
WrightTools Documentation

Release 0.3.1

WrightTools Developers

Aug 19, 2020

CONTENTS

1	Contents	3
1.1	Installation	3
1.2	Contributing	4
1.3	attune API	6
1.4	Gallery	14
2	Index	17
	Python Module Index	19
	Index	21

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

PyPI	
version	
conda-forge	
version	
Read the Docs	
stable	
latest	

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, committing 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 recommended that you add a test so that the bug cannot come back unannounced)
 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 follows [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 encouraged 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 attune API

1.3.1 attune.curve package

Module contents

OPA tuning curves.

<code>Curve(setpoints, dependents, name[, ...])</code>	Central object-type for all OPA tuning curves.
<code>Dependent(positions, name[, units, ...])</code>	Container class for dependent arrays.
<code>Setpoints(positions, name[, units])</code>	
<code>TopasCurve(setpoints, dependents, name[, ...])</code>	

attune.curve.Curve

class `attune.curve.Curve` (*setpoints*, *dependents*, *name*, *interaction=None*, *kind='curve'*, *method=<class 'attune.interpolator_linear.Linear'>*, *subcurve=None*, *source_setpoints=None*, *fmt=None*, ***kwargs*)

Central object-type for all OPA tuning curves.

`__init__` (*setpoints*, *dependents*, *name*, *interaction=None*, *kind='curve'*, *method=<class 'attune.interpolator_linear.Linear'>*, *subcurve=None*, *source_setpoints=None*, *fmt=None*, ***kwargs*)

Create a Curve object.

Parameters

- **setpoints** (*attune.Setpoints*) – The setpoint destinations for the curve.
- **dependents** (*list of Dependent objects*) – Dependent positions for each setpoint.
- **name** (*str*) – Name of curve.
- **kind** (*string*) – The kind of curve (for saving).
- **method** (*interpolation class*) – The interpolation method to use.

Methods

<code>__init__(setpoints, dependents, name[, ...])</code>	Create a Curve object.
<code>coerce_dependents()</code>	Coerce the dependent positions to lie exactly along the interpolation positions.
<code>convert(units, *[, convert_dependents])</code>	Convert the setpoints to new units.
<code>copy()</code>	Copy the curve object.
<code>get_dependent_positions(setpoint[, units, full])</code>	Get the dependent positions for a destination setpoint.

continues on next page

Table 2 – continued from previous page

<code>get_limits([units])</code>	Get the edges of the curve.
<code>get_source_setpoint(setpoint[, units])</code>	Get setpoint of source curve.
<code>interpolate([interpolate_subcurve])</code>	Generate the interpolator object.
<code>map_setpoints(setpoints[, units])</code>	Map the curve onto new tune points using the curve's own interpolation method.
<code>offset_by(dependent, amount)</code>	Offset a dependent by some amount.
<code>offset_to(dependent, destination, setpoint)</code>	Offset a dependent such that it evaluates to <i>destination</i> at <i>setpoint</i> .
<code>plot([autosave, save_path, title])</code>	Plot the curve.
<code>read(filepath[, subcurve])</code>	
<code>rename_dependent(old_name, new_name)</code>	
<code>save([save_directory, plot, verbose, full])</code>	Save the curve.
<code>sort()</code>	

Attributes

<code>dependent_names</code>	Get dependent names.
<code>dependent_units</code>	Get dependent names.

attune.curve.Dependent

class `attune.curve.Dependent` (*positions, name, units=None, differential=False, index=None*)

Container class for dependent arrays.

__init__ (*positions, name, units=None, differential=False, index=None*)

Create a Dependent object.

Parameters

- **positions** (*1D array*) – Dependent positions.
- **name** (*string*) – Name.

Methods

<code>__init__(positions, name[, units, ...])</code>	Create a Dependent object.
<code>convert(units)</code>	

attune.curve.Setpoints

class attune.curve.Setpoints (*positions, name, units=None*)

__init__ (*positions, name, units=None*)
 Initialize self. See help(type(self)) for accurate signature.

Methods

<code>__init__(positions, name[, units])</code>	Initialize self.
<code>convert(units)</code>	

attune.curve.TopasCurve

class attune.curve.TopasCurve (*setpoints, dependents, name, interaction=None, kind='curve', method=<class 'attune.interpolator._linear.Linear'>, subcurve=None, source_setpoints=None, fmt=None, **kwargs*)

__init__ (*setpoints, dependents, name, interaction=None, kind='curve', method=<class 'attune.interpolator._linear.Linear'>, subcurve=None, source_setpoints=None, fmt=None, **kwargs*)
 Create a Curve object.

Parameters

- **setpoints** (*attune.Setpoints*) – The setpoint destinations for the curve.
- **dependents** (*list of Dependent objects*) – Dependent positions for each setpoint.
- **name** (*str*) – Name of curve.
- **kind** (*string*) – The kind of curve (for saving).
- **method** (*interpolation class*) – The interpolation method to use.

Methods

<code>__init__(setpoints, dependents, name[, ...])</code>	Create a Curve object.
<code>coerce_dependents()</code>	Coerce the dependent positions to lie exactly along the interpolation positions.
<code>convert(units, *[, convert_dependents])</code>	Convert the setpoints to new units.
<code>copy()</code>	Copy the curve object.
<code>get_dependent_positions(setpoint[, units, full])</code>	Get the dependent positions for a destination setpoint.
<code>get_limits([units])</code>	Get the edges of the curve.
<code>get_source_setpoint(setpoint[, units])</code>	Get setpoint of source curve.
<code>interpolate([interpolate_subcurve])</code>	Generate the interpolator object.
<code>map_setpoints(setpoints[, units])</code>	Map the curve onto new tune points using the curve's own interpolation method.
<code>offset_by(dependent, amount)</code>	Offset a dependent by some amount.

continues on next page

Table 6 – continued from previous page

<code>offset_to(dependent, destination, setpoint)</code>	Offset a dependent such that it evaluates to <i>destination</i> at <i>setpoint</i> .
<code>plot([autosave, save_path, title])</code>	Plot the curve.
<code>read(filepaths, interaction_string)</code>	Create a curve object from a TOPAS crv file.
<code>read_all(filepaths)</code>	
<code>rename_dependent(old_name, new_name)</code>	
<code>save([save_directory, plot, verbose, full])</code>	Save a curve object.
<code>sort()</code>	

Attributes

<code>dependent_names</code>	Get dependent names.
<code>dependent_units</code>	Get dependent names.

1.3.2 attune.interpolator package

Module contents

<code><i>Interpolator</i>(setpoints, dependent)</code>	
<code><i>Linear</i>(setpoints, dependent)</code>	Linear interpolation.
<code><i>Poly</i>(*args, **kwargs)</code>	
<code><i>Spline</i>(setpoints, dependent)</code>	

attune.interpolator.Interpolator

class `attune.interpolator.Interpolator` (*setpoints, dependent*)

`__init__` (*setpoints, dependent*)
Create an Interpolator object.

Parameters

- **setpoints** (*1D array*) – Setpoints.
- **units** (*string*) – Units.
- **dependents** (*list of WrightTools.tuning.curve.Dependent*) – Dependents.

Methods

<code>__init__(setpoints, dependent)</code>	Create an Interpolator object.
---	--------------------------------

attune.interpolator.Linear

class `attune.interpolator.Linear` (*setpoints, dependent*)
Linear interpolation.

`__init__` (*setpoints, dependent*)
Create an Interpolator object.

Parameters

- **setpoints** (*1D array*) – Setpoints.
- **units** (*string*) – Units.
- **dependents** (*list of WrightTools.tuning.curve.Dependent*) – Dependents.

Methods

<code>__init__(setpoints, dependent)</code>	Create an Interpolator object.
---	--------------------------------

Attributes

<code>function</code>

attune.interpolator.Poly

class `attune.interpolator.Poly` (**args, **kwargs*)

`__init__` (**args, **kwargs*)
Create an Interpolator object.

Parameters

- **setpoints** (*1D array*) – Setpoints.
- **units** (*string*) – Units.
- **dependents** (*list of WrightTools.tuning.curve.Dependent*) – Dependents.

Methods

<code>__init__</code> (*args, **kwargs)	Create an Interpolator object.
---	--------------------------------

Attributes

<code>function</code>	
-----------------------	--

attune.interpolator.Spline

class attune.interpolator.**Spline** (*setpoints, dependent*)

`__init__` (*setpoints, dependent*)
Create an Interpolator object.

Parameters

- **setpoints** (*1D array*) – Setpoints.
- **units** (*string*) – Units.
- **dependents** (*list of WrightTools.tuning.curve.Dependent*) – Dependents.

Methods

<code>__init__</code> (setpoints, dependent)	Create an Interpolator object.
--	--------------------------------

Attributes

<code>function</code>	
-----------------------	--

1.3.3 attune.workup package

Module contents

Methods for processing OPA 800 tuning data.

<code>holistic</code> (data, channels, dependents, curve, *)	Workup multi-dependent tuning data.
<code>intensity</code> (data, channel, dependent[, curve, ...])	Workup a generic intensity plot for a single dependent.
<code>setpoint</code> (data, channel, dependent[, curve, ...])	Workup a generic setpoint plot for a single dependent.
<code>tune_test</code> (data, channel[, curve, level, ...])	Workup a Tune Test.

attune.workup.holistic

`attune.workup.holistic` (*data*, *channels*, *dependents*, *curve*, *, *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.
- **dependents** (*tuple of str*) – Names of the dependents to modify in the curve, in the same order as the axes of *data*.
- **curve** (*attune.Curve*) – Curve object to modify. Setpoints are determined from the curve.
- **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 curve 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)

attune.workup.intensity

`attune.workup.intensity` (*data*, *channel*, *dependent*, *curve=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
- **dependent** (*str*) – name of the dependent to modify in the curve
- **curve** (*attune.Curve*, *optional*) – curve object to modify (Default None: make a new curve)

- **level** (*bool, optional*) – toggle leveling data (Defaults 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 curve 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 curve object.

Return type `attune.Curve`

`attune.workup.setpoint`

`attune.workup.setpoint` (*data, channel, dependent, curve=None, *, autosave=True, save_directory=None, **spline_kwargs*)

Workup a generic setpoint plot for a single dependent.

Parameters

- **data** (*wt.data.Data object*) – should be in (setpoint, dependent)
- **channel** (*wt.data.Channel or int or str*) – channel to process
- **dependent** (*str*) – name of the dependent to modify in the curve
- **curve** (*attune.Curve, optional*) – curve object to modify (Default None: make a new curve)
- **autosave** (*bool, optional*) – toggles saving of curve 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 curve object.

Return type `attune.Curve`

`attune.workup.tune_test`

`attune.workup.tune_test` (*data, channel, curve=None, *, level=False, gtol=0.01, ltol=0.1, 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
- **curve** (*attune.Curve, optional*) – curve object to modify (Default None: make a new curve)

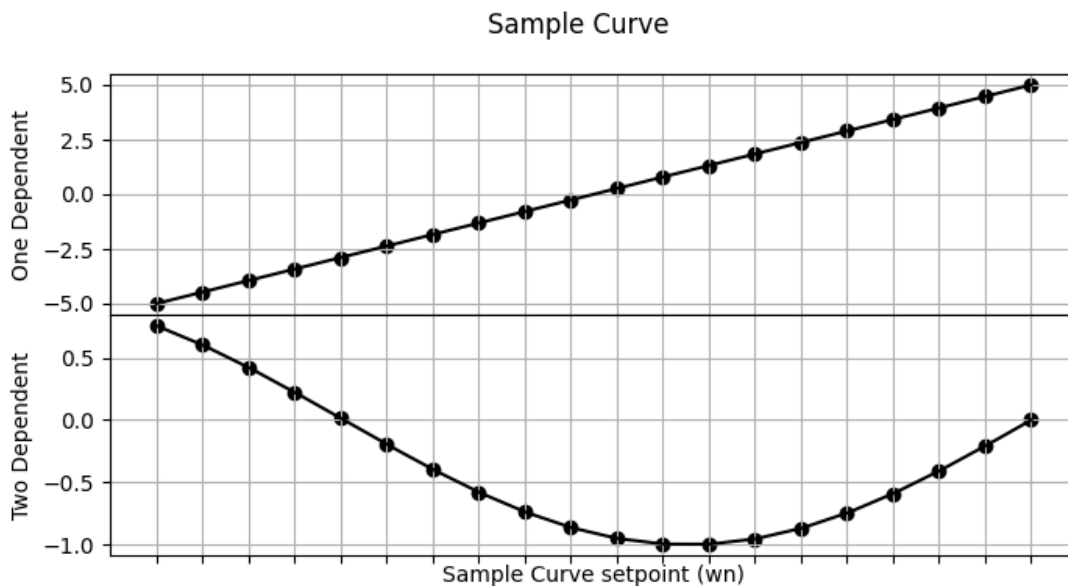
- **level** (*bool, optional*) – toggle leveling data (Defaults to False)
- **gtol** (*float, optional*) – global tolerance for rejecting noise level relative to global maximum
- **ltol** (*float, optional*) – local tolerance for rejecting data relative to slice maximum
- **autosave** (*bool, optional*) – toggles saving of curve 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 curve object.

Return type `attune.Curve`

1.4 Gallery

1.4.1 Simple Curve Plot



Out:

```

/home/docs/checkouts/readthedocs.org/user_builds/attune/envs/latest/lib/python3.6/
↳site-packages/WrightTools/units.py:133: UserWarning: conversion None to wn not
↳valid: returning input
  current_unit, destination_unit
/home/docs/checkouts/readthedocs.org/user_builds/attune/envs/latest/lib/python3.6/
↳site-packages/WrightTools/units.py:133: UserWarning: conversion None to wn not
↳valid: returning input
  current_unit, destination_unit
/home/docs/checkouts/readthedocs.org/user_builds/attune/envs/latest/lib/python3.6/
↳site-packages/attune/curve/_base.py:453: MatplotlibDeprecationWarning: Adding an
↳axes using the same arguments as a previous axes currently reuses the earlier
(continues on next page)
↳instance. In a future version, a new instance will always be created and returned.
↳ Meanwhile, this warning can be suppressed, and the future behavior ensured, by
passing a unique label to each axes instance.

```

(continued from previous page)

```
ax = plt.subplot(lowest_ax_dictionary[curve.name])
/home/docs/checkouts/readthedocs.org/user_builds/attune/envs/latest/lib/python3.6/
↳site-packages/attune/curve/_base.py:465: MatplotlibDeprecationWarning: Adding an
↳axes using the same arguments as a previous axes currently reuses the earlier
↳instance. In a future version, a new instance will always be created and returned.
↳ Meanwhile, this warning can be suppressed, and the future behavior ensured, by
↳ passing a unique label to each axes instance.
ax = plt.subplot(ax)
```

```
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 0.273 seconds)

INDEX

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

`attune.curve`, 6

`attune.interpolator`, 9

`attune.workup`, 11

Symbols

`__init__()` (*attune.curve.Curve method*), 6
`__init__()` (*attune.curve.Dependent method*), 7
`__init__()` (*attune.curve.Setpoints method*), 8
`__init__()` (*attune.curve.TopasCurve method*), 8
`__init__()` (*attune.interpolator.Interpolator method*),
 9
`__init__()` (*attune.interpolator.Linear method*), 10
`__init__()` (*attune.interpolator.Poly method*), 10
`__init__()` (*attune.interpolator.Spline method*), 11

A

`attune.curve`
 module, 6
`attune.interpolator`
 module, 9
`attune.workup`
 module, 11

C

`Curve` (*class in attune.curve*), 6

D

`Dependent` (*class in attune.curve*), 7

H

`holistic()` (*in module attune.workup*), 12

I

`intensity()` (*in module attune.workup*), 12
`Interpolator` (*class in attune.interpolator*), 9

L

`Linear` (*class in attune.interpolator*), 10

M

`module`
 `attune.curve`, 6
 `attune.interpolator`, 9
 `attune.workup`, 11

P

`Poly` (*class in attune.interpolator*), 10

S

`setpoint()` (*in module attune.workup*), 13
`Setpoints` (*class in attune.curve*), 8
`Spline` (*class in attune.interpolator*), 11

T

`TopasCurve` (*class in attune.curve*), 8
`tune_test()` (*in module attune.workup*), 13