WrightTools Documentation

Release 0.4.4

WrightTools Developers

Feb 24, 2022

CONTENTS

`attune` is a package for representing, generating, and updating tuning curves.

PyPI	
version	
conda-forge	
version	
Read the Docs	
stable	
latest	

CHAPTER

ONE

CONTENTS

1.1 Installation

`attune` requires Python 3.6 or newer.

1.1.1 conda-forge

Conda is a multilingual package/environment manager. It seamlessly handles non-Python library dependencies which many scientific Python tools rely upon. Conda is recommended, especially for Windows users. If you don't have Python yet, start by installing Anaconda or miniconda.

conda-forge is a community-driven conda channel. conda-forge contains an attune feedstock.

```
conda config --add channels conda-forge
conda install attune
```

To upgrade:

```
conda update attune
```

1.1.2 pip

pip is Python's official package manager. attune is hosted on PyPI.

```
pip install attune
```

To upgrade:

```
pip install attune --upgrade
```

1.2 Contributing

Thank you so much for contributing to `attune`! We really appreciate your help.

If you have any questions at all, please either open an issue on GitHub or email a `attune` maintainer. The current maintainers can always be found in CONTRIBUTORS.

1.2.1 Preparing

1. fork the attune repository (if you have push access to the main repository you can skip this step)
2. clone `attune` to your machine:

```
$ git clone <your fork>
```

3. in the cloned directory (note, to install to system python, you may need to use `sudo` for this command):

```
$ pip install -e .[dev]
```

4. run tests

```
$ pytest
```

1.2.2 Contributing

1. **ensure that the changes you intend to make have corresponding issues on GitHub**
   a) if you aren't sure how to break your ideas into atomic issues, feel free to open a discussion issue
   b) looking for low-hanging fruit? check out the help wanted label for beginner-friendly issues

```
$ # Create the branch, including remote
$ git branch <your branch> --set-upstream-to origin origin/<your branch>
$ git checkout <your branch> # Switch to the newly created branch
```

2. run all tests to ensure that nothing is broken right off the start

```
$ pytest
```

3. make your changes, commiting often

```
$ git status # See which files you have changed/added
$ git diff # See changes since your last commit
$ git add <files you wish to commit>
$ git commit -m "Description of changes" -m "More detail if needed"
```

4. mark your issues as resolved (within your commit message):

```
$ git commit -m "added crazy colormap (resolves #99)"
```

   a. If your commit is related to an issue, but does not resolve it, use `addresses #99` in the commit message

5. if appropriate, add tests that address your changes (if you just fixed a bug, it is strongly reccomended that you add a test so that the bug cannot come back unanounced)

6. once you are done with your changes, run your code through flake8 and pydocstyle

```
$ flake8 file.py
$ pydocstyle file.py
```

7. rerun tests
8. add yourself to CONTRIBUTORS
9. push your changes to the remote branch (github)

```
$ git pull # make sure your branch is up to date
$ git push
```

10. make a pull request to the master branch
11. communicate with the maintainers in your pull request, assuming any further work needs to be done
12. celebrate!

1.2.3 Style

Internally we use the following abbreviations:

WrightTools `import WrightTools as wt`

Matplotlib `import matplotlib as mpl`

Pyplot `from matplotlib import pyplot as plt`

NumPy `import numpy as np`

`attune` follows pep8, with the following modifications:

1. Maximum line length from 79 characters to 99 characters.

`attune` also folows numpy Docstring Convention, which is a set of adjustments to pep257. `attune` additionally ignores one guideline:

1. `attune` does not require all magic methods (e.g. `__add__`) to have a docstring.
    a) It remains encourged to add a docstring if there is any ambiguity of the meaning.

We use flake8 for automated code style enforcement, and pydocstyle for automated docstring style checking.

```
$ # These will check the whole directory (recursively)
$ flake8
$ pydocstyle
```

Consider using black for automated code corrections. Black is an opinionated code formatter for unambiguous standardization.

```
$ git commit -m "Describe changes"
$ black file.py
$ git diff # review changes
$ git add file.py
$ git commit -m "black style fixes"
```

We also provide a configuration to use git hooks to automatically apply `black` style to edited files. This hook can be installed using `pre-commit`:

```
$ pre-commit install
```

When committing, it will automatically apply the style, and prevent the commit from completing if changes are made. If that is the case, simply re-add the changed files and then commit again. This prevents noisy commit logs with changes that are purely style conformity.

1.3 API

1.3.1 attune.Arrangement

class attune.**Arrangement**(*name: str*, *tunes: Dict[str, Union[attune._discrete_tune.DiscreteTune, attune._tune.Tune, dict]]*)

Bases: object

__init__(*name: str*, *tunes: Dict[str, Union[attune._discrete_tune.DiscreteTune, attune._tune.Tune, dict]]*)

Arrangement of several Tunes to form one cohesive set.

Tunes may represent either motors or other arrangements, however semantic meaning is provided by the Instrument which contains the Arrangement, to the arrangement, they are all string keys mapped to tunes.

All tunes *must* have the same independent units, and must overlap.

Parameters

- **name** (*str*) – A name for this Arrangement, used to identify this Arrangement from other Arrangements which may depend on this one.
- **tunes** (*Dict[str, Tune]*) – Mapping of names to Tune objects which compose the Arrangement

as_dict()

Dictionary representation of the Arrangement

property ind_max

The maximum independant (input) value for this arrangement.

property ind_min

The minimum independant (input) value for this arrangement.

property independent

Returns a 1-dimensional numpy array with the set of all unique independent points.

Points closer together than 1/1000th of the total dynamic range are considered identical.

Only returns points within range of all tunes.

items()

Return the names and tunes in the arrangment.

keys()

Return the names of the tunes in the arrangment.

property name

The name of the arrangement.

property tunes

The tunes in the arrangement.

values()

Return the tunes in the arrangment.

1.3.2 attune.Instrument

class attune.**Instrument**(*arrangements: Dict[str, Union[attune._arrangement.Arrangement, dict]], setables: Optional[Dict[str, Optional[Union[attune._setable.Setable, dict]]]] = None, *, name: Optional[str] = None, transition: Optional[Union[attune._transition.Transition, dict]] = None, load: Optional[float] = None*)

Bases: object

__init__(*arrangements: Dict[str, Union[attune._arrangement.Arrangement, dict]], setables: Optional[Dict[str, Optional[Union[attune._setable.Setable, dict]]]] = None, *, name: Optional[str] = None, transition: Optional[Union[attune._transition.Transition, dict]] = None, load: Optional[float] = None*)

Representation of a system of arrangements for an instrument.

Parameters

- **arrangements** (*Dict[str, Union[Arrangement, dict]]*) – Dictionary of arrangements in the instrument
- **setables** (*Dict[str, Optional[Union[Setable, dict]]]*) – Settable values in the instrument
- **name** (*Optional[str]*) – The name of the instrument, used to store/retrieve the instrument.
- **transition** (*Optional[Union[Transition, dict]]*) – The operation which creates this instrument. If not given, will be "create".
- **load** (*Optional[float]*) – POSIX timestamp of the tune when retrieved from the store. Ignore for instruments not retrieved from the store.

property arrangements

The arrangements associated with this instrument.

as_dict()

Dictionary representation for this Instrument.

property load

The POSIX timestamp for when this instrument was created, if it was stored.

property name

The name of the instrument.

This is the key that is used to store/retrieve the instrument.

save(*file*)

Save the JSON representation into an open file.

property setables

The setables associated with this instrument.

property transition

The transition operation that generated this instrument.

1.3.3 attune.Note

class attune.**Note**(*setables: Dict[str, attune._setable.Setable]*, *setable_positions: Dict[str, Union[float, str]]*, *arrangement_name: str*)

Bases: object

__init__(*setables: Dict[str, attune._setable.Setable]*, *setable_positions: Dict[str, Union[float, str]]*, *arrangement_name: str*)

A particular set of motor positions.

Parameters

- **setables** (*Dict[str, Setable]*) – The setables represented in the note
- **setable_positions** (*Dict[str, Union[float, str]]*) – Mapping of setable keys to positions
- **arrangement_name** (*str*) – The name of the arrangement used to make this note

items()

Items in the Note.

keys()

Settable keys in the Note.

values()

Settable values.

1.3.4 attune.Setable

class attune.**Setable**(*name: str*, *default: Optional[Union[str, float]] = None*, ***kwargs*)

Bases: object

__init__(*name: str*, *default: Optional[Union[str, float]] = None*, ***kwargs*)

Setable object representation.

Parameters **name** (*str*) – The key for this setable

as_dict()

Representation as a JSON encodable dictionary.

1.3.5 attune.Tune

class attune.**Tune**(*independent*, *dependent*, ***, *dep_units=None*, ***kwargs*)

Bases: object

__init__(*independent*, *dependent*, ***, *dep_units=None*, ***kwargs*)

A Tune which maps one set of inputs to associated output points.

Currently all tunes are assumed to have "nm" as their independent array units. All mappings are linear interpolations

Parameters

- **independent** (*1D array-like*) – The independent axis for input values to be mapped. Must be the same shape as dependent.
- **dependent** (*1D array-like*) – The depending axis for the mapping. Must be the same shape as independent.

- **dep_units** (*str (optional)*) – Units for the dependent axis
- **Note** (*kwargs are provided to make the serialized dictionary with ind_units*) –
- **object** (*easy to initialize into a Tune*) –
- **ignored.** (*but are currently*) –

as_dict()
Serialize this Tune as a python dictionary.

property dep_units
The units of the dependent (output) values.

property dependent
The dependent (output) values for the tune points.

property ind_max
The maximum independent (input) value for the tune.

property ind_min
The minimum independent (input) value for the tune.

property ind_units
The units of the independent (input) values.

property independent
The independent (input) values for the tune points.

1.3.6 attune.catalog

attune.**catalog**(*full=False*)
Access a catalog of instruments.

By default returns a list of keys available. If full is True, loads each instrument as a dictionary of keys to Instrument objects.

1.3.7 attune.holistic

attune.**holistic**(*, *data*, *channels*, *arrangement*, *tunes*, *instrument*, *spectral_axis=- 1*, *level=False*, *gtol=0.01*, *autosave=True*, *save_directory=None*, ***spline_kwargs*)
Workup multi-dependent tuning data.

Note: At this time, this function expects 2-dimensional motor space. The algorithm should generalize to N-dimensional motor space, however this is untested and plotting likely will fail.

Parameters

- **data** (*WrightTools.Data*) – The data object to process.
- **channels** (*WrightTools.data.Channel or int or str or 2-tuple*) – If singular: the spectral axis, from which the 0th and 1st moments will be taken to obtain amplitudes and centers. In this case, *spectral_axis* determines which axis is used to obtain the moments. If a tuple: (amplitudes, centers), then these channels will be used directly.
- **tunes** (*iterable of str*) – Names of the tunes to modify in the instrument, in the same order as the axes of *data*. Must not be DiscreteTunes.

- **instrument** (attune.Instrument) – Instrument object to modify. Setpoints are determined from the instrument.
- **Parameters** (*Keyword*) –
- **----------** –
- **spectral_axis** (*WrightTools.data.Axis or int or str (default -1)*) – The axis along which to take moments. Only applies if a single channel is given.
- **level** (*bool (default False)*) – Toggle leveling data. If two channels are given, only the amplitudes are leveled. If a single channel is given, leveling occurs before taking the moments.
- **gtol** (*float (default 0.01)*) – Global tolerance for rejecting noise level relative to the global maximum.
- **autosave** (*bool (default True)*) – Toggles saving of instrument files and images.
- **save_directory** (*Path-like (Defaults to current working directory)*) – Specify where to save files.
- ****spline_kwargs** – Extra arguments to pass to spline creation (e.g. s=0, k=1 for linear interpolation)

1.3.8 attune.intensity

attune.**intensity**(*, *data*, *channel*, *arrangement*, *tune*, *instrument=None*, *level=False*, *gtol=0.01*, *ltol=0.1*, *autosave=True*, *save_directory=None*, ***spline_kwargs*)

Workup a generic intensity plot for a single dependent.

Parameters

- **data** (*wt.data.Data*) – should be in (setpoint, dependent)
- **channel** (*wt.data.Channel or int or str*) – channel to process
- **arrangement** (*str*) – name of the arrangement to modify in the instrument
- **tune** (*str*) – name of the tune to modify in the instrument
- **instrument** (attune.Instrument, *optional*) – instrument object to modify (Default None: make a new instrument)
- **level** (*bool, optional*) – toggle leveling data (Defalts to False)
- **gtol** (*float, optional*) – global tolerance for rejecting noise level relative to global maximum
- **ltol** (*float, optional*) – local tolerance for rejecting data relative to slice maximum
- **autosave** (*bool, optional*) – toggles saving of instrument file and images (Defaults to True)
- **save_directory** (*Path-like*) – where to save (Defaults to current working directory)
- ****spline_kwargs** (*optional*) – extra arguments to pass to spline creation (e.g. s=0, k=1 for linear interpolation)

Returns New instrument object.

Return type *attune.Instrument*

1.3.9 attune.load

attune.**load**(*name: str*, *time=None*, *reverse: bool = True*)

Load an istrument of the given name.

Parameters

- **name** (*str*) – The key of the instrument to load
- **time** (*str, datetime, optional*) – The time for which to load the instrument. Allows loading previous instruments in the catalog. Allows for some natural language descriptions e.g. "5 minutes ago". By default uses the current timestamp.
- **reverse** (*boolean, optional*) – Direction to search, by default looks for a previous curve. If given as False, looks forward in time from the given timestamp.

1.3.10 attune.map_ind_limits

attune.**map_ind_limits**(*instrument*, *arrangement*, *tune*, *min*, *max*, *units=None*)

Map the independent values of a tune onto new limits.

The resultant tune will have the same number of points as the original, evenly spaced between `min` and `max`.

Parameters

- **instrument** (Instrument) – The instrument object to alter.
- **arrangement** (*str*) – The name of the arrangemnt to alter.
- **tune** (*str*) – The name of the tune to alter.
- **min** (*float*) – The new minimum setpoint.
- **max** (*float*) – The new maximum setpoint.
- **units** (*Optional[str]*) – The units of the new setpoints, will be converted to the original units. If not given, original units are assumed.

Returns The instrument with the tune remapped to new setpoints.

Return type *Instrument*

1.3.11 attune.map_ind_points

attune.**map_ind_points**(*instrument*, *arrangement*, *tune*, *setpoints*, *units=None*)

Map the independent values of a tune onto new setpoints.

Parameters

- **instrument** (Instrument) – The instrument object to alter.
- **arrangement** (*str*) – The name of the arrangemnt to alter.
- **tune** (*str*) – The name of the tune to alter.
- **setpoints** (*array-like*) – The new setpoints to map the tune to.
- **units** (*Optional[str]*) – The units of the new setpoints, will be converted to the original units. If not given, original units are assumed.

Returns The instrument with the tune remapped to new setpoints.

Return type *Instrument*

1.3.12 attune.offset_by

attune.**offset_by**(*instrument*, *arrangement*, *tune*, *amount*, *amount_units=None*)

1.3.13 attune.offset_to

attune.**offset_to**(*instrument*, *arrangement*, *tune*, *destination*, *setpoint*, *destination_units=None*, *setpoint_units=None*)

1.3.14 attune.open

attune.**open**(*path*, *, *load=False*)

Open an instrument stored in a JSON file.

Parameters

- **path** (*PathLike or FileLike*) – The path to a file which contains an instrument
- **load** (*datetime*) – Allows this method to be used for loading by providing its associated store time Should generally be avoided when used directly

Returns The instrument that was stored in the file

Return type *Instrument*

1.3.15 attune.restore

attune.**restore**(*name*, *time*, *reverse=True*)

Restore a previously applied instrument.

Parameters

- **name** (*str*) – The key of the instrument to load
- **time** (*str, datetime*) – The time for which to load the instrument. Allows loading previous instruments in the catalog. Allows for some natural language descriptions e.g. "5 minutes ago". By default uses the current timestamp.
- **reverse** (*boolean, optional*) – Direction to search, by default looks for a previous curve. If given as False, looks forward in time from the given timestamp.

1.3.16 attune.setpoint

attune.**setpoint**(*, *data*, *channel*, *arrangement*, *tune*, *instrument=None*, *autosave=True*, *save_directory=None*, ***spline_kwargs*)

Workup a generic setpoint plot for a single tune.

Parameters

- **data** (*wt.data.Data object*) – should be in (setpoint, tune)
- **channel** (*wt.data.Channel or int or str*) – channel to process
- **arrangement** (*str*) – name of the arrangement to modify
- **tune** (*str*) – name of the tune to modify in the instrument

- **instrument** (*attune.Curve, optional*) – instrument object to modify (Default None: make a new instrument)
- **autosave** (*bool, optional*) – toggles saving of instrument file and images (Defaults to True)
- **save_directory** (*Path-like*) – where to save (Defaults to current working directory)
- ****spline_kwargs** (*optional*) – extra arguments to pass to spline creation (e.g. s=0, k=1 for linear interpolation)

Returns New instrument object.

Return type attune.Curve

1.3.17 attune.store

attune.**store**(*instrument*, *warn=True*)

Store an instrument into the catalog.

Parameters

- **instrument** (Instrument) – The instrument to store.
- **warn** (*bool*) – Whether or not to warn if the store is equivalent to the current head.

1.3.18 attune.tune_test

attune.**tune_test**(***, *data*, *channel*, *arrangement*, *instrument*, *level=False*, *gtol=0.01*, *ltol=0.1*, *restore_setpoints=True*, *autosave=True*, *save_directory=None*, ***spline_kwargs*)

Workup a Tune Test.

Parameters

- **data** (*wt.data.Data*) – should be in (setpoint, dependent)
- **channel** (*wt.data.Channel or int or str*) – channel to process
- **arrangement** (*str*) – name of the arrangment to modify
- **instrument** (attune.Instrument) – instrument object to modify
- **level** (*bool, optional*) – toggle leveling data (Defalts to False)
- **gtol** (*float, optional*) – global tolerance for rejecting noise level relative to global maximum
- **ltol** (*float, optional*) – local tolerance for rejecting data relative to slice maximum
- **restore_setpoints** (*bool, optional*) – toggles remapping onto original setpoints for each tune (default is True)
- **autosave** (*bool, optional*) – toggles saving of instrument file and images (Defaults to True)
- **save_directory** (*Path-like*) – where to save (Defaults to current working directory)
- ****spline_kwargs** (*optional*) – extra arguments to pass to spline creation (e.g. s=0, k=1 for linear interpolation)

Returns New instrument object.

Return type *attune.Instrument*

1.3.19 attune.undo

attune.**undo**(*instrument*)
Undo one transition.

1.4 Gallery

1.4.1 Simple Curve Plot

```
import attune
import numpy as np

# d0 = attune.Dependent(np.linspace(-5, 5, 20), "One Dependent")
# d1 = attune.Dependent(np.sin(np.linspace(-4, 0, 20)), "Two Dependent")
# s = attune.Setpoints(np.linspace(1300, 1400, 20), "Some Setpoints", "wn")
# c = attune.Curve(s, [d0, d1], "Sample Curve")

# c.plot()
```

Total running time of the script: (0 minutes 0.360 seconds)

CHAPTER

TWO

INDEX

- genindex
- modindex
- search

INDEX

Symbols

A

C

D

H

I

K

L

M

N

O

R

S

T

U

V