
Pypianoroll

Hao-Wen Dong

Feb 13, 2021

CONTENTS

1	Features	3
2	Why Pypianoroll	5
3	Installation	7
4	Documentation	9
5	Citing	11
6	Lakh Pianoroll Dataset	13
7	Contents	15
7.1	Getting Started	15
7.2	Pypianoroll Classes	16
7.3	Save & Load	17
7.4	Read & Write	18
7.5	Visualization	20
7.6	Metrics	24
7.7	Technical Documentation	27
	Python Module Index	47
	Index	49

Pypianoroll is an open source Python library for working with piano rolls. It provides essential tools for handling multitrack piano rolls, including efficient I/O as well as manipulation, visualization and evaluation tools.

FEATURES

- Manipulate multitrack piano rolls intuitively
- Visualize multitrack piano rolls beautifully
- Save and load multitrack piano rolls in a space-efficient format
- Parse MIDI files into multitrack piano rolls
- Write multitrack piano rolls into MIDI files

WHY PYPIANOROLL

Our aim is to provide convenient classes for piano-roll matrix and MIDI-like track information (program number, track name, drum track indicator). Pypianoroll is also designed to provide efficient I/O for piano rolls, since piano rolls have long been considered an inefficient way to store music data due to the sparse nature.

INSTALLATION

To install Pypianoroll, please run `pip install pypianoroll`. To build Pypianoroll from source, please download the [source](#) and run `python setup.py install`.

DOCUMENTATION

Documentation is available [here](#) and as docstrings with the code.

CITING

Please cite the following paper if you use Pypianoroll in a published work:

Hao-Wen Dong, Wen-Yi Hsiao, and Yi-Hsuan Yang, “Pypianoroll: Open Source Python Package for Handling Multi-track Pianorolls,” in *Late-Breaking Demos of the 19th International Society for Music Information Retrieval Conference (ISMIR)*, 2018.

[\[homepage\]](#) [\[paper\]](#) [\[poster\]](#) [\[code\]](#) [\[documentation\]](#)

LAKH PIANOROLL DATASET

[Lakh Pianoroll Dataset](#) (LPD) is a new multitrack piano roll dataset using Pypianoroll for efficient data I/O and to save space, which is used as the training dataset in our [MuseGAN](#) project.

CONTENTS

7.1 Getting Started

Welcome to Pypianoroll! We will go through some basic concepts in this tutorial.

Hint: Be sure you have Pypianoroll installed. To install Pypianoroll, please run `pip install pypianoroll`.

In the following example, we will use [this MIDI file](#) as an example.

First of all, let's import the Pypianoroll library.

```
import pypianoroll
```

Now, let's read the example MIDI file into a Multitrack object.

```
multitrack = pypianoroll.read("fur-elise.mid")
print(multitrack)
```

Here's what we got.

```
Multitrack(name=None, resolution=24, tempo=array(shape=(8976, 1)),
↳downbeat=array(shape=(8976, 1)), tracks=[StandardTrack(name='', program=0, is_
↳drum=False, pianoroll=array(shape=(8976, 128))), StandardTrack(name='', program=0,
↳is_drum=False, pianoroll=array(shape=(8976, 128)))])
```

You can use dot notation to assess the data. For example, `multitrack.resolution` returns the temporal resolution (in time steps per quarter note) and `multitrack.tracks[0].pianoroll` returns the piano roll for the first track.

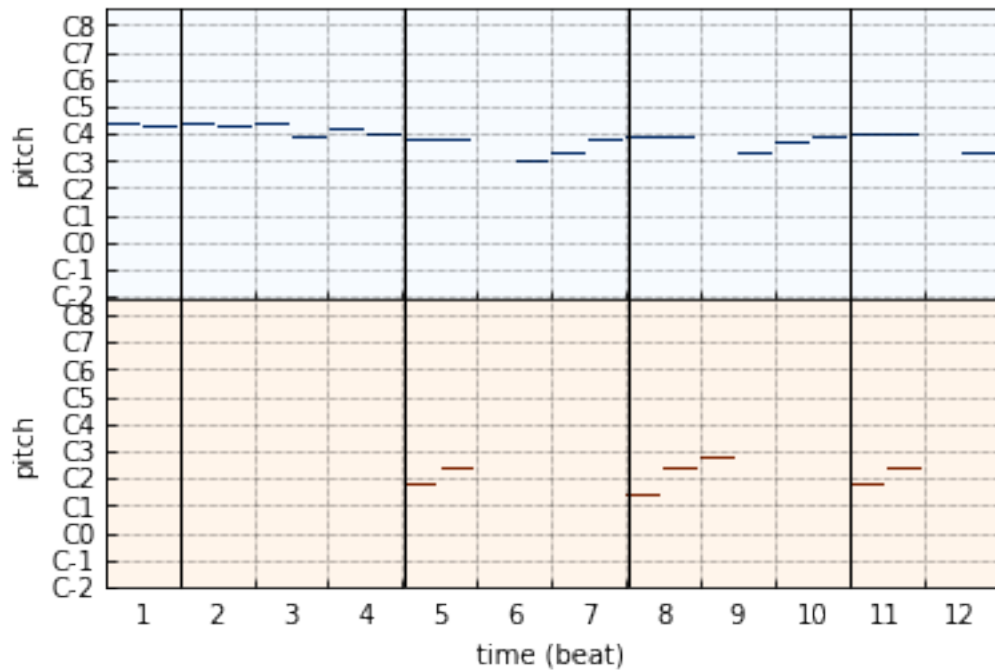
Pypianoroll provides some functions for manipulating the multitrack piano roll. For example, we can trim and binarize the multitrack as follows.

```
multitrack.trim(0, 12 * multitrack.resolution)
multitrack.binarize()
```

Pypianoroll also provides visualization supports.

```
multitrack.plot()
```

This will give us the following plot.



7.2 Pypianoroll Classes

We provide several classes for working with multitrack piano rolls.

- `pypianoroll.Multitrack`: A multitrack that stores track(s) along with global information.
- `pypianoroll.Track`: A track that stores the a piano roll along with track information.
- `pypianoroll.StandardTrack`: A standard track that holds a piano roll with velocity information.
- `pypianoroll.BinaryTrack`: A binary track that holds a piano roll without velocity information.

7.2.1 Multitrack Class

At-tribute	Description	Type	Default
name	Name of the multitrack	str	
resolu-tion	Time steps per quarter note	int	<code>pypianoroll.DEFAULT_RESOLUTION</code>
tempo	Tempo at each time step	NumPy array of dtype float	
down-beat	Downbeat positions	NumPy array of dtype bool	
tracks	Music tracks	list of <code>pypianoroll.Track</code>	<code>[]</code>

7.2.2 Track Class

Attribute	Description	Type	Default
name	Name of the track	str	
program	MIDI program number	int	<code>pypianoroll.DEFAULT_PROGRAM</code>
is_drum	If it is a drum track	bool	<code>pypianoroll.DEFAULT_IS_DRUM</code>
pianoroll	Downbeat positions	NumPy array	<code>np.zeros((0, 128))</code>

For `pypianoroll.StandardTrack`, the piano roll array must be of data type `uint8` and take values in `[0, 127]`. For `pypianoroll.BinaryTrack`, the piano roll array must be of data type `bool`.

7.3 Save & Load

Pypianoroll supports efficient I/O for `pypianoroll.Multitrack` objects. The piano rolls will be first converted to sparse matrices (specifically, instances of `scipy.sparse.csc_matrix`) and then stored in a NPZ file.

7.3.1 Functions

`pypianoroll.save(path: Union[str, pathlib.Path], multitrack: Multitrack, compressed: bool = True)`

Save a Multitrack object to a NPZ file.

Parameters

- **path** (*str or Path*) – Path to the NPZ file to save.
- **multitrack** (*pypianoroll.Multitrack*) – Multitrack to save.
- **compressed** (*bool*) – Whether to save to a compressed NPZ file. Defaults to `True`.

Notes

To reduce the file size, the piano rolls are first converted to instances of `scipy.sparse.csc_matrix`. The component arrays are then collected and saved to a npz file.

See also:

`pypianoroll.load()` Load a NPZ file into a Multitrack object.

`pypianoroll.write()` Write a Multitrack object to a MIDI file.

Note: The saved `.npz` file is basically a zip archive which contains the following files:

- component arrays of the piano rolls in compressed sparse column format:
 - `pianoroll_[index]_csc_data.npy`
 - `pianoroll_[index]_csc_indices.npy`
 - `pianoroll_[index]_csc_indptr.npy`
- `tempo.npy`: the tempo array
- `downbeat.npy`: the down beat array
- `info.json`: a JSON file that contains meta data and track information

`pypianoroll.load(path: Union[str, pathlib.Path]) → pypianoroll.multitrack.Multitrack`
Load a NPZ file into a Multitrack object.

Supports only files previously saved by `pypianoroll.save()`.

Parameters `path` (*str* or *Path*) – Path to the file to load.

See also:

`pypianoroll.save()` Save a Multitrack object to a NPZ file.

`pypianoroll.read()` Read a MIDI file into a Multitrack object.

7.4 Read & Write

Pypianoroll currently supports reading from and writing to MIDI files.

7.4.1 Functions

`pypianoroll.read(path: Union[str, pathlib.Path], **kwargs) → pypianoroll.multitrack.Multitrack`
Read a MIDI file into a Multitrack object.

Parameters

- `path` (*str* or *Path*) – Path to the file to read.
- `**kwargs` – Keyword arguments to pass to `pypianoroll.from_pretty_midi()`.

See also:

`pypianoroll.write()` Write a Multitrack object to a MIDI file.

`pypianoroll.load()` Load a NPZ file into a Multitrack object.

`pypianoroll.write(path: str, multitrack: Multitrack)`
Write a Multitrack object to a MIDI file.

Parameters

- `path` (*str*) – Path to write the file.
- `multitrack` (`pypianoroll.Multitrack`) – Multitrack to save.

See also:

`pypianoroll.read()` Read a MIDI file into a Multitrack object.

`pypianoroll.save()` Save a Multitrack object to a NPZ file.

`pypianoroll.from_pretty_midi(midi: pretty_midi.pretty_midi.PrettyMIDI, resolution: int = 24, mode: str = 'max', algorithm: str = 'normal', collect_onsets_only: bool = False, first_beat_time: Optional[float] = None) → pypianoroll.multitrack.Multitrack`

Return a Multitrack object converted from a PrettyMIDI object.

Parse a `pretty_midi.PrettyMIDI` object. The data type of the resulting piano rolls is automatically determined (int if 'mode' is 'sum' and np.uint8 if mode is 'max').

Parameters

- **midi** (`pretty_midi.PrettyMIDI`) – PrettyMIDI object to parse.
- **mode** (`{ 'max', 'sum' }`) – Merging strategy for duplicate notes. Defaults to ‘max’.
- **algorithm** (`{ 'normal', 'strict', 'custom' }`) – Algorithm for finding the location of the first beat (see Notes). Defaults to ‘normal’.
- **collect_onsets_only** (`bool`) – True to collect only the onset of the notes (i.e. note on events) in all tracks, where the note off and duration information are discarded. False to parse regular piano rolls. Defaults to False.
- **first_beat_time** (`float, optional`) – Location of the first beat, in sec. Required and only effective when using ‘custom’ algorithm.

Returns Converted Multitrack object.

Return type `pypianoroll.Multitrack`

Notes

There are three algorithms for finding the location of the first beat:

- ‘normal’ : Estimate the location of the first beat using `pretty_midi.PrettyMIDI.estimate_beat_start()`.
- ‘strict’ : Set the location of the first beat to the time of the first time signature change. Raise a `RuntimeError` if no time signature change is found.
- ‘custom’ : Set the location of the first beat to the value of argument `first_beat_time`. Raise a `ValueError` if `first_beat_time` is not given.

If an incomplete beat before the first beat is found, an additional beat will be added before the (estimated) beat starting time. However, notes before the (estimated) beat starting time for more than one beat are dropped.

`pypianoroll.to_pretty_midi` (`multitrack: Multitrack, default_tempo: Optional[float] = None, default_velocity: int = 64`) → `pretty_midi.pretty_midi.PrettyMIDI`

Return a Multitrack object as a PrettyMIDI object.

Parameters

- **default_tempo** (`int`) – Default tempo to use. Defaults to the first element of attribute `tempo`.
- **default_velocity** (`int`) – Default velocity to assign to binarized tracks. Defaults to 64.

Returns Converted PrettyMIDI object.

Return type `pretty_midi.PrettyMIDI`

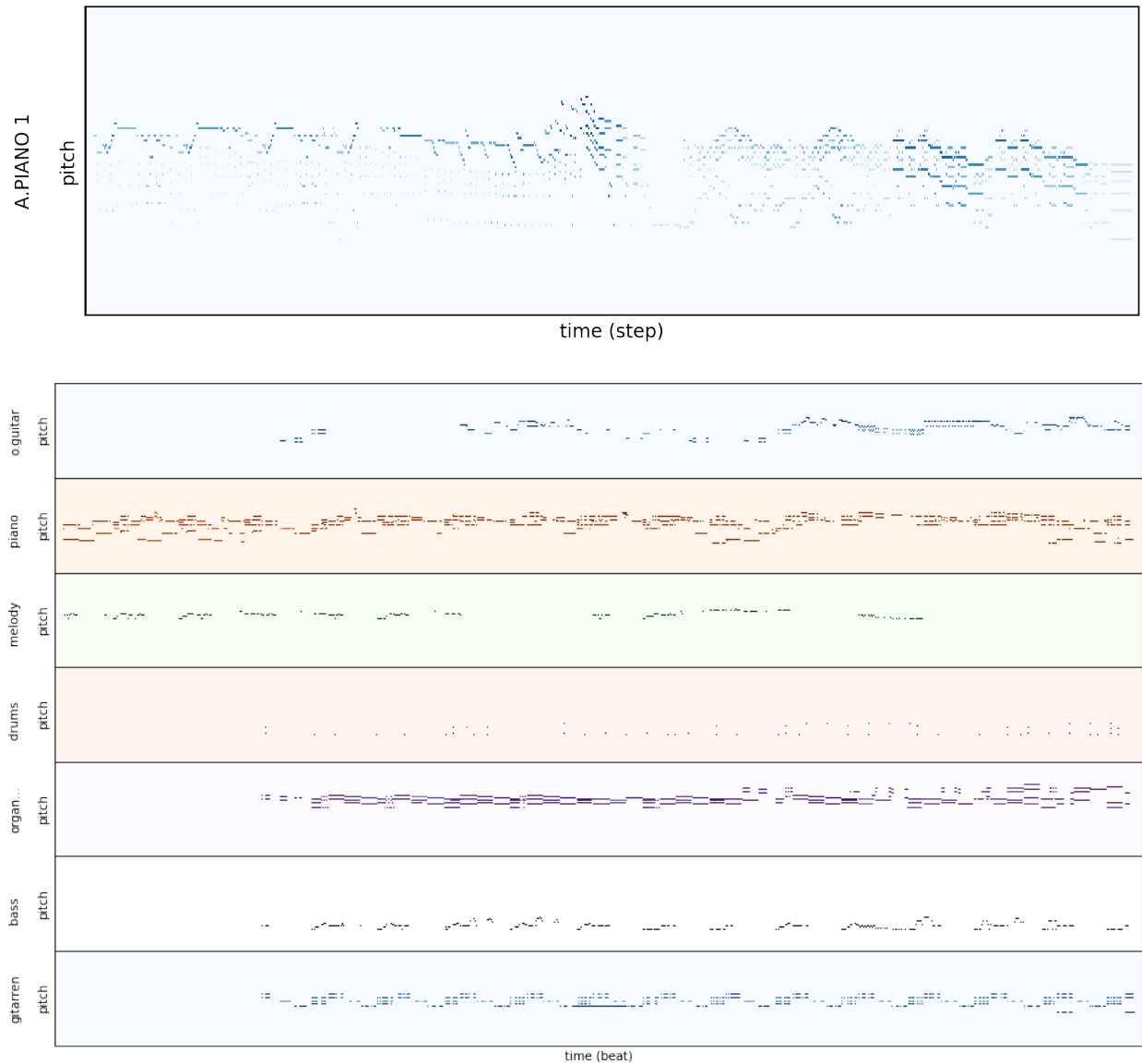
Notes

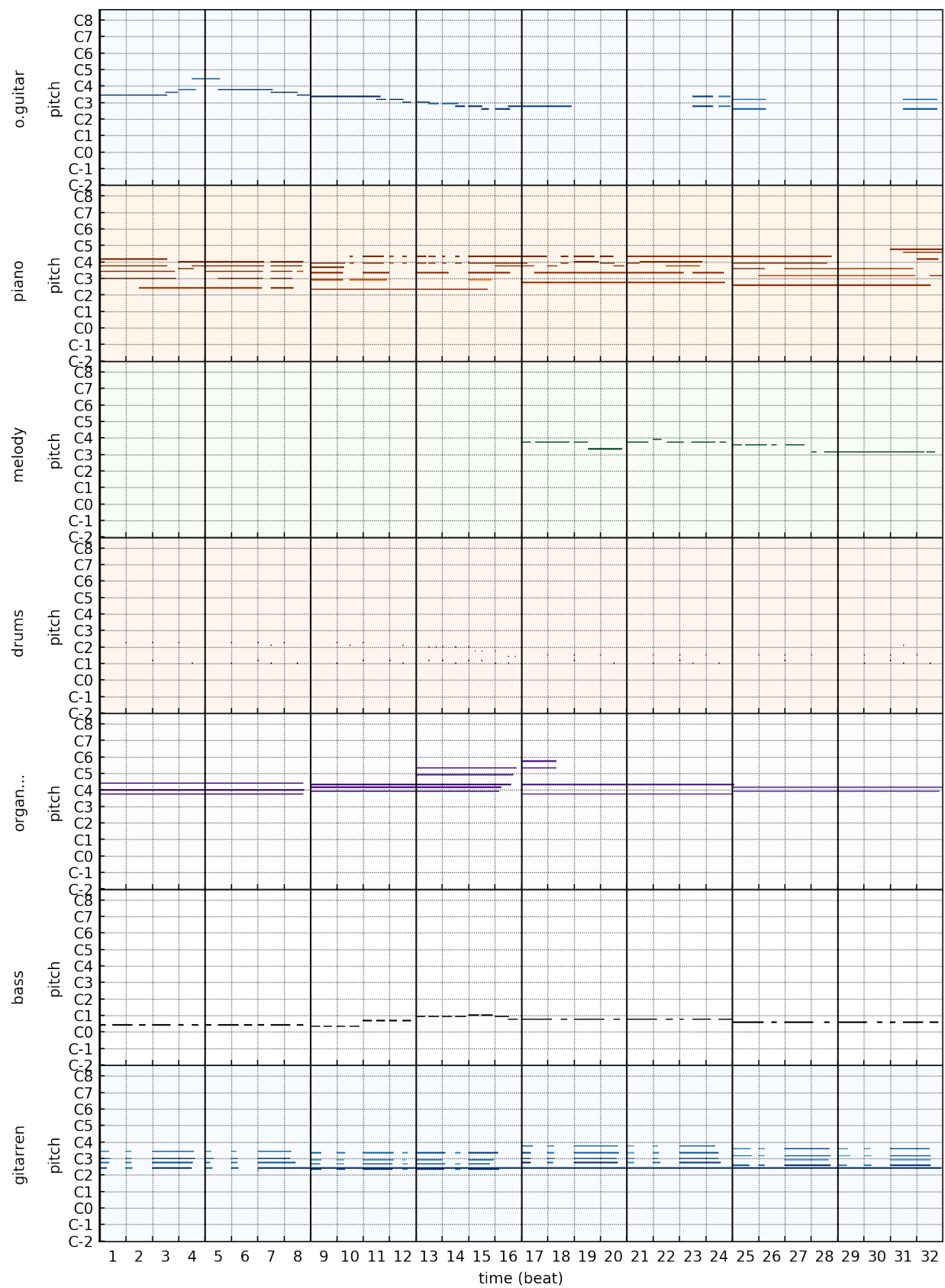
- Tempo changes are not supported.
- Time signature changes are not supported.
- The velocities of the converted piano rolls will be clipped to [0, 127].
- Adjacent nonzero values of the same pitch will be considered a single note with their mean as its velocity.

Note: Writing the tempo array and downbeat array to tempo change and time signature change events are not supported yet.

7.5 Visualization

Pypianoroll provides tools for visualizing piano rolls. Here are some examples.





7.5.1 Functions

`pypianoroll.plot` (*obj*: `_MultitrackOrTrack`, ***kwargs*) → `_MultitrackOrTrack`

Plot the object.

See `pypianoroll.plot_multitrack()` and `pypianoroll.plot_track()` for full documentation.

`pypianoroll.plot_multitrack` (*multitrack*: `Multitrack`, *axis*: `Optional[Sequence[matplotlib.axes._axes.Axes]]`, *mode*: `str = 'separate'`, *track_label*: `str = 'name'`, *preset*: `str = 'full'`, *cmaps*: `Optional[Sequence[str]] = None`, *xtick*: `str = 'auto'`, *ytick*: `str = 'octave'`, *xticklabel*: `bool = True`, *yticklabel*: `str = 'auto'`, *tick_loc*: `Sequence[str] = ('bottom', 'left')`, *tick_direction*: `str = 'in'`, *label*: `str = 'both'`, *grid_axis*: `str = 'both'`, *grid_linestyle*: `str = ':'`, *grid_linewidth*: `float = 0.5`, ***kwargs*) → `List[matplotlib.axes._axes.Axes]`

Plot the multitrack.

Parameters

- **multitrack** (`pypianoroll.Multitrack`) – Multitrack to plot.
- **axis** (sequence of `matplotlib.axes.Axes`) – Axes to plot the tracks on.
- **mode** (`{'separate', 'blended', 'hybrid'}`) – Plotting strategy for visualizing multiple tracks. For ‘separate’ mode, plot each track separately. For ‘blended’, blend and plot the pianoroll as a colored image. For ‘hybrid’ mode, drum tracks are blended into a ‘Drums’ track and all other tracks are blended into an ‘Others’ track. Defaults to ‘separate’.
- **track_label** (`{'name', 'program', 'family', 'off'}`) – Track label format. When *mode* is ‘hybrid’, all options other than ‘off’ will label the two track with ‘Drums’ and ‘Others’.
- **preset** (`{'full', 'frame', 'plain'}`) – Preset theme to use. For ‘full’ preset, ticks, grid and labels are on. For ‘frame’ preset, ticks and grid are both off. For ‘plain’ preset, the x- and y-axis are both off. Defaults to ‘full’.
- **cmaps** (*tuple or list*) – Colormaps. Will be passed to `matplotlib.pyplot.imshow()`. Only effective when *pianoroll* is 2D. Defaults to ‘Blues’. If *mode* is ‘separate’, defaults to (‘Blues’, ‘Oranges’, ‘Greens’, ‘Reds’, ‘Purples’, ‘Greys’). If *mode* is ‘blended’, defaults to (‘hsv’). If *mode* is ‘hybrid’, defaults to (‘Blues’, ‘Greens’).
- ****kwargs** – Keyword arguments to pass to `pypianoroll.plot_pianoroll()`.

Returns (Created) list of Axes objects.

Return type list of `matplotlib.axes.Axes`

`pypianoroll.plot_track` (*track*: `Track`, *ax*: `Optional[matplotlib.axes._axes.Axes] = None`, ***kwargs*) → `matplotlib.axes._axes.Axes`

Plot a track.

Parameters

- **track** (`pypianoroll.Track`) – Track to plot.
- **ax** (`matplotlib.axes.Axes`) – Axes to plot the piano roll on. Defaults to call `plt.gca()`.
- ****kwargs** – Keyword arguments to pass to `pypianoroll.plot_pianoroll()`.

Returns (Created) Axes object.

Return type `matplotlib.axes.Axes`

`pypianoroll.plot_pianoroll` (*ax*: `matplotlib.axes._axes.Axes`, *pianoroll*: `numpy.ndarray`, *is_drum*: `bool = False`, *resolution*: `Optional[int] = None`, *downbeats*: `Optional[Sequence[int]] = None`, *preset*: `str = 'full'`, *cmap*: `str = 'Blues'`, *xtick*: `str = 'auto'`, *ytick*: `str = 'octave'`, *xticklabel*: `bool = True`, *yticklabel*: `str = 'auto'`, *tick_loc*: `Sequence[str] = ('bottom', 'left')`, *tick_direction*: `str = 'in'`, *label*: `str = 'both'`, *grid_axis*: `str = 'both'`, *grid_linestyle*: `str = ':'`, *grid_linewidth*: `float = 0.5`, ***kwargs*)

Plot a piano roll.

Parameters

- **ax** (`matplotlib.axes.Axes`) – Axes to plot the piano roll on.
- **pianoroll** (`ndarray`, `shape=(?, 128)`, `(?, 128, 3)` or `(?, 128, 4)`) – Piano roll to plot. For a 3D piano-roll array, the last axis can be either RGB or RGBA.
- **is_drum** (`bool`) – Whether it is a percussion track. Defaults to `False`.
- **resolution** (`int`) – Time steps per quarter note. Required if *xtick* is ‘beat’.
- **downbeats** (`list`) – Boolean array that indicates whether the time step contains a downbeat (i.e., the first time step of a bar).
- **preset** (`{'full', 'frame', 'plain'}`) – Preset theme. For ‘full’ preset, ticks, grid and labels are on. For ‘frame’ preset, ticks and grid are both off. For ‘plain’ preset, the x- and y-axis are both off. Defaults to ‘full’.
- **cmap** (`str` or `matplotlib.colors.Colormap`) – Colormap. Will be passed to `matplotlib.pyplot.imshow()`. Only effective when *pianoroll* is 2D. Defaults to ‘Blues’.
- **xtick** (`{'auto', 'beat', 'step', 'off'}`) – Tick format for the x-axis. For ‘auto’ mode, set to ‘beat’ if *resolution* is given, otherwise set to ‘step’. Defaults to ‘auto’.
- **ytick** (`{'octave', 'pitch', 'off'}`) – Tick format for the y-axis. Defaults to ‘octave’.
- **xticklabel** (`bool`) – Whether to add tick labels along the x-axis.
- **yticklabel** (`{'auto', 'name', 'number', 'off'}`) – Tick label format for the y-axis. For ‘name’ mode, use pitch name as tick labels. For ‘number’ mode, use pitch number. For ‘auto’ mode, set to ‘name’ if *ytick* is ‘octave’ and ‘number’ if *ytick* is ‘pitch’. Defaults to ‘auto’.
- **tick_loc** (`sequence of {'bottom', 'top', 'left', 'right'}`) – Tick locations. Defaults to `(‘bottom’, ‘left’)`.
- **tick_direction** (`{'in', 'out', 'inout'}`) – Tick direction. Defaults to ‘in’.
- **label** (`{'x', 'y', 'both', 'off'}`) – Whether to add labels to x- and y-axes. Defaults to ‘both’.
- **grid_axis** (`{'x', 'y', 'both', 'off'}`) – Whether to add grids to the x- and y-axes. Defaults to ‘both’.
- **grid_linestyle** (`str`) – Grid line style. Will be passed to `matplotlib.axes.Axes.grid()`.
- **grid_linewidth** (`float`) – Grid line width. Will be passed to `matplotlib.axes.Axes.grid()`.
- ****kwargs** – Keyword arguments to be passed to `matplotlib.axes.Axes.imshow()`.

7.6 Metrics

Pypianoroll provides several objective metrics proposed in the literature. These objective metrics could be used to evaluate a music generation system by comparing the statistical difference between the training data and the generated samples.

7.6.1 Functions

`pypianoroll.empty_beat_rate` (*pianoroll*: *numpy.ndarray*, *resolution*: *int*) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero.

$$\text{empty_beat_rate} = \frac{\#(\text{empty_beats})}{\#(\text{beats})}$$

Parameters `pianoroll` (*ndarray*) – Piano roll to evaluate.

Returns Empty-beat rate.

Return type float

`pypianoroll.n_pitches_used` (*pianoroll*: *numpy.ndarray*) → int

Return the number of unique pitches used.

Parameters `pianoroll` (*ndarray*) – Piano roll to evaluate.

Returns Number of unique pitch classes used.

Return type int

See also:

`pypianoroll.n_pitch_class_used()` Compute the number of unique pitch classes used.

`pypianoroll.n_pitch_classes_used` (*pianoroll*: *numpy.ndarray*) → int

Return the number of unique pitch classes used.

Parameters `pianoroll` (*ndarray*) – Piano roll to evaluate.

Returns Number of unique pitch classes used.

Return type int

See also:

`pypianoroll.n_pitches_used()` Compute the number of unique pitches used.

`pypianoroll.pitch_range_tuple` (*pianoroll*) → Tuple[float, float]

Return the pitch range as a tuple (*lowest*, *highest*).

Returns

- *int or nan* – Highest active pitch.
- *int or nan* – Lowest active pitch.

See also:

`pypianoroll.pitch_range()` Compute the pitch range.

`pypianoroll.pitch_range(pianoroll) → float`

Return the pitch range.

Returns Pitch range (in semitones), i.e., difference between the highest and the lowest active pitches.

Return type int or nan

See also:

`pypianoroll.pitch_range_tuple()` Return the pitch range as a tuple.

`pypianoroll.qualified_note_rate(pianoroll: numpy.ndarray, threshold: float = 2) → float`

Return the ratio of the number of the qualified notes.

The qualified note rate is defined as the ratio of the number of qualified notes (notes longer than *threshold*, in time steps) to the total number of notes. Return NaN if no note is found.

$$\text{qualified_note_rate} = \frac{\#(\text{notes_longer_than_the_threshold})}{\#(\text{notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **threshold** (*int*) – Threshold of note length to count into the numerator.

Returns Qualified note rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.polyphonic_rate(pianoroll: numpy.ndarray, threshold: float = 2) → float`

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1], where it is called *polyphonicity*.

$$\text{polyphony_rate} = \frac{\#(\text{time_steps_where_multiple_pitches_are_on})}{\#(\text{time_steps})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

Returns Polyphony rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.drum_in_pattern_rate` (*pianoroll*: *numpy.ndarray*, *resolution*: *int*, *tolerance*: *float* = 0.1) → float

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$\text{drum_in_pattern_rate} = \frac{\#(\text{drum_notes_in_pattern})}{\#(\text{drum_notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **resolution** (*int*) – Time steps per beat.
- **tolerance** (*float*) – Tolerance. Defaults to 0.1.

Returns Drum-in-pattern rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.in_scale_rate` (*pianoroll*: *numpy.ndarray*, *root*: *int* = 3, *mode*: *str* = 'major') → float

Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return NaN if no note is found. This metric is used in [1].

$$\text{pitch_in_scale_rate} = \frac{\#(\text{notes_in_scale})}{\#(\text{notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str*, {'major', 'minor'}) – Mode of the scale.

Returns Pitch-in-scale rate.

Return type float

See also:

`muspy.scale_consistency()` Compute the largest pitch-in-class rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.tonal_distance` (*pianoroll_1*: *numpy.ndarray*, *pianoroll_2*: *numpy.ndarray*, *resolution*: *int*, *radii*: *Sequence[float] = (1.0, 1.0, 0.5)*) → *float*

Return the tonal distance [1] between the two input piano rolls.

Parameters

- **pianoroll_1** (*ndarray*) – First piano roll to evaluate.
- **pianoroll_2** (*ndarray*) – Second piano roll to evaluate.
- **resolution** (*int*) – Time steps per beat.
- **radii** (*tuple of float*) – Radii of the three tonal circles (see Equation 3 in [1]).

References

1. Christopher Harte, Mark Sandler, and Martin Gasser, “Detecting harmonic change in musical audio,” in Proceedings of the 1st ACM workshop on Audio and music computing multimedia, 2006.

7.7 Technical Documentation

A Python library for handling multitrack pianorolls.

Pypianoroll is an open source Python library for working with piano rolls. It provides essential tools for handling multitrack piano rolls, including efficient I/O as well as manipulation, visualization and evaluation tools.

7.7.1 Features

- Manipulate multitrack piano rolls intuitively
- Visualize multitrack piano rolls beautifully
- Save and load multitrack piano rolls in a space-efficient format
- Parse MIDI files into multitrack piano rolls
- Write multitrack piano rolls into MIDI files

```
class pypianoroll.BinaryTrack (name: Optional[str] = None, program: Optional[int] = None, is_drum: Optional[bool] = None, pianoroll: Optional[numpy.ndarray] = None)
```

A container for single-track, binary piano rolls.

name

Track name.

Type str, optional

program

Program number according to General MIDI specification [1]. Defaults to 0 (Acoustic Grand Piano).

Type int, 0-127, optional

is_drum

Whether it is a percussion track. Defaults to False.

Type bool, optional

pianoroll

Piano-roll matrix. The first dimension represents time, and the second dimension represents pitch. Cast to bool if not of data type bool.

Type ndarray, dtype=bool, shape=(?, 128), optional

References

1. <https://www.midi.org/specifications/item/gm-level-1-sound-set>

set_nonzeros (*value: int*) → pypianoroll.track.StandardTrack

Assign a constant value to all nonzeros entries.

Parameters *value* (*int*) – Value to assign.

Returns

Return type Converted StandardTrack object.

```
class pypianoroll.Multitrack (name: Optional[str] = None, resolution: Optional[int] =  
                             None, tempo: Optional[numpy.ndarray] = None, down-  
                             beat: Optional[numpy.ndarray] = None, tracks: Op-  
                             tional[Sequence[pypianoroll.track.Track]] = None)
```

A container for multitrack piano rolls.

This is the core class of Pypianoroll.

name

Multitrack name.

Type str, optional

resolution

Time steps per quarter note.

Type int

tempo

Tempo (in qpm) at each time step. Length is the total number of time steps. Cast to float if not of data type float.

Type ndarray, dtype=float, shape=(?, 1), optional

downbeat

Boolean array that indicates whether the time step contains a downbeat (i.e., the first time step of a bar). Length is the total number of time steps.

Type ndarray, dtype=bool, shape=(?, 1), optional

tracks

Music tracks.

Type sequence of *pypianoroll.Track*, optional

append (*track: pypianoroll.track.Track*) → *_Multitrack*

Append a Track object to the track list.

Parameters *track* (*pypianoroll.Track*) – Track to append.

Returns**Return type** Object itself.**binarize** (*threshold: float = 0*) → `_Multitrack`

Binarize the piano rolls.

Parameters **threshold** (*int or float*) – Threshold to binarize the piano rolls. Defaults to zero.**Returns****Return type** Object itself.**blend** (*mode: Optional[str] = None*) → `numpy.ndarray`

Return the blended pianoroll.

Parameters **mode** (*{ 'sum', 'max', 'any' }, optional*) – Blending strategy to apply along the track axis. For ‘sum’ mode, integer summation is performed for binary piano rolls. Defaults to ‘sum’.**Returns** Blended piano roll.**Return type** `ndarray`, shape=(?, 128)**clip** (*lower: int = 0, upper: int = 127*) → `_Multitrack`Clip (limit) the the piano roll into [*lower, upper*].**Parameters**

- **lower** (*int*) – Lower bound. Defaults to 0.
- **upper** (*int*) – Upper bound. Defaults to 127.

Returns**Return type** Object itself.

Note: Only affect `StandardTrack` instances.

copy ()

Return a copy of the multitrack.

Returns**Return type** A copy of the object itself.**Notes**Arrays are copied using `numpy.copy()`.**count_downbeat** () → `int`

Return the number of downbeats.

Returns Number of downbeats.**Return type** `int`

Note: Return value is calculated based only on the attribute *downbeat*.

get_downbeat_steps() → `numpy.ndarray`

Return the indices of time steps that contain downbeats.

Returns Indices of time steps that contain downbeats.

Return type `ndarray`, `dtype=int`

get_length() → `int`

Return the maximum active length of the piano rolls.

Returns Maximum active length (in time steps) of the piano rolls, where active length is the length of the piano roll without trailing silence.

Return type `int`

get_max_length() → `int`

Return the maximum length of the piano rolls.

Returns Maximum length (in time steps) of the piano rolls.

Return type `int`

is_valid(*attr: Optional[str] = None*) → `bool`

Return True if an attribute is valid.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns Whether the attribute has a valid type and value.

Return type `bool`

is_valid_type(*attr: Optional[str] = None*) → `bool`

Return True if an attribute is of a valid type.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns Whether the attribute is of a valid type.

Return type `bool`

pad(*pad_length*) → `_Multitrack`

Pad the piano rolls.

Notes

The lengths of the resulting piano rolls are not guaranteed to be the same.

Parameters **pad_length** (*int*) – Length to pad along the time axis.

Returns

Return type Object itself.

See also:

`pypianoroll.Multitrack.pad_to_multiple()` Pad the piano rolls so that their lengths are some multiples.

`pypianoroll.Multitrack.pad_to_same()` Pad the piano rolls so that they have the same length.

pad_to_multiple (*factor: int*) → *_Multitrack*

Pad the piano rolls so that their lengths are some multiples.

Pad the piano rolls at the end along the time axis of the minimum length that makes the lengths of the resulting piano rolls multiples of *factor*.

Parameters **factor** (*int*) – The value which the length of the resulting piano rolls will be a multiple of.

Returns

Return type Object itself.

Notes

Lengths of the resulting piano rolls are necessarily the same.

See also:

pypianoroll.Multitrack.pad() Pad the piano rolls.

pypianoroll.Multitrack.pad_to_same() Pad the piano rolls so that they have the same length.

pad_to_same () → *_Multitrack*

Pad the piano rolls so that they have the same length.

Pad shorter piano rolls at the end along the time axis so that the resulting piano rolls have the same length.

Returns

Return type Object itself.

See also:

pypianoroll.Multitrack.pad() Pad the piano rolls.

pypianoroll.Multitrack.pad_to_multiple() Pad the piano rolls so that their lengths are some multiples.

plot (*axs: Optional[Sequence[matplotlib.axes._axes.Axes]] = None, **kwargs*) → *numpy.ndarray*

Plot the multitrack piano roll.

Refer to *pypianoroll.plot_multitrack()* for full documentation.

remove_empty () → *_Multitrack*

Remove tracks with empty pianorolls.

save (*path: str, compressed: bool = True*)

Save to a NPZ file.

Refer to *pypianoroll.save()* for full documentation.

set_nonzeros (*value: int*) → *_Multitrack*

Assign a constant value to all nonzero entries.

Parameters **value** (*int*) – Value to assign.

Returns

Return type Object itself.

set_resolution (*resolution: int, rounding: Optional[str] = 'round'*) → *_Multitrack*

Set the resolution.

Parameters

- **resolution** (*int*) – Target resolution.
- **rounding** ({*'round', 'ceil', 'floor'*}) – Rounding mode. Defaults to *'round'*.

Returns

Return type Object itself.

stack () → *numpy.ndarray*

Return the piano rolls stacked as a 3D tensor.

Returns Stacked piano roll, provided as (*track, time, pitch*).

Return type ndarray, shape=(?, ?, 128)

to_pretty_midi (***kwargs*)

Return as a PrettyMIDI object.

Refer to *pypianoroll.to_pretty_midi()* for full documentation.

transpose (*semitone: int*) → *_Multitrack*

Transpose the piano rolls by a number of semitones.

Parameters **semitone** (*int*) – Number of semitones to transpose. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

Notes

Drum tracks are skipped.

trim (*start: Optional[int] = None, end: Optional[int] = None*) → *_Multitrack*

Trim the trailing silences of the piano rolls.

Parameters

- **start** (*int, optional*) – Start time. Defaults to 0.
- **end** (*int, optional*) – End time. Defaults to active length.

Returns

Return type Object itself.

validate (*attr=None*) → *_Multitrack*

Raise an error if an attribute has an invalid type or value.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns

Return type Object itself.

validate_type (*attr=None*)

Raise an error if an attribute has an invalid type.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns**Return type** Object itself.**write** (*path*: *str*)

Write to a MIDI file.

Refer to `pypianoroll.write()` for full documentation.

```
class pypianoroll.StandardTrack (name: Optional[str] = None, program: Optional[int] = None, is_drum: Optional[bool] = None, pianoroll: Optional[numpy.ndarray] = None)
```

A container for single-track piano rolls with velocities.

name

Track name.

Type *str*, optional**program**

Program number according to General MIDI specification [1]. Defaults to 0 (Acoustic Grand Piano).

Type *int*, 0-127, optional**is_drum**

Whether it is a percussion track. Defaults to False.

Type *bool*, optional**pianoroll**

Piano-roll matrix. The first dimension represents time, and the second dimension represents pitch. Cast to uint8 if not of data type uint8.

Type *ndarray*, *dtype*=uint8, *shape*=(?, 128), optional**References**

1. <https://www.midi.org/specifications/item/gm-level-1-sound-set>

clip (*lower*: *int* = 0, *upper*: *int* = 127) → *_StandardTrack*Clip (limit) the the piano roll into [*lower*, *upper*].**Parameters**

- **lower** (*int*) – Lower bound. Defaults to 0.
- **upper** (*int*) – Upper bound. Defaults to 127.

Returns**Return type** Object itself.**set_nonzeros** (*value*: *int*) → *_StandardTrack*

Assign a constant value to all nonzeros entries.

Parameters **value** (*int*) – Value to assign.**Returns****Return type** Object itself.

```
class pypianoroll.Track (name: Optional[str] = None, program: Optional[int] = None, is_drum: Optional[bool] = None, pianoroll: Optional[numpy.ndarray] = None)
```

A generic container for single-track piano rolls.

name
Track name.
Type str, optional

program
Program number according to General MIDI specification [1]. Defaults to 0 (Acoustic Grand Piano).
Type int, 0-127, optional

is_drum
Whether it is a percussion track. Defaults to False.
Type bool, optional

pianoroll
Piano-roll matrix. The first dimension represents time, and the second dimension represents pitch.
Type ndarray, shape=(?, 128), optional

References

1. <https://www.midi.org/specifications/item/gm-level-1-sound-set>

binarize (*threshold: float = 0*) → pypianoroll.track.BinaryTrack
Binarize the track.

This will binarize the piano roll by the given threshold.

Parameters **threshold** (*int or float*) – Threshold. Defaults to 0.

Returns

Return type Converted BinaryTrack object.

copy ()
Return a copy of the track.

Returns

Return type A copy of the object itself.

Notes

The piano-roll array is copied using `numpy.copy()`.

get_length () → int
Return the active length of the piano roll.

Returns Length (in time steps) of the piano roll without trailing silence.

Return type int

is_valid (*attr: Optional[str] = None*) → bool
Return True if an attribute is valid.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns Whether the attribute has a valid type and value.

Return type bool

is_valid_type (*attr: Optional[str] = None*) → bool
Return True if an attribute is of a valid type.

Parameters `attr` (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns Whether the attribute is of a valid type.

Return type bool

pad (*pad_length: int*) → `_Track`

Pad the piano roll.

Parameters `pad_length` (*int*) – Length to pad along the time axis.

Returns

Return type Object itself.

See also:

`pypianoroll.Track.pad_to_multiple()` Pad the piano roll so that its length is some multiple.

pad_to_multiple (*factor: int*) → `_Track`

Pad the piano roll so that its length is some multiple.

Pad the piano roll at the end along the time axis of the minimum length that makes the length of the resulting piano roll a multiple of *factor*.

Parameters `factor` (*int*) – The value which the length of the resulting piano roll will be a multiple of.

Returns

Return type Object itself.

See also:

`pypianoroll.Track.pad()` Pad the piano roll.

plot (*ax: Optional[matplotlib.axes._axes.Axes] = None, **kwargs*) → `matplotlib.axes._axes.Axes`

Plot the piano roll.

Refer to `pypianoroll.plot_track()` for full documentation.

standardize () → `pypianoroll.track.StandardTrack`

Standardize the track.

This will clip the piano roll to [0, 127] and cast to `np.uint8`.

Returns

Return type Converted `StandardTrack` object.

transpose (*semitone: int*) → `_Track`

Transpose the piano roll by a number of semitones.

Parameters `semitone` (*int*) – Number of semitones to transpose. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

trim (*start: Optional[int] = None, end: Optional[int] = None*) → `_Track`

Trim the piano roll.

Parameters

- **start** (*int, optional*) – Start time. Defaults to 0.

- **end** (*int*, *optional*) – End time. Defaults to active length.

Returns

Return type Object itself.

validate (*attr=None*)

Raise an error if an attribute has an invalid type or value.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns

Return type Object itself.

validate_type (*attr=None*)

Raise an error if an attribute has an invalid type.

Parameters **attr** (*str*) – Attribute to validate. Defaults to validate all attributes.

Returns

Return type Object itself.

`pypianoroll.binarize` (*obj*: *Union[pypianoroll.multitrack.Multitrack, pypianoroll.track.StandardTrack]*, *threshold: int = 0*)

Binarize the piano roll(s).

Parameters

- **obj** (*pypianoroll.Multitrack* or *pypianoroll.StandardTrack*) – Object to binarize.
- **threshold** (*int*) – Threshold. Defaults to 0.

`pypianoroll.clip` (*obj: _StandardTrack*, *lower: int = 0*, *upper: int = 127*) → *_StandardTrack*

Clip (limit) the the piano roll(s) into [*lower*, *upper*].

Parameters

- **obj** (*pypianoroll.Multitrack* or *pypianoroll.StandardTrack*) – Object to clip.
- **lower** (*int*) – Lower bound. Defaults to 0.
- **upper** (*int*) – Upper bound. Defaults to 127.

Returns

Return type Object itself.

`pypianoroll.drum_in_pattern_rate` (*pianoroll: numpy.ndarray*, *resolution: int*, *tolerance: float = 0.1*) → *float*

Return the ratio of drum notes in a certain drum pattern.

The drum-in-pattern rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Only drum tracks are considered. Return NaN if no drum note is found. This metric is used in [1].

$$\text{drum_in_pattern_rate} = \frac{\#(\text{drum_notes_in_pattern})}{\#(\text{drum_notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **resolution** (*int*) – Time steps per beat.
- **tolerance** (*float*) – Tolerance. Defaults to 0.1.

Returns Drum-in-pattern rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.empty_beat_rate` (*pianoroll*: `numpy.ndarray`, *resolution*: `int`) → float

Return the ratio of empty beats.

The empty-beat rate is defined as the ratio of the number of empty beats (where no note is played) to the total number of beats. Return NaN if song length is zero.

$$\text{empty_beat_rate} = \frac{\#(\text{empty_beats})}{\#(\text{beats})}$$

Parameters `pianoroll` (`ndarray`) – Piano roll to evaluate.

Returns Empty-beat rate.

Return type float

`pypianoroll.from_pretty_midi` (*midi*: `pretty_midi.pretty_midi.PrettyMIDI`, *resolution*: `int` = 24, *mode*: `str` = 'max', *algorithm*: `str` = 'normal', *collect_onsets_only*: `bool` = False, *first_beat_time*: `Optional[float]` = None) → `pypianoroll.multitrack.Multitrack`

Return a Multitrack object converted from a PrettyMIDI object.

Parse a `pretty_midi.PrettyMIDI` object. The data type of the resulting piano rolls is automatically determined (int if 'mode' is 'sum' and `np.uint8` if *mode* is 'max').

Parameters

- **midi** (`pretty_midi.PrettyMIDI`) – PrettyMIDI object to parse.
- **mode** ({ 'max', 'sum' }) – Merging strategy for duplicate notes. Defaults to 'max'.
- **algorithm** ({ 'normal', 'strict', 'custom' }) – Algorithm for finding the location of the first beat (see Notes). Defaults to 'normal'.
- **collect_onsets_only** (`bool`) – True to collect only the onset of the notes (i.e. note on events) in all tracks, where the note off and duration information are discarded. False to parse regular piano rolls. Defaults to False.
- **first_beat_time** (`float`, *optional*) – Location of the first beat, in sec. Required and only effective when using 'custom' algorithm.

Returns Converted Multitrack object.

Return type `pypianoroll.Multitrack`

Notes

There are three algorithms for finding the location of the first beat:

- ‘normal’ : Estimate the location of the first beat using `pretty_midi.PrettyMIDI.estimate_beat_start()`.
- ‘strict’ : Set the location of the first beat to the time of the first time signature change. Raise a `RuntimeError` if no time signature change is found.
- ‘custom’ : Set the location of the first beat to the value of argument `first_beat_time`. Raise a `ValueError` if `first_beat_time` is not given.

If an incomplete beat before the first beat is found, an additional beat will be added before the (estimated) beat starting time. However, notes before the (estimated) beat starting time for more than one beat are dropped.

`pypianoroll.in_scale_rate(pianoroll: numpy.ndarray, root: int = 3, mode: str = 'major') → float`
Return the ratio of pitches in a certain musical scale.

The pitch-in-scale rate is defined as the ratio of the number of notes in a certain scale to the total number of notes. Drum tracks are ignored. Return `NaN` if no note is found. This metric is used in [1].

$$\text{pitch_in_scale_rate} = \frac{\#(\text{notes_in_scale})}{\#(\text{notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **root** (*int*) – Root of the scale.
- **mode** (*str*, { 'major', 'minor' }) – Mode of the scale.

Returns Pitch-in-scale rate.

Return type float

See also:

`muspy.scale_consistency()` Compute the largest pitch-in-class rate.

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.load(path: Union[str, pathlib.Path]) → pypianoroll.multitrack.Multitrack`
Load a NPZ file into a Multitrack object.

Supports only files previously saved by `pypianoroll.save()`.

Parameters **path** (*str* or *Path*) – Path to the file to load.

See also:

`pypianoroll.save()` Save a Multitrack object to a NPZ file.

`pypianoroll.read()` Read a MIDI file into a Multitrack object.

`pypianoroll.n_pitch_classes_used(pianoroll: numpy.ndarray) → int`
Return the number of unique pitch classes used.

Parameters `pianoroll` (*ndarray*) – Piano roll to evaluate.

Returns Number of unique pitch classes used.

Return type `int`

See also:

`pypianoroll.n_pitches_used()` Compute the number of unique pitches used.

`pypianoroll.n_pitches_used(pianoroll: numpy.ndarray) → int`

Return the number of unique pitches used.

Parameters `pianoroll` (*ndarray*) – Piano roll to evaluate.

Returns Number of unique pitch classes used.

Return type `int`

See also:

`pypianoroll.n_pitch_class_used()` Compute the number of unique pitch classes used.

`pypianoroll.pad(obj: _MultitrackOrTrack, pad_length: int) → _MultitrackOrTrack`

Pad the piano roll(s).

Notes

The lengths of the resulting piano rolls are not guaranteed to be the same. See `pypianoroll.Multitrack.pad_to_same()`.

Parameters

- **obj** (`pypianoroll.Multitrack` or `pypianoroll.Track`) – Object to pad.
- **pad_length** (*int*) – Length to pad along the time axis.

Returns

Return type Object itself.

See also:

`pypianoroll.pad_to_same()` Pad the piano rolls so that they have the same length.

`pypianoroll.pad_to_multiple()` Pad the piano rolls so that their lengths are some multiples.

`pypianoroll.pad_to_multiple(obj: _MultitrackOrTrack, factor: int) → _MultitrackOrTrack`

Pad the piano roll(s) so that their lengths are some multiples.

Pad the piano rolls at the end along the time axis of the minimum length that makes the lengths of the resulting piano rolls multiples of *factor*.

Parameters

- **obj** (`pypianoroll.Multitrack` or `pypianoroll.Track`) – Object to pad.
- **factor** (*int*) – The value which the length of the resulting pianoroll(s) will be a multiple of.

Returns

Return type Object itself.

Notes

Lengths of the resulting piano rolls are necessarily the same.

See also:

`pypianoroll.pad()` Pad the piano rolls.

`pypianoroll.pad_to_same()` Pad the piano rolls so that they have the same length.

`pypianoroll.pad_to_same(obj: _Multitrack) → _Multitrack`

Pad the piano rolls so that they have the same length.

Pad shorter piano rolls at the end along the time axis so that the resulting piano rolls have the same length.

Parameters `obj` (`pypianoroll.Multitrack`) – Object to pad.

Returns

Return type Object itself.

See also:

`pypianoroll.pad()` Pad the piano rolls.

`pypianoroll.pad_to_multiple()` Pad the piano rolls so that their lengths are some multiples.

`pypianoroll.pitch_range(pianoroll) → float`

Return the pitch range.

Returns Pitch range (in semitones), i.e., difference between the highest and the lowest active pitches.

Return type int or nan

See also:

`pypianoroll.pitch_range_tuple()` Return the pitch range as a tuple.

`pypianoroll.pitch_range_tuple(pianoroll) → Tuple[float, float]`

Return the pitch range as a tuple (*lowest, highest*).

Returns

- *int or nan* – Highest active pitch.
- *int or nan* – Lowest active pitch.

See also:

`pypianoroll.pitch_range()` Compute the pitch range.

`pypianoroll.plot(obj: _MultitrackOrTrack, **kwargs) → _MultitrackOrTrack`

Plot the object.

See `pypianoroll.plot_multitrack()` and `pypianoroll.plot_track()` for full documentation.

```
pypianoroll.plot_multitrack(multitrack: Multitrack, axes: Optional[Sequence[matplotlib.axes._axes.Axes]], mode: str = 'separate', track_label: str = 'name', preset: str = 'full', cmaps: Optional[Sequence[str]] = None, xtick: str = 'auto', ytick: str = 'octave', xticklabel: bool = True, yticklabel: str = 'auto', tick_loc: Sequence[str] = ('bottom', 'left'), tick_direction: str = 'in', label: str = 'both', grid_axis: str = 'both', grid_linestyle: str = ':', grid_linewidth: float = 0.5, **kwargs) → List[matplotlib.axes._axes.Axes]
```

Plot the multitrack.

Parameters

- **multitrack** (`pypianoroll.Multitrack`) – Multitrack to plot.
- **axes** (sequence of `matplotlib.axes.Axes`) – Axes to plot the tracks on.
- **mode** (`{'separate', 'blended', 'hybrid'}`) – Plotting strategy for visualizing multiple tracks. For ‘separate’ mode, plot each track separately. For ‘blended’, blend and plot the pianoroll as a colored image. For ‘hybrid’ mode, drum tracks are blended into a ‘Drums’ track and all other tracks are blended into an ‘Others’ track. Defaults to ‘separate’.
- **track_label** (`{'name', 'program', 'family', 'off'}`) – Track label format. When *mode* is ‘hybrid’, all options other than ‘off’ will label the two track with ‘Drums’ and ‘Others’.
- **preset** (`{'full', 'frame', 'plain'}`) – Preset theme to use. For ‘full’ preset, ticks, grid and labels are on. For ‘frame’ preset, ticks and grid are both off. For ‘plain’ preset, the x- and y-axis are both off. Defaults to ‘full’.
- **cmaps** (*tuple or list*) – Colormaps. Will be passed to `matplotlib.pyplot.imshow()`. Only effective when *pianoroll* is 2D. Defaults to ‘Blues’. If *mode* is ‘separate’, defaults to (‘Blues’, ‘Oranges’, ‘Greens’, ‘Reds’, ‘Purples’, ‘Greys’). If *mode* is ‘blended’, defaults to (‘hsv’). If *mode* is ‘hybrid’, defaults to (‘Blues’, ‘Greens’).
- ****kwargs** – Keyword arguments to pass to `pypianoroll.plot_pianoroll()`.

Returns (Created) list of Axes objects.

Return type list of `matplotlib.axes.Axes`

```
pypianoroll.plot_pianoroll(ax: matplotlib.axes._axes.Axes, pianoroll: numpy.ndarray, is_drum: bool = False, resolution: Optional[int] = None, downbeats: Optional[Sequence[int]] = None, preset: str = 'full', cmap: str = 'Blues', xtick: str = 'auto', ytick: str = 'octave', xticklabel: bool = True, yticklabel: str = 'auto', tick_loc: Sequence[str] = ('bottom', 'left'), tick_direction: str = 'in', label: str = 'both', grid_axis: str = 'both', grid_linestyle: str = ':', grid_linewidth: float = 0.5, **kwargs)
```

Plot a piano roll.

Parameters

- **ax** (`matplotlib.axes.Axes`) – Axes to plot the piano roll on.
- **pianoroll** (`ndarray, shape=(?, 128), (?, 128, 3) or (?, 128, 4)`) – Piano roll to plot. For a 3D piano-roll array, the last axis can be either RGB or RGBA.
- **is_drum** (`bool`) – Whether it is a percussion track. Defaults to False.
- **resolution** (`int`) – Time steps per quarter note. Required if *xtick* is ‘beat’.
- **downbeats** (`list`) – Boolean array that indicates whether the time step contains a downbeat (i.e., the first time step of a bar).

- **preset** (`{'full', 'frame', 'plain'}`) – Preset theme. For ‘full’ preset, ticks, grid and labels are on. For ‘frame’ preset, ticks and grid are both off. For ‘plain’ preset, the x- and y-axis are both off. Defaults to ‘full’.
- **cmap** (`str` or `matplotlib.colors.Colormap`) – Colormap. Will be passed to `matplotlib.pyplot.imshow()`. Only effective when *pianoroll* is 2D. Defaults to ‘Blues’.
- **xtick** (`{'auto', 'beat', 'step', 'off'}`) – Tick format for the x-axis. For ‘auto’ mode, set to ‘beat’ if *resolution* is given, otherwise set to ‘step’. Defaults to ‘auto’.
- **ytick** (`{'octave', 'pitch', 'off'}`) – Tick format for the y-axis. Defaults to ‘octave’.
- **xticklabel** (`bool`) – Whether to add tick labels along the x-axis.
- **yticklabel** (`{'auto', 'name', 'number', 'off'}`) – Tick label format for the y-axis. For ‘name’ mode, use pitch name as tick labels. For ‘number’ mode, use pitch number. For ‘auto’ mode, set to ‘name’ if *ytick* is ‘octave’ and ‘number’ if *ytick* is ‘pitch’. Defaults to ‘auto’.
- **tick_loc** (`sequence of {'bottom', 'top', 'left', 'right'}`) – Tick locations. Defaults to (`'bottom', 'left'`).
- **tick_direction** (`{'in', 'out', 'inout'}`) – Tick direction. Defaults to ‘in’.
- **label** (`{'x', 'y', 'both', 'off'}`) – Whether to add labels to x- and y-axes. Defaults to ‘both’.
- **grid_axis** (`{'x', 'y', 'both', 'off'}`) – Whether to add grids to the x- and y-axes. Defaults to ‘both’.
- **grid_linestyle** (`str`) – Grid line style. Will be passed to `matplotlib.axes.Axes.grid()`.
- **grid_linewidth** (`float`) – Grid line width. Will be passed to `matplotlib.axes.Axes.grid()`.
- ****kwargs** – Keyword arguments to be passed to `matplotlib.axes.Axes.imshow()`.

`pypianoroll.plot_track` (*track*: `Track`, *ax*: `Optional[matplotlib.axes._axes.Axes] = None`, ****kwargs**)
→ `matplotlib.axes._axes.Axes`

Plot a track.

Parameters

- **track** (`pypianoroll.Track`) – Track to plot.
- **ax** (`matplotlib.axes.Axes`) – Axes to plot the piano roll on. Defaults to call `plt.gca()`.
- ****kwargs** – Keyword arguments to pass to `pypianoroll.plot_pianoroll()`.

Returns (Created) Axes object.

Return type `matplotlib.axes.Axes`

`pypianoroll.polyphonic_rate` (*pianoroll*: `numpy.ndarray`, *threshold*: `float = 2`) → `float`

Return the ratio of time steps where multiple pitches are on.

The polyphony rate is defined as the ratio of the number of time steps where multiple pitches are on to the total number of time steps. Drum tracks are ignored. Return NaN if song length is zero. This metric is used in [1],

where it is called *polyphonicity*.

$$\text{polyphony_rate} = \frac{\#(\text{time_steps_where_multiple_pitches_are_on})}{\#(\text{time_steps})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **threshold** (*int*) – Threshold of number of pitches to count into the numerator.

Returns Polyphony rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.qualified_note_rate` (*pianoroll: numpy.ndarray, threshold: float = 2*) → float

Return the ratio of the number of the qualified notes.

The qualified note rate is defined as the ratio of the number of qualified notes (notes longer than *threshold*, in time steps) to the total number of notes. Return NaN if no note is found.

$$\text{qualified_note_rate} = \frac{\#(\text{notes_longer_than_the_threshold})}{\#(\text{notes})}$$

Parameters

- **pianoroll** (*ndarray*) – Piano roll to evaluate.
- **threshold** (*int*) – Threshold of note length to count into the numerator.

Returns Qualified note rate.

Return type float

References

1. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, and Yi-Hsuan Yang, “MuseGAN: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment,” in Proceedings of the 32nd AAAI Conference on Artificial Intelligence (AAAI), 2018.

`pypianoroll.read` (*path: Union[str, pathlib.Path], **kwargs*) → `pypianoroll.multitrack.Multitrack`

Read a MIDI file into a Multitrack object.

Parameters

- **path** (*str or Path*) – Path to the file to read.
- ****kwargs** – Keyword arguments to pass to `pypianoroll.from_pretty_midi()`.

See also:

`pypianoroll.write()` Write a Multitrack object to a MIDI file.

`pypianoroll.load()` Load a NPZ file into a Multitrack object.

Return type `pretty_midi.PrettyMIDI`

Notes

- Tempo changes are not supported.
- Time signature changes are not supported.
- The velocities of the converted piano rolls will be clipped to [0, 127].
- Adjacent nonzero values of the same pitch will be considered a single note with their mean as its velocity.

`pypianoroll.tonal_distance` (*pianoroll_1*: `numpy.ndarray`, *pianoroll_2*: `numpy.ndarray`, *resolution*: `int`, *radii*: `Sequence[float] = (1.0, 1.0, 0.5)`) → `float`

Return the tonal distance [1] between the two input piano rolls.

Parameters

- **pianoroll_1** (`ndarray`) – First piano roll to evaluate.
- **pianoroll_2** (`ndarray`) – Second piano roll to evaluate.
- **resolution** (`int`) – Time steps per beat.
- **radii** (*tuple of float*) – Radii of the three tonal circles (see Equation 3 in [1]).

References

1. Christopher Harte, Mark Sandler, and Martin Gasser, “Detecting harmonic change in musical audio,” in Proceedings of the 1st ACM workshop on Audio and music computing multimedia, 2006.

`pypianoroll.transpose` (*obj*: `_MultitrackOrTrack`, *semitone*: `int`) → `_MultitrackOrTrack`

Transpose the piano roll(s) by a number of semitones.

Positive values are for a higher key, while negative values are for a lower key. Drum tracks are ignored.

Parameters

- **obj** (`pypianoroll.Multitrack` or `pypianoroll.Track`) – Object to transpose.
- **semitone** (`int`) – Number of semitones to transpose. A positive value raises the pitches, while a negative value lowers the pitches.

Returns

Return type Object itself.

`pypianoroll.trim` (*obj*: `_MultitrackOrTrack`, *start*: `Optional[int] = None`, *end*: `Optional[int] = None`) → `_MultitrackOrTrack`

Trim the trailing silences of the piano roll(s).

Parameters

- **obj** (`pypianoroll.Multitrack` or `pypianoroll.Track`) – Object to trim.
- **start** (`int`, *optional*) – Start time. Defaults to 0.
- **end** (`int`, *optional*) – End time. Defaults to active length.

Returns

Return type Object itself.

`pypianoroll.write(path: str, multitrack: Multitrack)`

Write a Multitrack object to a MIDI file.

Parameters

- **path** (*str*) – Path to write the file.
- **multitrack** (*pypianoroll.Multitrack*) – Multitrack to save.

See also:

`pypianoroll.read()` Read a MIDI file into a Multitrack object.

`pypianoroll.save()` Save a Multitrack object to a NPZ file.

PYTHON MODULE INDEX

p

`pypianoroll`, [27](#)

A

`append()` (*py pianoroll.Multitrack method*), 28

B

`binarize()` (*in module py pianoroll*), 36
`binarize()` (*py pianoroll.Multitrack method*), 29
`binarize()` (*py pianoroll.Track method*), 34
`BinaryTrack` (*class in py pianoroll*), 27
`blend()` (*py pianoroll.Multitrack method*), 29

C

`clip()` (*in module py pianoroll*), 36
`clip()` (*py pianoroll.Multitrack method*), 29
`clip()` (*py pianoroll.StandardTrack method*), 33
`copy()` (*py pianoroll.Multitrack method*), 29
`copy()` (*py pianoroll.Track method*), 34
`count_downbeat()` (*py pianoroll.Multitrack method*), 29

D

`downbeat` (*py pianoroll.Multitrack attribute*), 28
`drum_in_pattern_rate()` (*in module py pianoroll*), 36

E

`empty_beat_rate()` (*in module py pianoroll*), 37

F

`from_pretty_midi()` (*in module py pianoroll*), 37

G

`get_downbeat_steps()` (*py pianoroll.Multitrack method*), 29
`get_length()` (*py pianoroll.Multitrack method*), 30
`get_length()` (*py pianoroll.Track method*), 34
`get_max_length()` (*py pianoroll.Multitrack method*), 30

I

`in_scale_rate()` (*in module py pianoroll*), 38
`is_drum` (*py pianoroll.BinaryTrack attribute*), 27

`is_drum` (*py pianoroll.StandardTrack attribute*), 33
`is_drum` (*py pianoroll.Track attribute*), 34
`is_valid()` (*py pianoroll.Multitrack method*), 30
`is_valid()` (*py pianoroll.Track method*), 34
`is_valid_type()` (*py pianoroll.Multitrack method*), 30
`is_valid_type()` (*py pianoroll.Track method*), 34

L

`load()` (*in module py pianoroll*), 38

M

`module`
 py pianoroll, 27
`Multitrack` (*class in py pianoroll*), 28

N

`n_pitch_classes_used()` (*in module py pianoroll*), 38
`n_pitches_used()` (*in module py pianoroll*), 39
`name` (*py pianoroll.BinaryTrack attribute*), 27
`name` (*py pianoroll.Multitrack attribute*), 28
`name` (*py pianoroll.StandardTrack attribute*), 33
`name` (*py pianoroll.Track attribute*), 33

P

`pad()` (*in module py pianoroll*), 39
`pad()` (*py pianoroll.Multitrack method*), 30
`pad()` (*py pianoroll.Track method*), 35
`pad_to_multiple()` (*in module py pianoroll*), 39
`pad_to_multiple()` (*py pianoroll.Multitrack method*), 30
`pad_to_multiple()` (*py pianoroll.Track method*), 35
`pad_to_same()` (*in module py pianoroll*), 40
`pad_to_same()` (*py pianoroll.Multitrack method*), 31
`pianoroll` (*py pianoroll.BinaryTrack attribute*), 28
`pianoroll` (*py pianoroll.StandardTrack attribute*), 33
`pianoroll` (*py pianoroll.Track attribute*), 34
`pitch_range()` (*in module py pianoroll*), 40
`pitch_range_tuple()` (*in module py pianoroll*), 40
`plot()` (*in module py pianoroll*), 40
`plot()` (*py pianoroll.Multitrack method*), 31

`plot()` (*pypianoroll.Track* method), 35
`plot_multitrack()` (*in module pypianoroll*), 40
`plot_pianoroll()` (*in module pypianoroll*), 41
`plot_track()` (*in module pypianoroll*), 42
`polyphonic_rate()` (*in module pypianoroll*), 42
`program` (*pypianoroll.BinaryTrack* attribute), 27
`program` (*pypianoroll.StandardTrack* attribute), 33
`program` (*pypianoroll.Track* attribute), 34
`pypianoroll`
 module, 27

Q

`qualified_note_rate()` (*in module pypianoroll*), 43

R

`read()` (*in module pypianoroll*), 43
`remove_empty()` (*pypianoroll.Multitrack* method), 31
`resolution` (*pypianoroll.Multitrack* attribute), 28

S

`save()` (*in module pypianoroll*), 43
`save()` (*pypianoroll.Multitrack* method), 31
`set_nonzeros()` (*in module pypianoroll*), 44
`set_nonzeros()` (*pypianoroll.BinaryTrack* method), 28
`set_nonzeros()` (*pypianoroll.Multitrack* method), 31
`set_nonzeros()` (*pypianoroll.StandardTrack* method), 33
`set_resolution()` (*in module pypianoroll*), 44
`set_resolution()` (*pypianoroll.Multitrack* method), 31
`stack()` (*pypianoroll.Multitrack* method), 32
`standardize()` (*pypianoroll.Track* method), 35
`StandardTrack` (*class in pypianoroll*), 33

T

`tempo` (*pypianoroll.Multitrack* attribute), 28
`to_pretty_midi()` (*in module pypianoroll*), 44
`to_pretty_midi()` (*pypianoroll.Multitrack* method), 32
`tonal_distance()` (*in module pypianoroll*), 45
`Track` (*class in pypianoroll*), 33
`tracks` (*pypianoroll.Multitrack* attribute), 28
`transpose()` (*in module pypianoroll*), 45
`transpose()` (*pypianoroll.Multitrack* method), 32
`transpose()` (*pypianoroll.Track* method), 35
`trim()` (*in module pypianoroll*), 45
`trim()` (*pypianoroll.Multitrack* method), 32
`trim()` (*pypianoroll.Track* method), 35

V

`validate()` (*pypianoroll.Multitrack* method), 32

`validate()` (*pypianoroll.Track* method), 36
`validate_type()` (*pypianoroll.Multitrack* method), 32
`validate_type()` (*pypianoroll.Track* method), 36

W

`write()` (*in module pypianoroll*), 45
`write()` (*pypianoroll.Multitrack* method), 33