hop_size = 1024:

```
>>> s = aubio.source('https://aubio.org/drum.ogg', hop_size=1024)
>>> s.uri, s.samplerate, s.channels, s.hop_size
('https://aubio.org/drum.ogg', 48000, 2, 1024)
```

See also:

`sink` write audio samples to a file.

`__call__`()

Read at most *hop_size* new samples from self, return them in a tuple with the number of samples actually read.

The returned tuple contains:

- a vector of shape *(hop_size,)*, filled with the *read* next samples available, zero-padded if *read* < *hop_size*
- *read*, an integer indicating the number of samples read

If opened with more than one channel, the frames will be down-mixed to produce the new samples.

Returns A tuple of one array of samples and one integer.

Return type (array, int)

See also:

`__next__()`

Example

```
>>> src = aubio.source('stereo.wav')
>>> while True:
...     samples, read = src()
...     if read < src.hop_size:
...         break
```

`__next__`()

Read at most *hop_size* new frames from self, return them in an array.

If source was opened with one channel, next(self) returns an array of shape *(read,)*, where *read* is the actual number of frames read (*0 <= read <= hop_size*).

If *source* was opened with more then one channel, the returned arrays will be of shape *(channels, read)*, where *read* is the actual number of frames read (*0 <= read <= hop_size*).

Returns A tuple of one array of frames and one integer.

Return type (array, int)

See also:

__call__()

Example

```
>>> for frames in aubio.source('song.flac')
...     print(samples.shape)
```

__iter__()
Implement iter(self).

See also:

__next__()

__enter__()
Implement context manager interface. The file will be opened upon entering the context. See *with* statement.

Example

```
>>> with aubio.source('loop.ogg') as src:
...     src.uri, src.samplerate, src.channels
```

__exit__()
Implement context manager interface. The file will be closed before exiting the context. See *with* statement.

See also:

__enter__()

close()
Close this source now.

Note: Closing twice a source will **not** raise any exception.

do()
Read vector of audio samples.

If the audio stream in the source has more than one channel, the channels will be down-mixed.

Returns

- **samples** (*numpy.ndarray*) – *fvec* of size *hop_size* containing the new samples.
- **read** (*int*) – Number of samples read from the source, equals to *hop_size* before the end-of-file is reached, less when it is reached, and *0* after.

See also:

do_multi()

Examples

```
>>> src = aubio.source('sample.wav', hop_size=1024)
>>> src.do()
(array([-0.00123001, -0.00036685,  0.00097106, ..., -0.2031033 ,
       -0.2025854 , -0.20221856], dtype=float32), 1024)
```

do_multi()

Read multiple channels of audio samples.

If the source was opened with the same number of channels found in the stream, each channel will be read individually.

If the source was opened with less channels than the number of channels in the stream, only the first channels will be read.

If the source was opened with more channels than the number of channel in the original stream, the first channels will be duplicated on the additional output channel.

Returns

- **samples** (*numpy.ndarray*) – NumPy array of shape *(hop_size, channels)* containing the new audio samples.
- **read** (*int*) – Number of samples read from the source, equals to *hop_size* before the end-of-file is reached, less when it is reached, and *0* after.

See also:

do()

Examples

```
>>> src = aubio.source('sample.wav')
>>> src.do_multi()
(array([[ 0.00668335,  0.0067749 ,  0.00714111, ..., -0.05737305,
         -0.05856323, -0.06018066],
       [-0.00842285, -0.0072937 , -0.00576782, ..., -0.09405518,
         -0.09558105, -0.09725952]], dtype=float32), 512)
```

get_channels()

Get number of channels in source.

Returns Number of channels.

Return type int

get_samplerate()

Get sampling rate of source.

Returns Sampling rate, in Hz.

Return type int

seek(*position*)

Seek to position in file.

If the source was not opened with its original sampling-rate, *position* corresponds to the position in the re-sampled file.

Parameters **position** (*str*) – position to seek to, in samples

channels
number of channels

Type int (read-only)

duration
total number of frames in the source

Can be estimated, for instance if the opened stream is a compressed media or a remote resource.

Example

```
>>> n = 0
>>> src = aubio.source('track1.mp3')
>>> for samples in src:
...     n += samples.shape[-1]
...
>>> n, src.duration
(9638784, 9616561)
```

Type int (read-only)

hop_size
number of samples read per iteration

Type int (read-only)

samplerate
sampling rate

Type int (read-only)

uri
pathname or URL

Type str (read-only)

class aubio.**sink**(*path*, *samplerate=44100*, *channels=1*)
Write audio samples to file.

Parameters

- **path** (*str*) – Pathname of the file to be opened for writing.
- **samplerate** (*int*) – Sampling rate of the file, in Hz.
- **channels** (*int*) – Number of channels to create the file with.

Examples

Create a new sink at 44100Hz, mono:

```
>>> snk = aubio.sink('out.wav')
```

Create a new sink at 32000Hz, stereo, write 100 samples into it:

```
>>> snk = aubio.sink('out.wav', samplerate=16000, channels=3)
>>> snk(aubio.fvec(100), 100)
```

Open a new sink at 48000Hz, stereo, write *1234* samples into it:

```
>>> with aubio.sink('out.wav', samplerate=48000, channels=2) as src:
...     snk(aubio.fvec(1024), 1024)
...     snk(aubio.fvec(210), 210)
...
```

See also:

`source` read audio samples from a file.

`__call__`(*vec*, *length*)
Write *length* samples from *vec*.

Parameters

- **vec** (*array*) – input vector to write from
- **length** (*int*) – number of samples to write

Example

```
>>> with aubio.sink('foo.wav') as snk:
...     snk(aubio.fvec(1025), 1025)
```

`close`()
Close this sink now.

By default, the sink will be closed before being deleted. Explicitely closing a sink can be useful to control the number of files simultaneously opened.

`do`(*vec*, *write*)
Write a single channel vector to sink.

Parameters

- **vec** (`fvec`) – input vector *(n,)* where *n >= 0*.
- **write** (*int*) – Number of samples to write.

`do_multi`(*mat*, *write*)
Write a matrix containing vectors from multiple channels to sink.

Parameters

- **mat** (*numpy.ndarray*) – input matrix of shape *(channels, n)*, where *n >= 0*.
- **write** (*int*) – Number of frames to write.

`channels`
Number of channels with which the sink was created.

Type int (read-only)

`samplerate`
Samplerate at which the sink was created.

Type int (read-only)

`uri`
Path at which the sink was created.

Type str (read-only)

Utilities

This section documents various helper functions included in the aubio library.

Note name conversion

`aubio.note2midi(note)`

Convert note name to midi note number.

Input string *note* should be composed of one note root and one octave, with optionally one modifier in between.

List of valid components:

- note roots: *C*, *D*, *E*, *F*, *G*, *A*, *B*,
- modifiers: *b*, *#*, as well as unicode characters , , , and ,
- octave numbers: *-1* -> *11*.

Parameters **note** (*str*) – note name

Returns corresponding midi note number

Return type int

Examples

```
>>> aubio.note2midi('C#4')
61
>>> aubio.note2midi('B5')
82
```

Raises

- `TypeError` – If *note* was not a string.
- `ValueError` – If an error was found while converting *note*.

See also:

midi2note(), *freqtomidi()*, *miditofreq()*

`aubio.midi2note(midi)`

Convert midi note number to note name.

Parameters **midi** (*int [0, 128]*) – input midi note number

Returns note name

Return type str

Examples

```
>>> aubio.midi2note(70)
'A#4'
>>> aubio.midi2note(59)
'B3'
```

Raises

- `TypeError` – If *midi* was not an integer.
- `ValueError` – If *midi* is out of the range *[0, 128]*.

See also:

note2midi(), *miditofreq()*, *freqtomidi()*

`aubio.`**`freq2note`**(*freq*)

Convert frequency in Hz to nearest note name.

Parameters **freq** (*float [0, 23000[*) – input frequency, in Hz

Returns name of the nearest note

Return type str

Example

```
>>> aubio.freq2note(440)
'A4'
>>> aubio.freq2note(220.1)
'A3'
```

`aubio.`**`note2freq`**(*note*)

Convert note name to corresponding frequency, in Hz.

Parameters **note** (*str*) – input note name

Returns **freq** – frequency, in Hz

Return type float [0, 23000[

Example

```
>>> aubio.note2freq('A4')
440
>>> aubio.note2freq('A3')
220.1
```

Frequency conversion

`aubio.`**`freqtomidi`**(*x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True*[, *signature, extobj*])

Convert frequency *[0; 23000[* to midi *[0; 128[*.

Parameters **x** (*numpy.ndarray*) – Array of frequencies, in Hz.

Returns Converted frequencies, in midi note.

Return type numpy.ndarray

`aubio.`**`miditofreq`**(*x, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True*[, *signature, extobj*])

Convert midi *[0; 128[* to frequency *[0, 23000]*.

Parameters **x** (*numpy.ndarray*) – Array of frequencies, in midi note.

Returns Converted frequencies, in Hz

Return type numpy.ndarray

`aubio.meltohz(m, htk=False)`

Convert a scalar from mel scale to frequency.

Parameters

- **m** (*float*) – input mel
- **htk** (*bool*) – if *True*, use Htk mel scale instead of Slaney.

Returns output frequency, in Hz

Return type float

See also:

hztomel()

`aubio.hztomel(f, htk=False)`

Convert a scalar from frequency to mel scale.

Parameters

- **m** (*float*) – input frequency, in Hz
- **htk** (*bool*) – if *True*, use Htk mel scale instead of Slaney.

Returns output mel

Return type float

See also:

meltohz()

`aubio.bintomidi(fftbin, samplerate, fftsize)`

Convert FFT bin to frequency in midi note, given the sampling rate and the size of the FFT.

Parameters

- **fftbin** (*float*) – input frequency bin
- **samplerate** (*float*) – sampling rate of the signal
- **fftsize** (*float*) – size of the FFT

Returns Frequency converted to midi note.

Return type float

Example

```
>>> aubio.bintomidi(10, 44100, 1024)
68.62871551513672
```

`aubio.miditobin(midi, samplerate, fftsize)`

Convert frequency in midi note to FFT bin, given the sampling rate and the size of the FFT.

Parameters

- **midi** (*float*) – input frequency, in midi note
- **samplerate** (*float*) – sampling rate of the signal

- **fftsize** (*float*) – size of the FFT

Returns Frequency converted to FFT bin.

Return type float

Examples

```
>>> aubio.miditobin(69, 44100, 1024)
10.216779708862305
>>> aubio.miditobin(75.08, 32000, 512)
10.002175331115723
```

`aubio.`**bintofreq**(*fftbin*, *samplerate*, *fftsize*)

Convert FFT bin to frequency in Hz, given the sampling rate and the size of the FFT.

Parameters

- **fftbin** (*float*) – input frequency bin
- **samplerate** (*float*) – sampling rate of the signal
- **fftsize** (*float*) – size of the FFT

Returns Frequency converted to Hz.

Return type float

Example

```
>>> aubio.bintofreq(10, 44100, 1024)
430.6640625
```

`aubio.`**freqtobin**(*freq*, *samplerate*, *fftsize*)

Convert frequency in Hz to FFT bin, given the sampling rate and the size of the FFT.

Parameters

- **midi** (*float*) – input frequency, in midi note
- **samplerate** (*float*) – sampling rate of the signal
- **fftsize** (*float*) – size of the FFT

Returns Frequency converted to FFT bin.

Return type float

Examples

```
>>> aubio.freqtobin(440, 44100, 1024)
10.216779708862305
```

Audio file slicing

aubio.**slice_source_at_stamps**(*source_file*, *timestamps*, *timestamps_end=None*, *output_dir=None*, *samplerate=0*, *hopsize=256*, *create_first=False*)

Slice a sound file at given timestamps.

This function reads *source_file* and creates slices, new smaller files each starting at *t* in *timestamps*, a list of integer corresponding to time locations in *source_file*, in samples.

If *timestamps_end* is unspecified, the slices will end at *timestamps_end[n] = timestamps[n+1]-1*, or the end of file. Otherwise, *timestamps_end* should be a list with the same length as *timestamps* containing the locations of the end of each slice.

If *output_dir* is unspecified, the new slices will be written in the current directory. If *output_dir* is a string, new slices will be written in *output_dir*, after creating the directory if required.

The default *samplerate* is 0, meaning the original sampling rate of *source_file* will be used. When using a sampling rate different to the one of the original files, *timestamps* and *timestamps_end* should be expressed in the re-sampled signal.

The *hopsize* parameter simply tells `source` to use this hopsize and does not change the output slices.

If *create_first* is True and *timestamps* does not start with *0*, the first slice from *0* to *timestamps[0] - 1* will be automatically added.

Parameters

- **source_file** (*str*) – path of the resource to slice
- **timestamps** (list of int) – time stamps at which to slice, in samples
- **timestamps_end** (list of int (optional)) – time stamps at which to end the slices
- **output_dir** (*str (optional)*) – output directory to write the slices to
- **samplerate** (*int (optional)*) – samplerate to read the file at
- **hopsize** (*int (optional)*) – number of samples read from source per iteration
- **create_first** (*bool (optional)*) – always create the slice at the start of the file

Examples

Create two slices: the first slice starts at the beginning of the input file *loop.wav* and lasts exactly one second, starting at sample *0* and ending at sample *44099*; the second slice starts at sample *44100* and lasts until the end of the input file:

```
>>> aubio.slice_source_at_stamps('loop.wav', [0, 44100])
```

Create one slice, from 1 second to 2 seconds:

```
>>> aubio.slice_source_at_stamps('loop.wav', [44100], [44100 * 2 - 1])
```

Notes

Slices may be overlapping. If *timestamps_end* is *1* element shorter than *timestamps*, the last slice will end at the end of the file.

Windowing

`aubio.window(window_type, size)`

Create a window of length *size*. *window_type* should be one of the following:

- *default* (same as *hanningz*).
- *ones*
- *rectangle*
- *hamming*
- *hanning*
- *hanningz*[1]
- *blackman*
- *blackman_harris*
- *gaussian*
- *welch*
- *parzen*

Parameters

- **window_type** (*str*) – Type of window.
- **size** (*int*) – Length of window.

Returns Array of shape *(length,)* containing the new window.

Return type *fvec*

See also:

`pvoc()`, `fft()`

Examples

Compute a zero-phase Hann window on *1024* points:

```
>>> aubio.window('hanningz', 1024)
array([  0.00000000e+00,   9.41753387e-06,   3.76403332e-05, ...,
         8.46982002e-05,   3.76403332e-05,   9.41753387e-06], dtype=float32)
```

Plot different window types with matplotlib:

```
>>> import matplotlib.pyplot as plt
>>> modes = ['default', 'ones', 'rectangle', 'hamming', 'hanning',
...          'hanningz', 'blackman', 'blackman_harris', 'gaussian',
...          'welch', 'parzen']; n = 2048
>>> for m in modes: plt.plot(aubio.window(m, n), label=m)
...
>>> plt.legend(); plt.show()
```

[1] Amalia de Götzen, Nicolas Bernardini, and Daniel Arfib. Traditional (?) implementations of a phase vocoder: the tricks of the trade. In *Proceedings of the International Conference on Digital Audio Effects* (DAFx-00), pages 37–44, University of Verona, Italy, 2000. (online version).

Note: The following examples contain the equivalent source code to compute each type of window with NumPy:

```
>>> n = 1024; x = np.arange(n, dtype=aubio.float_type)
>>> ones = np.ones(n).astype(aubio.float_type)
>>> rectangle = 0.5 * ones
>>> hanning = 0.5 - 0.5 * np.cos(2 * np.pi * x / n)
>>> hanningz = 0.5 * (1 - np.cos(2 * np.pi * x / n))
>>> hamming = 0.54 - 0.46 * np.cos(2.*np.pi * x / (n - 1))
>>> blackman = 0.42 \
...          - 0.50 * np.cos(2 * np.pi * x / (n - 1)) \
...          + 0.08 * np.cos(4 * np.pi * x / (n - 1))
>>> blackman_harris = 0.35875 \
...        - 0.48829 * np.cos(2 * np.pi * x / (n - 1)) \
...        + 0.14128 * np.cos(4 * np.pi * x / (n - 1)) \
...        + 0.01168 * np.cos(6 * np.pi * x / (n - 1))
>>> gaussian = np.exp( - 0.5 * ((x - 0.5 * (n - 1)) \
...                           / (0.25 * (n - 1)) )**2 )
>>> welch = 1 - ((2 * x - n) / (n + 1))**2
>>> parzen = 1 - np.abs((2 * x - n) / (n + 1))
>>> default = hanningz
```

References

Audio level detection

aubio.**level_lin**(*x*)

Compute sound pressure level of *x*, on a linear scale.

Parameters **x** (fvec) – input vector

Returns Linear level of *x*.

Return type float

Example

```
>>> aubio.level_lin(aubio.fvec(numpy.ones(1024)))
1.0
```

Note: Computed as the average of the squared amplitudes:

$$L = \frac{\sum_{n=0}^{N-1} {x_n}^2}{N}$$

See also:

db_spl(), *silence_detection()*, *level_detection()*

aubio.**db_spl**(*x*)

Compute Sound Pressure Level (SPL) of *x*, in dB.

Parameters **x** (fvec) – input vector

Returns Level of *x*, in dB SPL.

Return type float

Example

```
>>> aubio.db_spl(aubio.fvec(np.ones(1024)))
1.0
>>> aubio.db_spl(0.7*aubio.fvec(np.ones(32)))
-3.098040819168091
```

Note: Computed as *log10* of `level_lin()`:

$$SPL_{dB} = log10\frac{\sum_{n=0}^{N-1} {x_n}^2}{N}$$

This quantity is often incorrectly called 'loudness'.

See also:

`level_lin()`, `silence_detection()`, `level_detection()`

`aubio.`**`silence_detection`**`(`*vec*, *level*`)`

Check if level of *vec*, in dB SPL, is under a given threshold.

Parameters

- **`vec`** (`fvec`) – input vector
- **`level`** (*`float`*) – level threshold, in dB SPL

Returns *1* if level of *vec*, in dB SPL, is under *level*, *0* otherwise.

Return type int

Examples

```
>>> aubio.silence_detection(aubio.fvec(32), -100.)
1
>>> aubio.silence_detection(aubio.fvec(np.ones(32)), 0.)
0
```

See also:

`level_detection()`, `db_spl()`, `level_lin()`

`aubio.`**`level_detection`**`(`*vec*, *level*`)`

Check if *vec* is above threshold *level*, in dB SPL.

Parameters

- **`vec`** (`fvec`) – input vector
- **`level`** (*`float`*) – level threshold, in dB SPL

Returns *1.0* if level of *vec* in dB SPL is under *level*, *db_spl(vec)* otherwise.

Return type float

Example

```
>>> aubio.level_detection(0.7*aubio.fvec(np.ones(1024)), -3.)
1.0
>>> aubio.level_detection(0.7*aubio.fvec(np.ones(1024)), -4.)
-3.0980708599090576
```

See also:

silence_detection(), *db_spl()*, *level_lin()*

Vector utilities

`aubio.`**`alpha_norm`**`(`*vec*, *alpha*`)`

Compute *alpha* normalisation factor of vector *vec*.

Parameters

- **vec** (fvec) – input vector
- **alpha** (*float*) – norm factor

Returns p-norm of the input vector, where *p=alpha*

Return type float

Example

```
>>> a = aubio.fvec(np.arange(10)); alpha = 2
>>> aubio.alpha_norm(a, alpha), (sum(a**alpha)/len(a))**(1./alpha)
(5.338539123535156, 5.338539126015656)
```

Note: Computed as:

$$l_\alpha = \| \frac{\sum_{n=0}^{N-1} {x_n}^\alpha}{N} \|^{1/\alpha}$$

`aubio.`**`zero_crossing_rate`**`(`*vec*`)`

Compute zero-crossing rate of *vec*.

Parameters **vec** (fvec) – input vector

Returns Zero-crossing rate.

Return type float

Example

```
>>> a = np.linspace(-1., 1., 1000, dtype=aubio.float_type)
>>> aubio.zero_crossing_rate(a), 1/1000
(0.0010000000474974513, 0.001)
```

aubio.**min_removal**(*vec*)

Remove the minimum value of a vector to each of its element.

Modifies the input vector in-place and returns a reference to it.

Parameters **vec** (fvec) – input vector

Returns modified input vector

Return type *fvec*

Example

```
>>> aubio.min_removal(aubio.fvec(np.arange(1,4)))
array([0., 1., 2.], dtype=float32)
```

aubio.**shift**(*vec*)

Swap left and right partitions of a vector, in-place.

Parameters **vec** (fvec) – input vector to shift

Returns The swapped vector.

Return type *fvec*

Notes

The input vector is also modified.

For a vector of length N, the partition is split at index N - N//2.

Example

```
>>> aubio.shift(aubio.fvec(np.arange(3)))
array([2., 0., 1.], dtype=float32)
```

See also:

ishift()

aubio.**ishift**(*vec*)

Swap right and left partitions of a vector, in-place.

Parameters **vec** (fvec) – input vector to shift

Returns The swapped vector.

Return type *fvec*

Notes

The input vector is also modified.

Unlike with *shift()*, the partition is split at index N//2.

Example

```
>>> aubio.ishift(aubio.fvec(np.arange(3)))
array([1., 2., 0.], dtype=float32)
```

See also:

shift()

aubio.**unwrap2pi**(*x*, */*, *out=None*, ***, *where=True*, *casting='same_kind'*, *order='K'*, *dtype=None*, *subok=True*[, *signature*, *extobj*])

Map angle to unit circle $[-\pi, \pi[$.

Parameters **x** (*numpy.ndarray*) – input array

Returns values clamped to the unit circle $[-\pi, \pi[$

Return type numpy.ndarray

Examples

Below is a short selection of examples using the aubio module.

Read a sound file

Here is a simple script, `demo_source_simple.py` that reads all the samples from a media file using *source*:

```
#! /usr/bin/env python

"""A simple example using aubio.source."""

import sys
import aubio

samplerate = 0  # use original source samplerate
hop_size = 256  # number of frames to read in one block
src = aubio.source(sys.argv[1], samplerate, hop_size)
total_frames = 0

while True:
    samples, read = src()  # read hop_size new samples from source
    total_frames += read   # increment total number of frames
    if read < hop_size:    # end of file reached
        break

fmt_string = "read {:d} frames at {:d}Hz from {:s}"
print(fmt_string.format(total_frames, src.samplerate, src.uri))
```

Filter a sound file

Here is another example, `demo_filter.py`, which applies a filter to a sound file and writes the filtered signal in another file:

- read audio samples from a file with *source*
- filter them using an A-weighting filter using `digital_filter`

- write the filtered samples to a new file with *sink*.

```
#! /usr/bin/env python

import sys
import os.path
import aubio


def apply_filter(path, target):
    # open input file, get its samplerate
    s = aubio.source(path)
    samplerate = s.samplerate

    # create an A-weighting filter
    f = aubio.digital_filter(7)
    f.set_a_weighting(samplerate)

    # create output file
    o = aubio.sink(target, samplerate)

    total_frames = 0
    while True:
        # read from source
        samples, read = s()
        # filter samples
        filtered_samples = f(samples)
        # write to sink
        o(filtered_samples, read)
        # count frames read
        total_frames += read
        # end of file reached
        if read < s.hop_size:
            break

    # print some info
    duration = total_frames / float(samplerate)
    input_str = "input: {:s} ({:.2f} s, {:d} Hz)"
    output_str = "output: {:s}, A-weighting filtered ({:d} frames total)"
    print(input_str.format(s.uri, duration, samplerate))
    print(output_str.format(o.uri, total_frames))

if __name__ == '__main__':
    usage = "{:s} <input_file> [output_file]".format(sys.argv[0])
    if not 1 < len(sys.argv) < 4:
        print(usage)
        sys.exit(1)
    if len(sys.argv) < 3:
        input_path = sys.argv[1]
        basename = os.path.splitext(os.path.basename(input_path))[0] + ".wav"
        output_path = "filtered_" + basename
    else:
        input_path, output_path = sys.argv[1:]
    # run function
    apply_filter(input_path, output_path)
```

More examples

For more examples showing how to use other components of the module, see the python demos folder.

4.6.2 Introduction

This document provides a reference guide. For documentation on how to install aubio, see *Installing aubio for Python*.

Examples included in this guide and within the code are written assuming both *aubio* and numpy have been imported:

```
>>> import aubio
>>> import numpy as np
```

Changed in 0.4.8 : Prior to this version, almost no documentation was provided with the python module. This version adds documentation for some classes, including *fvec*, *cvec*, *source*, and *sink*.

4.7 Command line tools

The python module comes with the following tools:

- aubio estimate and extract descriptors from sound files
- aubiocut slices sound files at onset or beat timestamps

More command line tools are included along with the library.

- aubioonset outputs the time stamp of detected note onsets
- aubiopitch attempts to identify a fundamental frequency, or pitch, for each frame of the input sound
- aubiomfcc computes Mel-frequency Cepstrum Coefficients
- aubiotrack outputs the time stamp of detected beats
- aubionotes emits midi-like notes, with an onset, a pitch, and a duration
- aubioquiet extracts quiet and loud regions

4.7.1 aubio

```
NAME
  aubio - a command line tool to extract information from sound files

SYNOPSIS

  aubio [-h] [-V] <command> ...

COMMANDS

  The general syntax is "aubio <command> <soundfile> [options]". The following
  commands are available:

  onset         get onset times
  pitch         extract fundamental frequency
  beat          get locations of beats
  tempo         get overall tempo in bpm
```

(continues on next page)

(continued from previous page)

```
  notes        get midi-like notes
  mfcc         extract mel-frequency cepstrum coefficients
  melbands     extract mel-frequency energies per band

  For a list of available commands, use "aubio -h". For more info about each
  command, use "aubio <command> --help".

GENERAL OPTIONS

  These options can be used before any command has been specified.

  -h, --help  show help message and exit

  -V, --version  show version

COMMON OPTIONS

  The following options can be used with all commands:

  <source_uri>, -i <source_uri>, --input <source_uri>  input sound file to
  analyse (required)

  -r <freq>, --samplerate <freq>  samplerate at which the file should be
  represented (default: 0, e.g. samplerate of the input sound)

  -H <size>, --hopsize <size>  overlap size, number of samples between two
  consecutive analysis (default: 256)

  -B <size>, --bufsize <size>  buffer size, number of samples used for each
  analysis, (e.g. FFT length, default: 512)

  -h, --help  show help message and exit

  -T format, --time-format format  select time values output format (samples,
  ms, seconds) (default: seconds)

  -v, --verbose  be verbose (increment verbosity by 1, default: 1)

  -q, --quiet  be quiet (set verbosity to 0)

ONSET

  The following additional options can be used with the "onset" subcommand.

  -m <method>, --method <method>  onset novelty function
  <default|energy|hfc|complex|phase|specdiff|kl|mkl|specflux> (default:
  default)

  -t <threshold>, --threshold <threshold>  threshold (default: unset)

  -s <value>, --silence <value>  silence threshold, in dB (default: -70)

  -M <value>, --minioi <value>  minimum Inter-Onset Interval (default: 12ms)

PITCH

  The following additional options can be used with the "pitch" subcommand.
```

(continues on next page)

(continued from previous page)

```
  -m <method>, --method <method>  pitch detection method
  <default|yinfft|yin|mcomb|fcomb|schmitt> (default: default, e.g. yinfft)

  -t <threshold>, --threshold <threshold>  tolerance (default: unset)

  -s <value>, --silence <value>  silence threshold, in dB (default: -70)

  The default buffer size for the beat algorithm is 2048. The default hop size
  is 256.

BEAT

  The "beat" command accepts all common options and no additional options.

  The default buffer size for the beat algorithm is 1024. The default hop size
  is 512.

TEMPO

  The "tempo" command accepts all common options and no additional options.

  The default buffer size for the beat algorithm is 1024. The default hop size
  is 512.

NOTES

  The following additional options can be used with the "notes" subcommand.

  -s <value>, --silence <value>  silence threshold, in dB (default: -70)

  -d <value>, --release-drop <value>  release drop level, in dB. If the level
  drops more than this amount since the last note started, the note will be
  turned off (default: 10).

MFCC

  The "mfcc" command accepts all common options and no additional options.

MELBANDS

  The "melbands" command accepts all common options and no additional options.

EXAMPLES

  Extract onsets using a minimum inter-onset interval of 30ms:

    aubio onset /path/to/input_file -M 30ms

  Extract pitch with method "mcomb" and a silence threshold of -90dB:

    aubio pitch /path/to/input_file -m mcomb -s -90.0

  Extract MFCC using the standard Slaney implementation:

    aubio mfcc /path/to/input_file -r 44100
```

(continues on next page)

(continued from previous page)

```
SEE ALSO

  aubiocut(1)

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.2 aubiocut

```
NAME
  aubiocut - a command line tool to slice sound files at onset or beat timestamps

SYNOPSIS

  aubiocut source
  aubiocut [[-i] source]
           [-r rate] [-B win] [-H hop]
           [-O method] [-t thres]
           [-b] [-c]
           [-v] [-q] [-h]

OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If set
  to 0, the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -O, --onset method  The onset detection method to use. See ONSET METHODS
  below. Defaults to 'default'.

  -b, --beat  Use beat locations instead of onset locations.

  -t, --onset-threshold thres  Set the threshold value for the onset peak
  picking. Values are typically in the range [0.001, 0.900]. Lower threshold
  values imply more onsets detected. Increasing this threshold should reduce
  the number of incorrect detections. Defaults to 0.3.

  -c, --cut  Cut input sound file at detected labels. A new sound files for
```

(continues on next page)

(continued from previous page)

```
  each slice will be created in the current directory.

  -o, --output directory  Specify the directory path where slices of the
  original source should be created.

  --cut-until-nsamples n  How many extra samples should be added at the end of
  each slice (default 0).

  --cut-until-nslices n  How many extra slices should be added at the end of
  each slice (default 0).

  --create-first  Alway create first slice.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

  -q, --quiet  Be quiet.


ONSET METHODS

  Available methods: default, energy, hfc, complex, phase, specdiff, kl, mkl,
  specflux.

  See aubioonset(1) for details about these methods.


SEE ALSO

  aubioonset(1),
  aubiopitch(1),
  aubiotrack(1),
  aubionotes(1),
  aubioquiet(1),
  and
  aubiomfcc(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.3 aubioonset

```
NAME
  aubioonset - a command line tool to extract musical onset times

SYNOPSIS

  aubioonset source
  aubioonset [[-i] source] [-o sink]
             [-r rate] [-B win] [-H hop]
```

(continues on next page)

(continued from previous page)

```
             [-O method] [-t thres]
             [-T time-format]
             [-s sil] [-m] [-f]
             [-j] [-N miditap-note] [-V miditap-velo]
             [-v] [-h]

DESCRIPTION

  aubioonset attempts to detect onset times, the beginning of discrete sound
  events, in audio signals.

  When started with an input source (-i/--input), the detected onset times are
  given on the console, in seconds.

  When started without an input source, or with the jack option (-j/--jack),
  aubioonset starts in jack mode.

OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -o, --output sink  Save results in this file. The file will be created on
  the model of the input file. Onset times are marked by a short wood-block
  like sound.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -O, --onset method  The onset detection method to use. See ONSET METHODS
  below. Defaults to 'default'.

  -t, --onset-threshold thres  Set the threshold value for the onset peak
  picking. Values are typically in the range [0.001, 0.900]. Lower threshold
  values imply more onsets detected. Increasing this threshold should reduce
  the number of incorrect detections. Defaults to 0.3.

  -M, --minioi value  Set the minimum inter-onset interval, in seconds, the
  shortest interval between two consecutive onsets. Defaults to 0.020

  -s, --silence sil  Set the silence threshold, in dB, under which the onset
  will not be detected. A value of -20.0 would eliminate most onsets but the
  loudest ones. A value of -90.0 would select all onsets. Defaults to -90.0.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.
```

(continues on next page)

(continued from previous page)

```
  -m, --mix-input  Mix source signal to the output signal before writing to
  sink.

  -f, --force-overwrite  Overwrite output file if it already exists.

  -j, --jack  Use Jack input/output. You will need a Jack connection
  controller to feed aubio some signal and listen to its output.

  -N, --miditap-note  Override note value for MIDI tap. Defaults to 69.

  -V, --miditap-velop  Override velocity value for MIDI tap. Defaults to 65.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

ONSET METHODS

  Available methods are:

  default  Default distance, currently hfc

  Default: 'default' (currently set to hfc)

  energy  Energy based distance

  This function calculates the local energy of the input spectral frame.

  hfc  High-Frequency content

  This method computes the High Frequency Content (HFC) of the input
  spectral frame. The resulting function is efficient at detecting
  percussive onsets.

  Paul Masri. Computer modeling of Sound for Transformation and Synthesis of
  Musical Signal. PhD dissertation, University of Bristol, UK, 1996.

  complex  Complex domain onset detection function

  This function uses information both in frequency and in phase to determine
  changes in the spectral content that might correspond to musical onsets.
  It is best suited for complex signals such as polyphonic recordings.

  Christopher Duxbury, Mike E. Davies, and Mark B. Sandler.  Complex domain
  onset detection for musical signals. In Proceedings of the Digital Audio
  Effects Conference, DAFx-03, pages 90-93, London, UK, 2003.

  phase  Phase based onset detection function

  This function uses information both in frequency and in phase to determine
  changes in the spectral content that might correspond to musical onsets. It
  is best suited for complex signals such as polyphonic recordings.

  Juan-Pablo Bello, Mike P. Davies, and Mark B. Sandler.  Phase-based note
  onset detection for music signals. In Proceedings of the IEEE International
  Conference on Acoustics Speech and Signal Processing, pages 441444,
  Hong-Kong, 2003.
```

(continues on next page)

(continued from previous page)

```
  specdiff  Spectral difference onset detection function

  Jonhatan Foote and Shingo Uchihashi. The beat spectrum: a new approach to
  rhythm analysis. In IEEE International Conference on Multimedia and Expo
  (ICME 2001), pages 881884, Tokyo, Japan, August 2001.

  kl  Kulback-Liebler onset detection function

  Stephen Hainsworth and Malcom Macleod. Onset detection in music audio
  signals. In Proceedings of the International Computer Music Conference
  (ICMC), Singapore, 2003.

  mkl  Modified Kulback-Liebler onset detection function

  Paul Brossier, ``Automatic annotation of musical audio for interactive
  systems'', Chapter 2, Temporal segmentation, PhD thesis, Centre for
  Digital music, Queen Mary University of London, London, UK, 2006.

  specflux  Spectral flux

  Simon Dixon, Onset Detection Revisited, in ``Proceedings of the 9th
  International Conference on Digital Audio Effects'' (DAFx-06), Montreal,
  Canada, 2006.

SEE ALSO

  aubiopitch(1),
  aubiotrack(1),
  aubionotes(1),
  aubioquiet(1),
  aubiomfcc(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.4 aubiopitch

```
NAME
  aubiopitch - a command line tool to extract musical pitch

SYNOPSIS

  aubiopitch source
  aubiopitch [[-i] source] [-o sink]
             [-r rate] [-B win] [-H hop]
             [-p method] [-u unit] [-l thres]
             [-T time-format]
             [-s sil] [-f]
```

(continues on next page)

(continued from previous page)

```
             [-v] [-h] [-j]

DESCRIPTION

  aubiopitch attempts to detect the pitch, the perceived height of a musical
  note.

  When started with an input source (-i/--input), the detected pitch are
  printed on the console, prefixed by a timestamp in seconds. If no pitch
  candidate is found, the output is 0.

  When started without an input source, or with the jack option (-j/--jack),
  aubiopitch starts in jack mode.

OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -o, --output sink  Save results in this file. The file will be created on
  the model of the input file. The detected frequency is played at the
  detected loudness.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 2048.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -p, --pitch method  The pitch detection method to use. See PITCH METHODS
  below. Defaults to 'default'.

  -u, --pitch-unit unit  The unit to be used to print frequencies. Possible
  values include midi, bin, cent, and Hz. Defaults to 'Hz'.

  -l, --pitch-tolerance thres  Set the tolerance for the pitch detection
  algorithm. Typical values range between 0.2 and 0.9. Pitch candidates found
  with a confidence less than this threshold will not be selected. The higher
  the threshold, the more confidence in the candidates. Defaults to unset.

  -s, --silence sil  Set the silence threshold, in dB, under which the onset
  will not be detected. A value of -20.0 would eliminate most onsets but the
  loudest ones. A value of -90.0 would select all onsets. Defaults to -90.0.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.

  -m, --mix-input  Mix source signal to the output signal before writing to
  sink.
```

(continues on next page)

(continued from previous page)

```
  -f, --force-overwrite  Overwrite output file if it already exists.

  -j, --jack  Use Jack input/output. You will need a Jack connection
  controller to feed aubio some signal and listen to its output.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

PITCH METHODS

  Available methods are:

  default  use the default method

  Currently, the default method is set to yinfft.

  schmitt  Schmitt trigger

  This pitch extraction method implements a Schmitt trigger to estimate the
  period of a signal. It is computationally very inexpensive, but also very
  sensitive to noise.

  fcomb  a fast harmonic comb filter

  This pitch extraction method implements a fast harmonic comb filter to
  determine the fundamental frequency of a harmonic sound.

  mcomb  multiple-comb filter

  This fundamental frequency estimation algorithm implements spectral
  flattening, multi-comb filtering and peak histogramming.

  specacf  Spectral auto-correlation function

  yin  YIN algorithm

  This algorithm was developed by A. de Cheveigne and H. Kawahara and
  was first published in:

  De Cheveigné, A., Kawahara, H. (2002) "YIN, a fundamental frequency
  estimator for speech and music", J. Acoust. Soc. Am. 111, 1917-1930.

  yinfft  Yinfft algorithm

  This algorithm was derived from the YIN algorithm. In this implementation, a
  Fourier transform is used to compute a tapered square difference function,
  which allows spectral weighting. Because the difference function is tapered,
  the selection of the period is simplified.

  Paul Brossier, Automatic annotation of musical audio for interactive systems,
  Chapter 3, Pitch Analysis, PhD thesis, Centre for Digital music, Queen Mary
  University of London, London, UK, 2006.

  yinfast  YIN algorithm (accelerated)

  An optimised implementation of the YIN algorithm, yielding results identical
```

(continues on next page)

(continued from previous page)

```
  to the original YIN algorithm, while reducing its computational cost from
  O(n^2) to O(n log(n)).

SEE ALSO

  aubioonset(1),
  aubiotrack(1),
  aubionotes(1),
  aubioquiet(1),
  aubiomfcc(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.5 aubiomfcc

```
NAME
  aubiomfcc - a command line tool to compute Mel-Frequency Cepstrum Coefficients

SYNOPSIS

  aubiomfcc source
  aubiomfcc [[-i] source]
            [-r rate] [-B win] [-H hop]
            [-T time-format]
            [-v] [-h]

DESCRIPTION

  aubiomfcc compute the Mel-Frequency Cepstrum Coefficients (MFCC).

  MFCCs are coefficients that make up for the mel-frequency spectrum, a
  representation of the short-term power spectrum of a sound. By default, 13
  coefficients are computed using 40 filters.

  When started with an input source (-i/--input), the coefficients are given on
  the console, prefixed by their timestamps in seconds.

OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.
```

(continues on next page)

(continued from previous page)

```
  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

REFERENCES

  Using the default parameters, the filter coefficients will be computed
  according to Malcolm Slaney's Auditory Toolbox, available at the following
  url:

  https://engineering.purdue.edu/~malcolm/interval/1998-010/ (see file mfcc.m)

SEE ALSO

  aubioonset(1),
  aubiopitch(1),
  aubiotrack(1),
  aubionotes(1),
  aubioquiet(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.6 aubiotrack

```
NAME
  aubiotrack - a command line tool to extract musical beats from audio signals

SYNOPSIS

  aubiotrack source
  aubiotrack [[-i] source] [-o sink]
             [-r rate] [-B win] [-H hop]
             [-T time-format]
             [-s sil] [-m]
             [-j] [-N miditap-note] [-V miditap-velo]
             [-v] [-h]

DESCRIPTION
```

(continues on next page)

(continued from previous page)

```
  aubiotrack attempts to detect beats, the time where one would intuitively be
  tapping his foot.

  When started with an input source (-i/--input), the detected beats are given
  on the console, in seconds.

  When started without an input source, or with the jack option (-j/--jack),
  aubiotrack starts in jack mode.

OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -o, --output sink  Save results in this file. The file will be created on the
  model of the input file. Beats are marked by a short wood-block like sound.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -s, --silence sil  Set the silence threshold, in dB, under which the pitch
  will not be detected. A value of -20.0 would eliminate most onsets but the
  loudest ones. A value of -90.0 would select all onsets. Defaults to -90.0.

  -m, --mix-input  Mix source signal to the output signal before writing to
  sink.

  -f, --force-overwrite  Overwrite output file if it already exists.

  -j, --jack  Use Jack input/output. You will need a Jack connection
  controller to feed aubio some signal and listen to its output.

  -N, --miditap-note  Override note value for MIDI tap. Defaults to 69.

  -V, --miditap-velop  Override velocity value for MIDI tap. Defaults to 65.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

BEAT TRACKING METHODS

  Aubio currently implements one the causal beat tracking algorithm designed by
```

(continues on next page)

(continued from previous page)

```
  Matthew Davies and described in the following articles:

  Matthew E. P. Davies and Mark D. Plumbley. Causal tempo tracking of audio.
  In Proceedings of the International Symposium on Music Information Retrieval
  (ISMIR), pages 164169, Barcelona, Spain, 2004.

  Matthew E. P. Davies, Paul Brossier, and Mark D. Plumbley. Beat tracking
  towards automatic musical accompaniment. In Proceedings of the Audio
  Engineering Society 118th Convention, Barcelona, Spain, May 2005.

SEE ALSO

  aubioonset(1),
  aubiopitch(1),
  aubionotes(1),
  aubioquiet(1),
  aubiomfcc(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.7 aubionotes

```
NAME
  aubionotes - a command line tool to extract musical notes

SYNOPSIS

  aubionotes source
  aubionotes [[-i] source]
             [-r rate] [-B win] [-H hop]
             [-O method] [-t thres] [-d drop]
             [-p method] [-u unit] [-l thres]
             [-T time-format]
             [-s sil]
             [-j] [-v] [-h]

DESCRIPTION

  aubionotes attempts to detect notes by looking for note onsets and pitches.
  Consecutive events are segmented using onset detection, while a fundamental
  frequency extraction algorithm determines their pitch.

  When started with an input source (-i/--input), the detected notes are
  printed on standard output, in seconds and midi note number.

  When started without an input source, or with the jack option (-j/--jack),
  aubionotes starts in jack mode.
```

(continues on next page)

(continued from previous page)

```
OPTIONS

  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -O, --onset method  The onset detection method to use. See ONSET METHODS
  below. Defaults to 'default'.

  -t, --onset-threshold thres  Set the threshold value for the onset peak
  picking. Typical values are typically within 0.001 and 0.900. Defaults to
  0.1. Lower threshold values imply more onsets detected. Try 0.5 in case of
  over-detections. Defaults to 0.3.

  -M, --minioi value  Set the minimum inter-onset interval, in seconds, the
  shortest interval between two consecutive notes. Defaults to 0.030

  -p, --pitch method  The pitch detection method to use. See PITCH METHODS
  below. Defaults to 'default'.

  -u, --pitch-unit unit  The unit to be used to print frequencies. Possible
  values include midi, bin, cent, and Hz. Defaults to 'Hz'.

  -l, --pitch-tolerance thres  Set the tolerance for the pitch detection
  algorithm. Typical values range between 0.2 and 0.9. Pitch candidates found
  with a confidence less than this threshold will not be selected. The higher
  the threshold, the more confidence in the candidates. Defaults to unset.

  -s, --silence sil  Set the silence threshold, in dB, under which the pitch
  will not be detected. A value of -20.0 would eliminate most onsets but the
  loudest ones. A value of -90.0 would select all onsets. Defaults to -90.0.

  -d, --release-drop  Set the release drop threshold, in dB. If the level drops
  more than this amount since the last note started, the note will be turned
  off. Defaults to 10.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.

  -j, --jack  Use Jack input/output. You will need a Jack connection
  controller to feed aubio some signal and listen to its output.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.
```

(continues on next page)

(continued from previous page)

```
ONSET METHODS

  Available methods: default, energy, hfc, complex, phase, specdiff, kl, mkl,
  specflux.

  See aubioonset(1) for details about these methods.

PITCH METHODS

  Available methods: default, schmitt, fcomb, mcomb, specacf, yin, yinfft,
  yinfast.

  See aubiopitch(1) for details about these methods.

SEE ALSO

  aubioonset(1),
  aubiopitch(1),
  aubiotrack(1),
  aubioquiet(1),
  aubiomfcc(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission is
  granted to copy, distribute and/or modify this document under the terms of
  the GNU General Public License as published by the Free Software Foundation,
  either version 3 of the License, or (at your option) any later version.
```

4.7.8 aubioquiet

```
NAME
  aubioquiet - a command line tool to extracts quiet and loud regions from a file

SYNOPSIS

  aubioquiet source
  aubioquiet [[-i] source]
             [-r rate] [-B win] [-H hop]
             [-T time-format]
             [-s sil]
             [-v] [-h]

DESCRIPTION

  aubioquiet will print a timestamp each time it detects a new silent region or
  a new loud region in a sound file.

  When started with an input source (-i/--input), the detected timestamps are
  printed on the console, in seconds.

OPTIONS
```

(continues on next page)

(continued from previous page)

```
  This program follows the usual GNU command line syntax, with long options
  starting with two dashes (--). A summary of options is included below.

  -i, --input source  Run analysis on this audio file. Most uncompressed and
  compressed are supported, depending on how aubio was built.

  -r, --samplerate rate  Fetch the input source, resampled at the given
  sampling rate. The rate should be specified in Hertz as an integer. If 0,
  the sampling rate of the original source will be used. Defaults to 0.

  -B, --bufsize win  The size of the buffer to analyze, that is the length
  of the window used for spectral and temporal computations. Defaults to 512.

  -H, --hopsize hop  The number of samples between two consecutive analysis.
  Defaults to 256.

  -s, --silence sil  Set the silence threshold, in dB, under which the pitch
  will not be detected. Defaults to -90.0.

  -T, --timeformat format  Set time format (samples, ms, seconds). Defaults to
  seconds.

  -h, --help  Print a short help message and exit.

  -v, --verbose  Be verbose.

EXAMPLE OUTPUT

  NOISY: 28.775330

  QUIET: 28.914648


SEE ALSO

  aubioonset(1),
  aubiopitch(1),
  aubiotrack(1),
  aubionotes(1),
  aubiomfcc(1),
  and
  aubiocut(1).

AUTHOR

  This manual page was written by Paul Brossier <piem@aubio.org>. Permission
  is granted to copy, distribute and/or modify this document under the terms
  of the GNU General Public License, Version 3 any later version published
  by the Free Software Foundation.
```

4.7.9 Command line features

feat vs. prg	onset	pitch	mfcc	track	notes	quiet	cut1	short options
input	Y	Y	Y	Y	Y	Y	Y	-i
output	Y	Y	N	Y	Y	N	Y!1	-o,-m,-f
Hz/buf/hop	Y	Y	Y	Y	Y	Y!2	Y	-r,-B-,H
jack	Y	Y	N	Y	Y	N!3	N	-j
onset	Y	N	N	Y!8	Y!6	N	Y	-O,-t,-M
pitch	N	Y	N	N	Y!6	N	N!5	-p,-u,-l
silence	Y	Y	N	Y	Y!7	Y	N!4	-s
timefmt	Y	Y	Y	Y	Y	Y	!	-T
help	Y	Y	Y	Y	Y	Y	Y	-h
verbose	Y	Y	Y	Y	Y	Y	Y	-v

1. `aubiocut --output` is used to specify a directory, not a file.
2. Option `--bufsize` is useless for `aubioquiet`
3. `aubioquiet` could have a jack output
4. Regression, re-add slicing at silences to `aubiocut`
5. `aubiocut` could cut on notes
6. `aubionotes` needs onset/pitch setters.
7. Silence was different for pitch and onset, test.
8. Some `aubiotrack` options should be disabled (minioi, threshold).

4.8 Developing with aubio

Here is a brief overview of the C library.

For a more detailed list of available functions, see the API documentation.

To report issues, ask questions, and request new features, use Github Issues

4.8.1 Design Basics

The library is written in C and is optimised for speed and portability.

All memory allocations take place in the *new_* methods. Each successful call to *new_* should have a matching call to *del_* to deallocate the object.

```
// new_ to create an object foobar
aubio_foobar_t * new_aubio_foobar(void * args);
// del_ to delete foobar
void del_aubio_foobar (aubio_foobar_t * foobar);
```

The main computations are done in the *_do* methods.

```
// _do to process output = foobar(input)
audio_foobar_do (aubio_foobar_t * foobar, fvec_t * input, cvec_t * output);
```

Most parameters can be read and written at any time:

```
// _get_param to get foobar.param
smpl_t aubio_foobar_get_a_parameter (aubio_foobar_t * foobar);
// _set_param to set foobar.param
uint_t aubio_foobar_set_a_parameter (aubio_foobar_t * foobar, smpl_t a_parameter);
```

In some case, more functions are available:

```
// non-real time functions
uint_t aubio_foobar_reset(aubio_foobar_t * t);
```

4.8.2 Basic Types

```
// integers
uint_t n = 10;                // unsigned
sint_t delay = -90;           // signed

// float
smpl_t a = -90.;              // simple precision
lsmp_t f = 0.024;             // double precision

// vector of floats (simple precision)
fvec_t * vec = new_fvec(n);
vec->data[0] = 1;
vec->data[vec->length-1] = 1.; // vec->data has n elements
fvec_print(vec);
del_fvec(vec);

// complex data
cvec_t * fftgrain = new_cvec(n);
vec->norm[0] = 1.;            // vec->norm has n/2+1 elements
vec->phas[n/2] = 3.1415;      // vec->phas as well
del_cvec(fftgrain);

// matrix
fmat_t * mat = new_fmat (height, length);
mat->data[height-1][0] = 1;   // mat->data has height rows
mat->data[0][length-1] = 10;  // mat->data[0] has length columns
del_fmat(mat);
```

4.8.3 Reading a sound file

In this example, aubio_source is used to read a media file.

First, define a few variables and allocate some memory.

```
  uint_t samplerate = 0;
  uint_t hop_size = 256;
  uint_t n_frames = 0, read = 0;

  aubio_source_t* s =
    new_aubio_source(source_path, samplerate, hop_size);
  fvec_t *vec = new_fvec(hop_size);
```

Note: With `samplerate = 0`, `aubio_source` will be created with the file's original samplerate.

Now for the processing loop:

```
do {
  aubio_source_do(s, vec, &read);
  fvec_print (vec);
  n_frames += read;
} while ( read == hop_size );
```

At the end of the processing loop, memory is deallocated:

```
del_fvec (vec);
del_aubio_source (s);
```

See the complete example: `test-source.c`.

4.8.4 Computing a spectrum

Now let's create a phase vocoder:

```
uint_t win_s = 32; // window size
uint_t hop_s = win_s / 4; // hop size

fvec_t * in = new_fvec (hop_s); // input buffer
cvec_t * fftgrain = new_cvec (win_s); // fft norm and phase
fvec_t * out = new_fvec (hop_s); // output buffer
```

The processing loop could now look like:

```
while ( n-- ) {
  // get some fresh input data
  // ..

  // execute phase vocoder
  aubio_pvoc_do (pv,in,fftgrain);

  // do something with fftgrain
  // ...
  cvec_print (fftgrain);

  // optionally rebuild the signal
  aubio_pvoc_rdo(pv,fftgrain,out);

  // and do something with the result
  // ...
  fvec_print (out);
}
```

Time to clean up the previously allocated memory:

```
// clean up
del_fvec(in);
del_cvec(fftgrain);
del_fvec(out);
```

(continues on next page)

(continued from previous page)

```
del_aubio_pvoc(pv);
aubio_cleanup();
```

See the complete example: `test-phasevoc.c`.

4.8.5 Doxygen documentation

The latest version of the API documentation is built using Doxygen and is available at:

> https://aubio.org/doc/latest/

4.8.6 Contribute

Please report any issue and feature request at the Github issue tracker. Patches and pull-requests welcome!

4.9 About

This library gathers a collection of music signal processing algorithms written by several people. The documentation of each algorithms contains a brief description and references to the corresponding papers.

4.9.1 Credits

Many thanks to everyone who contributed to aubio, including:

- Martin Hermant (MartinHN)
- Eduard Müller (emuell)
- Nils Philippsen (nphilipp)
- Tres Seaver (tseaver)
- Dirkjan Rijnders (dirkjankrijnders)
- Jeffrey Kern (anwserman)
- Sam Alexander (sxalexander)

Special thanks to Juan Pablo Bello, Chris Duxbury, Samer Abdallah, Alain de Cheveigne for their help. Also many thanks to Miguel Ramirez and Nicolas Wack for their advices and help fixing bugs.

4.9.2 Publications

Substantial informations about several of the algorithms and their evaluation are gathered in:

- Paul Brossier, Automatic annotation of musical audio for interactive systems, PhD thesis, Centre for Digital music, Queen Mary University of London, London, UK, 2006.

Additional results obtained with this software were discussed in the following papers:

- P. M. Brossier and J. P. Bello and M. D. Plumbley, Real-time temporal segmentation of note objects in music signals in *Proceedings of the International Computer Music Conference*, 2004, Miami, Florida, ICMA

- P. M. Brossier and J. P. Bello and M. D. Plumbley, *Fast labelling of note objects in music signals <https://aubio.org/articles/brossier04fastnotes.pdf>*, in *Proceedings of the International Symposium on Music Information Retrieval*, 2004, Barcelona, Spain

4.9.3 Citation

Please refer to the Zenodo link in the file README.md to cite this release.

4.9.4 Copyright

4.9.5 License

aubio is a free and open source software; **you** can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Note: aubio is not MIT or BSD licensed. Contact us if you need it in your commercial product.

Index