pyAudioDspTools

Release 0.7.9

Arjaan Auinger

Jan 07, 2021

CONTENTS:

pyAudioDspTools is a python 3 package for manipulating audio by just using numpy. This can be from a .wav or as a stream via pyAudio for example. pyAudioDspTool's only requirement is Numpy. The package is only a few kilobytes in size and well documented.

You can use pyAudioDspTools to start learning about audio dsp because all relevant operations are in plain sight, no C or C++ code will be called and nearly no blackboxing takes place. You can also easily modify all available audio effects and start writing your own, you only need to know python and numpy as well as audio dsp basics. As this package is released under the MIT licence, you can use it as you see fit.

To install pyAudioDspTools simply open a terminal in your venv and use pip:

```
pip install pyAudioDspTools
```

The package is only a few kB in size, so it should download in an instant. After installing the package you can import it to your module in Python via:

```
import pyAudioDspTools
```

pyAudioDspTools is device centered. Every sound effect processor is a class, imagine it being like an audio-device. You first create a class/device with certain settings and then run some numpy-arrays (which is your audio data) through them. This always follows a few simple steps, depending on if you want to modify data from a .wav file or realtime-stream. All classes are ready for realtime streaming and manage all relevant variables themselves.

```
image of pipeline here
```

CHAPTER

ONE

USING PYAUDIODSPTOOLS

Below you will find 2 simple examples of processing you data.Example 1 will read a .wav file, process the data and write it to a second .wav file. Example 2 will create a stream via the pyAudio package and process everything in realtime

1.1 Processing from a .wav file.

```
import pyAudioDspTools

# Importing a mono .wav file and then splitting the resulting numpy-array in smaller␣
↪chunks.
full_data = pyAudioDspTools.MonoWavToNumpyFloat("some_path/your_audiofile.wav")
split_data = pyAudioDspTools.MakeChunks(full_data)

# Creating the class/device, which is a lowcut filter
filter_device = pyAudioDspTools.CreateLowCutFilter(800)

# Setting a counter and process the chunks via filter_device.apply
counter = 0
for counter in range(len(split_data)):
    split_data[counter] = filter_device.apply(split_data[counter])
    counter += 1

# Merging the numpy-array back into a single big one and write it to a .wav file.
merged_data = pyAudioDspTools.CombineChunks(split_data)
pyAudioDspTools.NumpyFloatToWav("some_path/output_audiofile.wav", merged_data)
```

1.2 Processing a live feed with pyaudio

```
# Example 2: Creating a live audio stream and processing it by running the data␣
↪though a lowcut filter.
# Is MONO.
# Has to be manually terminated in the IDE.

import pyaudio
import pyAudioDspTools
import time
import numpy
import sys
```

(continues on next page)

(continued from previous page)

```
pyAudioDspTools.sampling_rate = 44100
pyAudioDspTools.chunk_size = 512

filterdevice = pyAudioDspTools.CreateLowCutFilter(300)


# Instantiate PyAudio
pyaudioinstance = pyaudio.PyAudio()

# The callback function first reads the current input and converts it to a numpy␣
↪array, filters it and returns it.
def callback(in_data, frame_count, time_info, status):
   in_data = numpy.frombuffer(in_data, dtype=numpy.float32)
   in_data = filterdevice.apply(in_data)
   #print(numpydata)
   return (in_data, pyaudio.paContinue)


# The stream class of pyaudio. Setting all the variables, pretty self explanatory.
stream = pyaudioinstance.open(format=pyaudio.paFloat32,
               channels=1,
               rate=pyAudioDspTools.sampling_rate,
               input = True,
               output = True,
               frames_per_buffer = pyAudioDspTools.chunk_size,
               stream_callback = callback)

# start the stream
stream.start_stream()

# wait
while stream.is_active():
   time.sleep(5)
   print("Cpu load:", stream.get_cpu_load())

# stop stream
stream.stop_stream()
stream.close()

# close PyAudio
pyaudioinstance.terminate()
sys.exit()
```

CHAPTER

TWO

SUBMODULES

class pyAudioDspTools.EffectCompressor.**CreateCompressor**(*threshold_in_db=-15*, *ratio=0.6*, *attack_in_ms=3.1*, *release_in_ms=30.1*)

Bases: object

Creating a compressor audio-effect class/device

Can be used to limit dynamic range of a signal. Very effective on drums for example. Is overloaded with basic settings. This class introduces no latency.

Parameters

- **threshold_in_db** (*int or float*) – Sets the threshold when the gate becomes active. Must be negative.
- **ratio** (*float*) – The depth of the effect. Must be a value between >0 and <1.0
- **attack** (*float*) – The attack-time of the gate in milliseconds
- **release** (*float*) – The release-time of the gate in milliseconds

apply(*int_array_input*)

Applying the Gate to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectDelay.**CreateDelay**(*time_in_ms=500*, *feedback_loops=2*, *lowcut_filter_frequency=40*, *highcut_filter_frequency=12000*, *use_lowcut_filter=False*, *use_highcut_filter=False*, *wet=False*)

Bases: object

Creating a Delay audio-effect class/device.

Is overloaded with basic settings. This class introduces no latency

Parameters

- **time_in_ms** (*int or float*) – Sets the delay-time in milliseconds.
- **feedback_loops** (*int or float*) – Sets the amount of repetitions of the delay.

- **lowcut_filter_frequency** (*int or float*) – The frequency of the audio filter, if use_lowcut_filter is set to True
- **highcut_filter_frequency** (*int or float*) – The frequency of the audio filter, if use_highcut_filter is set to True
- **use_lowcut_filter** (*bool*) – If use_lowcut_filter is set to True, it will apply a lowcut to the processed input array.
- **use_highcut_filter** (*bool*) – If use_highcut_filter is set to True, it will apply a highcut to the processed input array.
- **wet** (*bool*) – If set to True it will just return the delay and not mix it with the original signal. Used for parallel processing.

apply(*float32_array_input*)
Applying the 3 Band FFT EQ to a numpy-array.

Parameters **float32_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectEQ3Band.**CreateEQ3Band**(*low_shelf_frequency*, *low_shelf_gain*, *mid_frequency*, *mid_gain*, *high_shelf_frequency*, *high_shelf_gain*)

Bases: object

Creating a 3Band FFT EQ audio-effect class/device.

Can be used to manipulate frequencies in your audio numpy-array. Is based on Robert Bristow-Johnson's Audio EQ Cookbook. Is the slower one, the faster, FFT based one being CreateEQ3BandFFT. Is NOT overloaded with basic settings. This class introduces no latency.

Parameters

- **low_shelf_frequency** (*int or float*) – Sets the frequency of the lowshelf-band in Hertz.
- **low_shelf_gain** (*int or float*) – Increase or decrease the lows in decibel.
- **mid_frequency** (*int or float*) – Sets the frequency of the mid-band in Hertz. Has a fixed Q.
- **mid_gain** (*int or float*) – Increase or decrease the selected mids in decibel.
- **high_shelf_frequency** (*int or float*) – Sets the frequency of the highshelf-band in Hertz.
- **high_shelf_gain** (*int or float*) – Increase or decrease the highs in decibel.

applyhighband(*float_array_input*)
Applying the high-band to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

applylowband(*float_array_input*)
Applying the low-band to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

applymidband(*float_array_input*)
Applying the mid-band to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectEQ3BandFFT.**CreateEQ3BandFFT**(*lowshelf_frequency*, *lowshelf_db*, *midband_frequency*, *midband_db*, *highshelf_frequency*, *highshelf_db*)

Bases: object

Creating a 3Band FFT EQ audio-effect class/device.

Can be used to manipulate frequencies in your audio numpy-array. Is the faster one, the slower, non FFT based one being CreateEQ3Band. Is NOT overloaded with basic settings. This class introduces latency equal to config.chunk_size.

Parameters

- **lowshelf_frequency** (*int or float*) – Sets the frequency of the lowshelf-band in Hertz.
- **lowshelf_db** (*int or float*) – Increase or decrease the lows in decibel.
- **midband_frequency** (*int or float*) – Sets the frequency of the mid-band in Hertz. Has a fixed Q.
- **midband_db** (*int or float*) – Increase or decrease the selected mids in decibel.
- **highshelf_frequency** (*int or float*) – Sets the frequency of the highshelf-band in Hertz.
- **highshelf_db** (*int or float*) – Increase or decrease the highs in decibel.

apply(*float32_array_input*)
Applying the 3 Band FFT EQ to a numpy-array.

Parameters **float32_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectFFTFilter.**CreateHighCutFilter**(*cutoff_frequency=8000*)

Bases: object

Creating a FFT filter audio-effect class/device.

Cuts the upper frequencies of a signal. Is overloaded with basic settings. This class introduces latency equal to chunk_size.

Parameters **cutoff_frequency** (*int or float*) – Sets the rolloff frequency for the high cut filter.

apply (*float32_array_input*)
Applying the filter to a numpy-array

Parameters **float32_array_input** (*float*) – The array, which the effect should be applied on.

Returns The previously processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectFFTFilter.**CreateLowCutFilter**(*cutoff_frequency=160*)
Bases: object

Creating a FFT filter audio-effect class/device.

Cuts the lower frequencies of a signal. Is overloaded with basic settings. This class introduces latency equal to chunk_size.

Parameters **cutoff_frequency** (*int or float*) – Sets the rolloff frequency for the high cut filter.

apply (*float32_array_input*)
Applying the filter to a numpy-array

Parameters **float32_array_input** (*float*) – The array, which the effect should be applied on.

Returns The previously processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectGate.**CreateGate**(*threshold_in_db=- 5*, *depth=0.1*, *attack=3.1*, *release=200.1*)
Bases: object

Creating a gate audio-effect class/device

Can be used to duck noise and bleed. Very effective on drums for example. For cleaner effect use short attack time and moderate release time. Is overloaded with basic settings. This class introduces no latency.

Parameters

- **threshold_in_db** (*int or float*) – Sets the threshold when the gate becomes active. Must be negative
- **depth** (*float*) – The depth of the effect. Must be a value between >0 and <1.0
- **attack** (*float*) – The attack-time of the gate in milliseconds
- **release** (*float*) – The release-time of the gate in milliseconds

apply (*int_array_input*)
Applying the Gate to a numpy-array

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectHardDistortion.**CreateHardDistortion**
Bases: object

Creating a distortion audio-effect class/device.

Its a wave-shaper and messes with dynamic range, but doesn't introduce latency.

Parameters **None** (*None*) –

apply(*float_array_input*)
Applying the distortion to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectSaturator.**CreateSaturator**(*saturation_threshold_in_db=-20.0*, *makeup_gain=2.0*, *mode='hard'*)
Bases: object

Creating a saturator audio-effect class/device

Is a wave-shaper and messes with dynamic range, but doesn't introduce latency.

Parameters

- **saturation_threshold_in_db** (*int or float*) – Sets the threshold when the saturator becomes active. Must be negative.
- **makeup_gain** (*float*) – Makeup for Volume-loss due to wave-shaping in decibel.
- **mode** (*string*) – The mode of the Saturator. Can be 'hard' or 'soft'

apply(*float_array_input*)
Applying the Saturator to a numpy-array

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectSoftClipper.**CreateSoftClipper**(*drive=0.44*)
Bases: object

Creating a limiter-kind audio-effect class/device

Its a wave-shaper and messes with dynamic range, but doesn't introduce latency.

Parameters **drive** (*float*) – A value between 0.0 and 1.0, 0.0 meaning no wave shaping at all and 1.0 full drive.

Notes

- You can go beyond 1.0, but I designed it to be at the sweet spot. Go to 70.0 if you want, but be warned.

apply(*float_array_input*)
Applying the Soft Clipper to a numpy-array

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

class pyAudioDspTools.EffectTremolo.**CreateTremolo**(*tremolo_depth=0.4*, *lfo_in_hertz=4.5*)

Bases: object

Creating a tremolo audio-effect class/device

Creates a LFO and applies it to the input array to modulate power.

Parameters

- **tremolo_depth** (*float*) – Sets the depth of the effect. Must be a value between >0 and <1.0
- **lfo_in_hertz** (*float*) – Sets the cycle of the LFO in seconds.

apply(*float_array_input*)

Applying the Tremolo to a numpy-array.

Parameters **float_array_input** (*float*) – The array, which the effect should be applied on.

Returns The processed array, should be the exact same size as the input array

Return type float

reset()

Resets the LFO of the Tremolo.

Parameters **None** (*None*) –

pyAudioDspTools.Generators.**CreateSinewave**(*sin_frequency*, *sin_length_in_samples*, *sin_sample_rate=44100*)

Generates a sine wave with selected properties.

Parameters

- **sin_frequency** (*int*) – The frequency of the sine wave.
- **sin_length_in_samples** (*int*) – The lenght of the sine wave in samples. Is your sin_sample_rate is 44100 and your sin_length_in_samples is set to 44100 your sine wave signal will be exactly 1 second long for example
- **sin_sample_rate** (*int*) – Is set to sampling_rate from your config.py by default. Use pyAudioDspTools.sampling_rate=48000 in your script to change your sampling rate globally to 48000 hertz for example.

Returns The created array

Return type numpy array

pyAudioDspTools.Generators.**CreateSquarewave**(*square_frequency*, *square_length_in_samples*, *square_sample_rate=44100*)

Generates a square wave with selected properties.

Parameters

- **square_frequency** (*int*) – The frequency of the sine wave.
- **square_length_in_samples** (*int*) – The lenght of the square wave in samples. Is your square_sample_rate is 44100 and your square_length_in_samples is set to 44100 your square wave signal will be exactly 1 second long for example

- **square_sample_rate** (*int*) – Is set to sampling_rate from your config.py by default. Use pyAudioDspTools.sampling_rate=48000 in your script to change your sampling rate globally to 48000 hertz for example.

Returns The created array

Return type numpy array

pyAudioDspTools.Generators.**CreateWhitenoise**(*noise_length_in_samples*, *sample_rate=44100*)

Generates noise with selected properties.

Parameters

- **noise_length_in_samples** (*int*) – The lenght of the sine wave in samples. Is your square_sample_rate is 44100 and your square_length_in_samples is set to 44100 your noise signal will be exactly 1 second long for example
- **square_sample_rate** (*int*) – Is set to sampling_rate from your config.py by default. Use pyAudioDspTools.sampling_rate=48000 in your script to change your sampling rate globally to 48000 hertz for example.

Returns The created array

Return type numpy array

pyAudioDspTools.Utility.**CombineChunks**(*float_array_input*)

Converts a sliced array back into one long one. Use this if you want to write to .wav

Parameters **float_array_input** (*float*) – The array, which you want to slice.

Returns The sliced arrays.

Return type numpy array

pyAudioDspTools.Utility.**Convert16BitTodBV**(*int_array_input*)

pyAudioDspTools.Utility.**ConvertdBVTo16Bit**(*float_array_input*)

pyAudioDspTools.Utility.**Dither16BitTo8Bit**(*int_array_input*)

pyAudioDspTools.Utility.**Dither32BitIntTo16BitInt**(*int_array_input*)

pyAudioDspTools.Utility.**InfodBV**(*float_array_input*)

Prints the average sum as decibel whereas 1.0 is 0dB.

Parameters **float_array_input** (*float*) – The audio data.

Returns **dBV** – Average power in dB.

Return type float

pyAudioDspTools.Utility.**InfodBV16Bit**(*int_array_input*)

Prints the average sum as decibel whereas 32767 is 0dB.

Parameters **int_array_input** (*int*) – The audio data.

Returns **dB16** – Average power in dB.

Return type float

pyAudioDspTools.Utility.**MakeChunks**(*float32_array_input*)

Converts a long numpy array in multiple small ones for processing

Parameters **float_array_input** (*float*) – The array, which you want to slice.

Returns The sliced arrays.

Return type numpy array

pyAudioDspTools.Utility.**MixSignals**(**args*)
Adds several numpy arrays. Used for mixing audio signals

Parameters **args** (*1D numpy-arrays*) – Multiple arrays.

Returns A single array.

Return type 1D numpy array

pyAudioDspTools.Utility.**MonoWavToNumpy16BitInt**(*wav_file_path*)
Imports a .wav file as a numpy array. All values will be scaled to be between -32768 and 32767.

Parameters **wav_file_path** (*string*) – Follows the normal python path rules.

Returns **numpy array** – The imported array

Return type int16

pyAudioDspTools.Utility.**MonoWavToNumpyFloat**(*wav_file_path*)
Imports a .wav file as a numpy array. All values will be scaled to be between -1.0 and 1.0 for further processing.

Parameters **wav_file_path** (*string*) – Follows the normal python path rules.

Returns **numpy array** – The imported array

Return type float

pyAudioDspTools.Utility.**NumpyFloatToWav**(*filename*, *data*)
Write a numpy array as a WAV file :param filename: Output wav file :type filename: string or open file handle :param rate: The sample rate (in samples/sec). :type rate: int :param data: A 1-D or 2-D numpy array of either integer or float data-type. :type data: ndarray

Notes

- The file can be an open file or a filename.
- Writes a simple uncompressed WAV file.
- The bits-per-sample will be determined by the data-type.
- To write multiple-channels, use a 2-D array of shape (Nsamples, Nchannels).

pyAudioDspTools.Utility.**VolumeChange**(*float_array_input*, *gain_change_in_db*, *overflow_protection=True*)
Increases or decreses the volume of a signal in decibel.

Parameters

- **float_array_input** (*float*) – The array, which you want to be processed.
- **gain_change_in_db** (*float*) – The amount of change in volume in decibel.
- **overflow_protection** (*bool*) – If true it will clip every value above 1.0 and below -1.0 to 1.0 and -1.0

Returns The processed array

Return type numpy array

Stores variables like chunk_size, also called buffer size by audio professionals, sampling rate and others. Has a default setting of 44100 Hz (44.1 kHz) and a chunk size of 512

param sampling_rate Sets the global sampling rate of all classes/devices. Defaults to 44100 hertz, which is an audio standard.

type sampling_rate int

param chunk_size The number of samples in a chunk. Audio professionals might also call this buffer size, as this is the term used in a number of DAWs such as Ableton, Logic and Pro Tools.

type chunk_size int

Notes

- To set the sampling rate and chunk_size simply overwrite them in your script. Write this in the beginning of your

script: 'pyAudioDspTools.sampling_rate = 48000' or 'pyAudioDspTools.chunk_size = 512'

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

M

N

P

R

V