## CONTENTS

1 Contents ................................. 3
   1.1 Installation ................................ 3
   1.2 Quick Start ................................ 4
   1.3 Data .................................. 12
   1.4 Collection ............................... 16
   1.5 Artists .................................. 18
   1.6 Units .................................. 29
   1.7 Datasets ................................ 30
   1.8 Contributing ............................. 31
   1.9 The wt5 File Format ..................... 33
   1.10 WrightTools API .......................... 36
   1.11 Gallery ................................ 121
   1.12 Citing WrightTools ....................... 165
   1.13 Alternatives ............................. 166

2 Index ..................................... 167

   Python Module Index ....................... 169

   Index ..................................... 171
WrightTools loads, processes, and plots multidimensional spectroscopy data.

“Multidimensional spectroscopy” (MDS) is a family of diverse analytical techniques that record the response of a material to multiple stimuli—typically multiple ultrafast pulses of light. Due to its diversity and dimensionality, MDS data is challenging to process and visualize. WrightTools is a freely available and openly licensed Python package that is made specifically for multidimensional spectroscopy. It aims to be a core toolkit that is general enough to handle all MDS datasets and processing workloads. Being built for and by MDS practitioners, WrightTools has an intuitive, high-level, object-oriented interface for spectroscopists.

For a more complete introduction to WrightTools, please click on the badge below to read our short three-page paper in the Journal of Open Source Software (https://doi.org/10.21105/joss.01141).
1.1 Installation

WrightTools 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 a WrightTools feedstock.

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

To upgrade:

```
conda update wrighttools
```

1.1.2 pip

pip is Python’s official package manager. WrightTools is hosted on PyPI.

```
pip install wrighttools
```

To upgrade:

```
pip install wrighttools --upgrade
```
1.2 Quick Start

This “quick start” page is designed to introduce a few commonly-used features that you should know immediately as a user of WrightTools. We assume that you have installed WrightTools and that you are somewhat comfortable using Python. If you are brand new to Python, it’s typically useful to run Python within an integrated development environment—our favorite is Spyder.

Each of the following code blocks builds on top of the previous code. Read this document like a series of commands typed into a Python shell. We recommend following along on your own machine.

1.2.1 Create a Data Object

There are many ways to create a WrightTools data object. One strategy is to open an existing wt5 file. When you downloaded WrightTools you also downloaded a few example files. The `WrightTools.datasets` package allows you to easily access the path to these files. Let’s create a data object now:

```python
import WrightTools as wt
# get the path to an example wt5 file
from WrightTools import datasets
p = datasets.wt5.v1p0p1_MoS2_TrEE_movie # just a filepath
# open data object
data = wt.open(p)
```

The data contains some helpful attributes. We can “inspect” these attributes by simply entering them into a Python shell. Let’s do that now:

```python
>>> data.channel_names
['ai0', 'ai1', 'ai2', 'ai3', 'ai4', 'mc']
>>> data.axis_expressions
['w2', 'w1=wm', 'd2']
>>> data.shape
(41, 41, 23)
```

Alternatively, we can use the `print_tree()` method to print out a whole bunch of information at once.

```python
>>> data.print_tree()
_001_dat (/tmp/811qwfvd.wt5)
  axes
    0: w2 (nm) (41, 1, 1)
    1: w1=wm (nm) (1, 41, 1)
    2: d2 (fs) (1, 1, 23)
  constants
  variables
    0: w2 (nm) (41, 1, 1)
    1: w1 (nm) (1, 41, 1)
    2: wm (nm) (1, 41, 1)
    3: d2 (fs) (1, 1, 23)
    4: w3 (nm) (1, 1, 1)
    5: d0 (fs) (1, 1, 1)
    6: d1 (fs) (1, 1, 1)
  channels
    0: ai0 (41, 41, 23)
    1: ai1 (41, 41, 23)
    2: ai2 (41, 41, 23)
    3: ai3 (41, 41, 23)
```

(continues on next page)
Notice that the data object is made out of axes, constants, variables, and channels. All of these are arrays, and they have different shapes and units associated with them. For now, this is all you need to understand about the contents of data objects—read Data when you’re ready to learn more. Next we’ll visualize our data.

### 1.2.2 Visualize Data

WrightTools strives to make data visualization as quick and painless as possible.

Axes, labels, and units are brought along implicitly.

WrightTools offers a few handy ways to quickly visualize a data object, shown below. For more information, see Artists, or check out our Gallery.

**quick1D**

*quick1D()* makes it as easy as possible to visualize a simple 1D slice of our data object. We have to specify an axis to plot along—for this example let’s choose $w_1=wm$. By default, *quick1D()* will plot all possible slices along our chosen axis. Optionally, we can narrow down the number of generated plots by specifying what particular coordinate we are interested in. In this example, we have fully specified all other axes using the at keyword argument, so only one plot will be generated.

```
wt.artists.quick1D(data, 'w1=wm', at={'w2': [2, 'eV'], 'd2': [-100, 'fs']})
```

$\lambda_2 = 621\text{nm}, \tau_2 = -120\text{fs}$
quick2D

`quick2D()` is built with the same goals as `quick1D()`, but for two dimensional representations. This time, we have to specify two axes to plot along—`w1=wm` and `d2`, in this example. Again, we use the `at` keyword argument so only one plot will be generated.

```python
wt.artists.quick2D(data, 'w1=wm', 'd2', at={'w2': [2, 'eV']})
```

\[ \lambda_2 = 621 \text{ nm} \]
interact2D

`WrightTools.artists.interact2D()` uses Matplotlib’s interactive widgets framework to present an interactive graphical interface to a multidimensional data object. You must choose two axes to plot against in the central two-dimensional plot. All other axes are automatically represented as “sliders”, and you can easily manipulate these two explore the dataset in its full dimensionality. See Artists for an example.

### 1.2.3 Process Data

Now let’s actually modify the arrays that make up our data object. Note that the raw data which we imported is not being modified, rather we are modifying the data as copied into our data object.

#### Convert

WrightTools has built in units support. This enables us to easily convert our data object from one unit system to another:

```python
g data.units ('nm', 'nm', 'fs') g data.convert('eV') axis w2 converted from nm to eV axis w1=wm converted from nm to eV g data.units ('eV', 'eV', 'fs')
```

Note that only compatible axes were converted—the trailing axis with units 'fs' was ignored. Want fine control? You can always convert individual axes, e.g. `data.w2.convert('wn')`. For more information see Units.

#### Split

Use `split()` to break your dataset into smaller pieces.

```python
g col = data.split('d2', -100.) split data into 2 pieces along <d2>: 0 : -inf to 0.00 fs (1, 1, 15) 1 : 0.00 to inf fs (1, 1, 8)
```

Note that `split()` accepts axis expressions and unit-aware coordinates, not axis indices.

#### Clip

Use `clip()` to ignore/remove points of a channel outside of a specific range.

```python
data.ai0.clip(min=0.0, max=0.1)
```
Transform

Use `transform()` to choose a different set of axes for your data object.

```
data.ai0.transform('w1=wm', 'w2-wm', 'd2')
```

### 1.2.4 Save Data

It’s easy to save your data objects using WrightTools.

#### Save, Open

Most simply, you can simply save...

```
data.save('my-path.wt5')
```

and then open...

```
data = wt.open('my-path.wt5')
```

You will pick right up at the state where you saved the object (even on different operating systems or machines)!
_001.dat
\[\lambda_2 = 621 \text{ nm}\]
\_001\_dat
\tau_2 = -120\text{fs}
Collections

Collections are containers that can hold multiple data objects. Collections can nest within each-other, much like folders in your computer's file system. Collections can help you store all associated data within a single wt5 file, keeping everything internally organized. Creating collections is easy:

```
>>> collection = wt.Collection(name='test')
```

Filling collections with data objects is easy as well. Again, let's use the WrightTools.datasets package:

```
>>> from WrightTools import datasets
>>> p = datasets.COLORS.v0p2_d1_d2_diagonal
>>> wt.data.from_COLORS(p, parent=collection)

```

Note that we are using from functions instead of `open()`. That's because these aren't wt5 files—they're raw data files output by various instruments. We use the `parent` keyword argument to create these data objects directly inside of our collection. See Data for a complete list of supported file formats.

Much like data objects, collection objects have a method `print_tree()` that prints out a bunch of information:

```
>>> collection.print_tree()
```

Collections can be saved inside of wt5 files, so be aware that `open()` may return a collection or a data object based on the contents of your wt5 file.
1.2.5 Learning More

We hope that this quick start page has been a useful introduction to you. Now it’s time to go forth and process data! If you want to read further, consider the following links:

- more about data objects: Data
- more about collection objects: Collection
- more about WrightTools artists: Artists
- a gallery of figures made using WrightTools (click for source code): Gallery
- a complete list of WrightTools units: Units
- a complete list of attributes and methods of the Data class: Data

1.3 Data

A data object contains your entire n-dimensional dataset, including axes, units, channels, and relevant metadata. Once you have a data object, all of the other capabilities of WrightTools are immediately open to you, including processing, fitting, and plotting tools.

Here we highlight some key features of the data object. For a complete list of methods and attributes, see WrightTools.data.Data in the API docs.

1.3.1 Instantiation

From Supported File Types

WrightTools aims to provide user-friendly ways of creating data directly from common spectroscopy file formats. Here are the formats currently supported.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>API</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrunoldRaman</td>
<td>Files from Brunold lab resonance raman measurements</td>
<td>from_BrunoldRaman()</td>
</tr>
<tr>
<td>Cary</td>
<td>Files from Varian’s Cary® Spectrometers</td>
<td>from_Cary()</td>
</tr>
<tr>
<td>COLORS</td>
<td>Files from Control Lots Of Research in Spectroscopy</td>
<td>from_COLORS()</td>
</tr>
<tr>
<td>JASCO</td>
<td>Files from JASCO optical spectrometers</td>
<td>from_JASCO()</td>
</tr>
<tr>
<td>KENT</td>
<td>Files from “ps control” by Kent Meyer</td>
<td>from_KENT()</td>
</tr>
<tr>
<td>Aramis</td>
<td>Horiba Aramis ngc binary files</td>
<td>from_Aramis()</td>
</tr>
<tr>
<td>Ocean Optics</td>
<td>.scope files from ocean optics spectrometers</td>
<td>from_ocean_optics()</td>
</tr>
<tr>
<td>PyCMDS</td>
<td>Files from PyCMDS</td>
<td>from_PyCMDS()</td>
</tr>
<tr>
<td>Shimadzu</td>
<td>Files from Shimadzu UV-VIS spectrophotometers</td>
<td>from_shimadzu()</td>
</tr>
<tr>
<td>SPCM</td>
<td>Files from Becker &amp; Hickl spcm software</td>
<td>from_spcm()</td>
</tr>
<tr>
<td>Solis</td>
<td>Files from Andor Solis software</td>
<td>from_Solis()</td>
</tr>
<tr>
<td>Tensor 27</td>
<td>Files from Bruker Tensor 27 FT-IR</td>
<td>from_Tensor27()</td>
</tr>
</tbody>
</table>

Is your favorite format missing? It’s easy to add—promise! Check out Contributing.

These functions accept both local and remote (http/ftp) files as well as transparent compression (gz/bz2). Compression detection is based on the file name, and file names for remote links are as appears in the link. Many download links (such as those from osf.io or Google drive) do not include extensions in the download link, and thus will cause Warnings/be unable to accept compressed files. This can often be worked around by adding a variable to the end of the url such as https://osf.io/xxxxx/download?fname=file.csv.gz. Google Drive direct download
links have the form https://drive.google.com/dc?id=XXXXXXXXXXXXXXXXXXXX (i.e. replace open
in the "share" links with dc).

From Bare Arrays

Got bare numpy arrays and dreaming of data? It is possible to create data objects directly, as shown below.

```python
# import
import numpy as np
import WrightTools as wt

# generate arrays for example
def my_resonance(xi, yi, intensity=1, FWHM=500, x0=7000):
    return intensity*(0.5*FWHM)**2/((xi-x0)**2+(0.5*FWHM)**2)

def single(arr, intensity=intensity, FWHM=FWHM, x0=x0):
    return single(xi) * single(yi)

xi = np.linspace(6000, 8000, 75)
yi = np.linspace(6000, 8000, 75)
zi = my_resonance(xi, yi)

# package into data object
data = wt.Data(name='example')
data.create_variable(name='w1', units='wn', values=xi)
data.create_variable(name='w2', units='wn', values=yi)
data.create_channel(name='signal', values=zi)
data.transform('w1', 'w2')
```

Note that NumPy has functions for reading data arrays from text files. Our favorite is `genfromtxt`. Lean on these
functions to read in data from unsupported file formats, then pass in the data as arrays. Of course, if you find yourself
processing a lot of data from a particular file format, consider contributing a new from function to WrightTools.

1.3.2 Structure & Attributes

So what is a data object anyway? To put it simply, `Data` is a collection of `WrightTools.data.Axis` and
`WrightTools.data.Channel` objects. `WrightTools.data.Axis` objects are composed of
`WrightTools.data.Variable` objects.

<table>
<thead>
<tr>
<th>attribute</th>
<th>tuple of...</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>Axis objects</td>
</tr>
<tr>
<td>constants</td>
<td>Constant objects</td>
</tr>
<tr>
<td>channels</td>
<td>Channel objects</td>
</tr>
<tr>
<td>variables</td>
<td>Variable objects</td>
</tr>
</tbody>
</table>

As mentioned above, the axes and channels within data can be accessed within the `data.axes` and `data.channels`
lists. Data also supports natural naming, so axis and channel objects can be accessed directly according
to their name. The natural syntax is recommended, as it tends to result in more readable code.

```python
>>> data.axis_expressions
('w1', 'w2')
>>> data.w2 == data.axes[1]
True
>>> data.channel_names
('signal', 'pyro1', 'pyro2', 'pyro3')
>>> data.pyro2 == data.channels[2]
True
```
The order of axes and channels is arbitrary. However many methods within WrightTools operate on the zero-indexed channel by default. For this reason, you can bring your favorite channel to zero-index using `bring_to_front()`.

**Variable**

The `WrightTools.data.Variable` class holds key coordinates of the data object. One `Variable` instance exists for each recorded independent variable. This includes scanned optomechanical hardware, but also still hardware, and other variables like lab time. A typical data object will have many variables (each a multidimensional array). Variables have the following key attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>LaTeX-formatted label, appropriate for plotting</td>
</tr>
<tr>
<td><code>max()</code></td>
<td>variable maximum</td>
</tr>
<tr>
<td><code>min()</code></td>
<td>variable minimum</td>
</tr>
<tr>
<td><code>natural_name</code></td>
<td>variable name</td>
</tr>
<tr>
<td>units</td>
<td>variable units</td>
</tr>
</tbody>
</table>

**Axis**

The `WrightTools.data.Axis` class defines the coordinates of a data object. Each `Axis` contains an expression, which dictates its relationship with one or more variables. Given 5 variables with names ['w1', 'w2', 'wm', 'd1', 'd2'], example valid expressions include 'w1', 'w1=wm', 'w1+w2', '2*w1', 'd1-d2', and 'wm-w1+w2'. Axes behave like arrays: you can slice into them, view their shape, get a min and max etc. But actually axes do not contain any new array information: they simply refer to the Variable arrays. Axes have the following key attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>label()</code></td>
<td>LaTeX-formatted label, appropriate for plotting</td>
</tr>
<tr>
<td><code>min()</code></td>
<td>coordinates minimum, in current units</td>
</tr>
<tr>
<td><code>max()</code></td>
<td>coordinates maximum, in current units</td>
</tr>
<tr>
<td><code>natural_name</code></td>
<td>axis name</td>
</tr>
<tr>
<td>units</td>
<td>current axis units (change with <code>convert()</code></td>
</tr>
<tr>
<td><code>variables</code></td>
<td>component variables</td>
</tr>
<tr>
<td><code>expression</code></td>
<td>expression</td>
</tr>
</tbody>
</table>

**Constant**

`WrightTools.data.Constant` objects are a special subclass of Axis objects, which is expected to be a single value. Constant adds the value to the label attribute, suitable for titles of plots to identify static values associated with the plot. Note that there is nothing enforcing that the value is actually static: constants still have shapes and can be indexed to get the underlying numpy array.

You can control how this label is generated using the attributes `format_spec` and `round_spec`. `label` uses the python built-in `format`, an thus `format_spec` is a specification as in the Format Specification Mini-Language. Common examples would be “0.2f” or “0.3e” for decimal representation with two digits past the decimal and engineers notation with 3 digits past the decimal, respectively. `round_spec` allows you to control the rounding of your number via the built-in `round()`. For instance, if you want a number rounded to the hundreds position, but represented as an integer, you may use `round_spec=-2; format_spec="0.0f"`.

For example, if you have a constant with value 123.4567 nm, a `format_spec` of 0.3f, and a `round_spec` of 2, you will get a label something like `'$\lambda_1 = 123.460 \, \text{nm}$'`, which will render as \(\lambda_1 = 123.460\,\text{nm}\).
An example of using constants/constant labels for plotting can be found in the gallery: *Custom Figure.*

In addition to the above attributes, constants add:

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
</table>
| format_spec | Format specification for how to represent the value, as in `format()`.
| round_spec | Specify which digit to round to, as in `round()`.
| label | LaTeX formatted label which includes a symbol and the constant value.
| value | The mean (ignoring NaNs) of the evaluated expression.
| std | The standard deviation of the points used to compute the value.

**Channel**

The `WrightTools.data.Channel` class contains the n-dimensional signals. A single data object may contain multiple channels corresponding to different detectors or measurement schemes. Channels have the following key attributes:

<table>
<thead>
<tr>
<th>attribute</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>label</td>
<td>LaTeX-formatted label, appropriate for plotting</td>
</tr>
<tr>
<td>mag()</td>
<td>channel magnitude (furthest deviation from null)</td>
</tr>
<tr>
<td>max()</td>
<td>channel maximum</td>
</tr>
<tr>
<td>min()</td>
<td>channel minimum</td>
</tr>
<tr>
<td>name</td>
<td>channel name</td>
</tr>
<tr>
<td>null</td>
<td>channel null (value of zero signal)</td>
</tr>
<tr>
<td>signed</td>
<td>flag to indicate if channel is signed</td>
</tr>
</tbody>
</table>

**1.3.3 Processing**

**Units aware & interpolation ready**

Experiments are taken over all kinds of dynamic range, with all kinds of units. You might wish to take the difference between a UV-VIS scan taken from 400 to 800 nm, 1 nm steps and a different scan taken from 1.75 to 2.00 eV, 1 meV steps. This can be a huge pain! Even if you converted them to the same unit system, you would still have to deal with the different absolute positions of the two coordinate arrays. `map_variable()` allows you to easily obtain a data object mapped onto a different set of coordinates. WrightTools data objects know all about units, and they are able to use interpolation to map between different absolute coordinates. Here we list some of the capabilities that are enabled by this behavior.

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
<th>gallery</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>heal()</code></td>
<td>use interpolation to guess the value of NaNs within a channel</td>
<td>Heal</td>
</tr>
<tr>
<td><code>join()</code></td>
<td>join together multiple data objects, accounting for dimensionality and overlap</td>
<td>Join</td>
</tr>
<tr>
<td><code>map_variable()</code></td>
<td>re-map data coordinates</td>
<td>Map-Variable</td>
</tr>
</tbody>
</table>
Dimensionality without the cursing

Working with multidimensional data can be intimidating. What axis am I looking at again? Where am I in the other axis? Is this slice unusual, or do they all look like that?

WrightTools tries to make multi-dimensional data easy to work with. The following methods deal directly with dimensionality manipulation.

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
<th>gallery</th>
</tr>
</thead>
<tbody>
<tr>
<td>chop()</td>
<td>chop data into a list of lower dimensional data</td>
<td></td>
</tr>
<tr>
<td>collapse()</td>
<td>destroy one dimension of data using a mathematical strategy</td>
<td></td>
</tr>
<tr>
<td>moment()</td>
<td>destroy one dimension of a channel by taking the nth moment</td>
<td></td>
</tr>
<tr>
<td>split()</td>
<td>split data at a series of coordinates, without reducing dimensionality</td>
<td>Split</td>
</tr>
<tr>
<td>transform()</td>
<td>transform the data on to a new combination of variables as axes</td>
<td>DOVE transform Fringes transform</td>
</tr>
</tbody>
</table>

WrightTools seamlessly handles dimensionality throughout. *Artists* is one such place where dimensionality is addressed explicitly.

Processing without the pain

There are many common data processing operations in spectroscopy. WrightTools endeavors to make these operations easy. A selection of important methods follows.

<table>
<thead>
<tr>
<th>method</th>
<th>description</th>
<th>gallery</th>
</tr>
</thead>
<tbody>
<tr>
<td>clip()</td>
<td>clip values outside of a given range (method of Channel)</td>
<td></td>
</tr>
<tr>
<td>gradient()</td>
<td>take the derivative along an axis</td>
<td>Gradient</td>
</tr>
<tr>
<td>join()</td>
<td>join multiple data objects into one</td>
<td>Join</td>
</tr>
<tr>
<td>level()</td>
<td>level the edge of data along a certain axis</td>
<td>Level</td>
</tr>
<tr>
<td>smooth()</td>
<td>smooth a channel via convolution with a n-dimensional Kaiser window</td>
<td></td>
</tr>
</tbody>
</table>

1.4 Collection

1.4.1 Collection

Collection objects are containers, like folders in a file system. They can contain any mixture of collections and data objects. The contents of a collection can be accessed in a variety of convenient ways with WrightTools. As an example, let’s create a simple wt5 file now.

```python
import WrightTools as wt
results = wt.Collection(name='results')
```

We have created a new file with a root-level collection named results. Let’s add some data to our collection.

```python
results.create_data(name='neat')
results.create_data(name='messy')
results.create_data(name='confusing')
```

We can access/treat our collection like a dictionary with methods keys, values, and items.
>>> list(results.values())
[<WrightTools.Data 'neat'>, <WrightTools.Data 'messy'>, <WrightTools.Data 'confusing'>]

We can also access by key, or by index. We can even use natural naming!

>>> results[1]
<WrightTools.Data 'messy'>
>>> results['neat']
<WrightTools.Data 'neat'>
>>> results.confusing
<WrightTools.Data 'confusing'>

Ever think to yourself “Jeez, it would be nice to also keep track of the calibration data from our experiment”? Let’s add a child collection called calibration within our root results collection. We’ll fill this collection with our calibration data.

```python
calibration = results.create_collection(name='calibration')
calibration.create_data(name='OPA1_tune_test')
calibration.create_data(name='OPA2_tune_test')
```

This child collection can be accessed in all of the ways mentioned above (dictionary, index, natural naming). The child collections and data objects hold a reference to the parent.

```python
>>> calibration.parent
<WrightTools.Collection 'results'>
```

In summary, we have created a wt5 file with the following structure:

```plaintext
collection results
  | data neat
  | data messy
  | data confusing
  | collection calibration
  |   | data OPA1_tune_test
  |   | data OPA2_tune_test
```

Collections can be nested and added to arbitrarily in order to optimally organize and share results.

Note that the collections do not directly contain datasets. Datasets are children of the data objects. We discussed data objects in the previous section.
1.5 Artists

The artists module contains a variety of data visualization tools.

1.5.1 Quick artists

To facilitate rapid and easy visualization of data, WrightTools offers “quick” artist functions which quickly generate 1D or 2D representations. These functions are made to make good representations by default, but they do have certain keyword arguments to make popular customization easy. These are particular useful functions within the context of auto-generated plots in acquisition software.

WrightTools.artists.quick1D() is a function that generates 1D representations.

```python
import WrightTools as wt
from WrightTools import datasets
import matplotlib.pyplot as plt
wt.artists.apply_rcparams('default')

# import data
p = datasets.wt5.vlp0p0_perovskite_TA  # axes w1=wm, w2, d2
data = wt.open(p)
data.transform("w1", "w2", "d2")

# probe frequency trace
wt.artists.quick1D(data, axis=0, at={"w2": [1.7, "eV"], "d2": [0, "fs"]})

# delay trace
wt.artists.quick1D(data, axis="d2", at={"w2": [1.7, "eV"], "w1": [1.65, "eV"]})
plt.show()
```

perovskite_TA

\[ \hbar \omega_2 = 1.7 \text{ eV}, \tau_{21} = 0.821 \text{ fs} \]

WrightTools.artists.quick2D() is a function that generates 2D representations.
perovskite_TA
\( \hbar \omega_1 = 1.65 \text{ eV}, \hbar \omega_2 = 1.7 \text{ eV} \)

```python
import WrightTools as wt
from WrightTools import datasets
import matplotlib.pyplot as plt
wt.artists.apply_rcparams('default')

# import data
p = datasets.wt5.v1p0p0_perovskite_TA
data = wt.open(p)
data.transform("w1", "w2", "d2")

# probe wigner
wt.artists.quick2D(data, xaxis=0, yaxis=2, at={"w2": [1.7, "eV"]})

# 2D-frequency
wt.artists.quick2D(data, xaxis="w1", yaxis="w2", at={"d2": [0, "fs"]})
plt.show()
```

Note that the actual quick functions are each one-liners. Keyword arguments such as `autosave` and `save_directory` may be supplied if the user desires to save images (not typical for users in interactive mode). The `channel` kwarg allows users to specify what channel they would like to plot.

Perhaps the most powerful feature of `WrightTools.artists.quick1D()` and `WrightTools.artists.quick2D()` are their ability to treat higher-dimensional datasets by automatically generating multiple figures. When handing a dataset of higher dimensionality to these artists, the user may choose which axes will be plotted against using keyword arguments. Any axis not plotted against will be iterated over such that an image will be generated at each coordinate in that axis. Users may also provide a dictionary with entries of the form `{axis_name: [position, units]}` to choose a specific coordinates along non-plotted axes. Positions along non-plotted axes are reported in the title of each plot and overlines are shown when applicable. These functionalities are derived from `WrightTools.data.Data.chop()`.

1.5. Artists
\( \hbar \omega_2 = 1.7 \text{eV} \)
perovskite_TA

\[ \tau_{21} = 0.821 \text{ fs} \]
1.5.2 Interactive artists

`WrightTools.artists.interact2D()` allows users to easily visualize 2D slices of arbitrarily high dimension data.

```python
import WrightTools as wt
from WrightTools import datasets
import matplotlib.pyplot as plt

# import data
p = datasets.wt5.v1p0p0_perovskite_TA  # axes w1=wm, w2, d2
data = wt.open(p)
interact = wt.artists.interact2D(data, xaxis=0, yaxis=2, local=True, verbose=False)

# show-off functionality. The following lines are not needed when in an interactive mode.
interact[1]['w2'].set_val(40)  # hack w2 slider
fig = plt.gcf()
# simulate mouse event to get crosshairs
fig.canvas.button_release_event(160, 375, 1)
plt.show()
```

Side plots show the x and y projections of the slice (shaded gray). For signed channels, side plots will also show projections of the negatively signed components and positively signed components. Left clicks on the main axes draw 1D slices on side plots at the coordinates selected. Right clicks remove the 1D slices. For 3+ dimensional data, sliders below the main axes are used to change which slice is viewed. `interact2D` also supports keyboard navigation:

<table>
<thead>
<tr>
<th>key</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>tab / ctrl+tab</td>
<td>cycle focus between the sliders and the plot</td>
</tr>
<tr>
<td>left/right arrow</td>
<td>decrement/increment slice (slider focus) or change y slice (plot focus)</td>
</tr>
<tr>
<td>up/down arrow</td>
<td>change x slice (plot focus)</td>
</tr>
</tbody>
</table>

Note that the left/right arrow navigation overrides the built-in undo/redo action of the qt viewer. Users can still undo/redo with the ‘c/v’ key presses, or through the GUI toolbar above the figure.

1.5.3 Colors

Two-dimensional data is often represented using “heatmaps”. Your choice of colormap is a crucial part of how your data is perceived. WrightTools has a few choices colormaps built-in.

All of these are held in the `colormaps` dictionary.

```python
>>> wt.artists.colormaps['default']
<matplotlib.colors.LinearSegmentedColormap at 0x7f6d8b658d30>
```

Throughout WrightTools you can refer to colormaps by their name. By default, WrightTools will use the “default” colormap when plotting unsigned channels and the “signed” colormap when plotting signed channels.

There are many great resources on how to choose the best colormap. Choosing Colormaps is a great place to start reading. WrightTools tries to use perceptual colormaps wherever possible. When a large dynamic range is needed, the data can always be scaled to accommodate.

The default colormap is based on the wonderful cubehelix color scheme.\(^1\) The cubehelix parameters have been fine-tuned to roughly mimic the colors of the historically popular “jet” colormap.

perovskite_TA

\[ \hbar \omega_1 = \hbar \omega_m \text{(eV)} \]

\[ \tau_{21} \text{(fs)} \]

\[ \text{dOD} \]

1.5. Artists
The isoluminant series are instances of the color scheme proposed by Kindlmann et al.\(^2\)
The skybar series were designed by Schuyler (Skye) Kain for use in his instrumental software package COLORS.
wright and signed_old are kept for legacy purposes.

### 1.5.4 Custom figures

WrightTools offers specialized tools for custom publication quality figures. As an example, we will break down the figure in Custom Figure, exploring the relationships between WrightTools and the underlying matplotlib.

The preprocessing of data is handled in tools covered in Data.

First, the full code and the image it creates:

```python
import matplotlib.pyplot as plt
import numpy as np
import WrightTools as wt
from WrightTools import datasets

# obtain and process data
p = datasets.wt5.vlp0p1_MoS2_TrEE_movie
data = wt.open(p)
data.level(0, 2, -3)
data.convert("eV", convert_variables=True, verbose=False)
data.smooth([2, 2, 2])
data.ai0.symmetric_root(2)
data.ai0.normalize()
data.ai0.clip(min=0, replace="value")
# chop out data of interest
d2_vals = [-50, -500]
w2_vals = [1.7, 1.8, 1.9, 2.0]
wigners = [data.chop("w1=wm", "d2", at={"w2": [w2, "eV"]})[0] for w2 in w2_vals]
traces1 = [
    data.chop("w1=wm", at={"w2": [w2, "eV"], "d2": [d2_vals[0], "fs"]})[0] for w2 in w2_vals
]
traces2 = [
    data.chop("w1=wm", at={"w2": [w2, "eV"], "d2": [d2_vals[1], "fs"]})[0] for w2 in w2_vals
]
tracess = [traces1, traces2]
# prepare spine colors
wigner_colors = ["C0", "C1", "C2", "C3"]
trace_colors = ["#FE4EDA", "#00B7EB"]
# prepare figure gridspec
cols = [1, 1, "cbar"]
aspects = [[[0, 0], .3]]
fig, gs = wt.artists.create_figure(
    width="double", cols=cols, nrows=3, aspects=aspects, wspace=.35, hspace=.35
)
# plot wigners
indxs = [(row, col) for row in range(1, 3) for col in range(2)]
```

\(^2\)Face-based luminance matching for perceptual colormap generation G. Kindlmann, E. Reinhard, and S Creem IEEE Visualization 2002
doi:10.1109/visual.2002.1183788
for indx, wigner, color in zip(indxs, wigners, wigner_colors):
    ax = plt.subplot(gs[indx])
    ax.pcolor(wigner, vmin=0, vmax=1)  # global colormap
    ax.contour(wigner)  # local contours
    ax.grid()
    wt.artists.set_ax_spines(ax=ax, c=color)
    # set title as value of w2
    wigner.constants[0].format_spec = "%.2f"
    wigner.round_spec = -1
    wt.artists.corner_text(wigner.constants[0].label, ax=ax)
    # plot overlines
    for d2, t_color in zip(d2_vals, trace_colors):
        ax.axhline(d2, color=t_color, alpha=.5, linewidth=6)
    # plot w2 placement
    ax.axvline(wigner.w2.points, color="grey", alpha=.75, linewidth=6)
    # plot traces
    indxs = [(0, col) for col in range(2)]
    for indx, color, traces in zip(indxs, trace_colors, tracess):
        ax = plt.subplot(gs[indx])
        for trace, w_color in zip(traces, wigner_colors):
            ax.plot(trace, color=w_color, linewidth=1.5)
        ax.grid()
        ax.set_xlim(trace.axes[0].min(), trace.axes[0].max())
        wt.artists.set_ax_spines(ax=ax, c=color)
    # plot colormap
    cax = plt.subplot(gs[1:3, -1])
    ticks = np.linspace(data.ai0.min(), data.ai0.max(), 11)
    wt.artists.plot_colorbar(cax=cax, label="amplitude", cmap="default", ticks=ticks)
    # set axis labels
    wt.artists.set_fig_labels(xlabel=data.w1__e__wm.label, ylabel=data.d2.label, col=slice(0, 1))
    # ylabel of zeroth row
    ax = plt.subplot(gs[0, 0])
    ax.set_ylabel("amplitude")
    # saving the figure as a png
    wt.artists.savefig("custom_fig.png", fig=fig, close=False)

_layout.png

Layout

WrightTools defines a handy function, `create_figure()`, for easily and flexibly making complicated figures. When made with this function, Axes created have additional functionality built in to work with Data objects directly.

`create_figure()` makes it easy to create figures the perfect size for "single" or double" column figures for journal articles (though they are convenient in other contexts as well).

`create_figure()` also creates a GridSpec to help layout subplots. Columns are created with a weighted list with the number of columns, passed as cols. A special weight, "cbar", provides a fixed width column intended for color bars. All other columns are proportionally distributed according to their weights. The number of rows in the
grid are specified with the `nrows` kwarg. You can modify the aspect ratio of particular rows independently using the `aspects` and `default_aspect` kwargs.

Spacing between figures can be adjusted with the `wspace` and `hspace` kwargs for the width and height, respectively.

Axes can be accessed with `matplotlib.pyplot.subplot()`. Importantly, axes may span multiple rows/columns by using slice syntax into the gridspec. This is demonstrated with the color bar axes here, which takes up two rows in the last column.

```python
# prepare figure gridspec
cols = [1, 1, "cbar"]
aspects = [[[0, 0], .3]]
fig, gs = wt.artists.create_figure(
    width="double", cols=cols, nrows=3, aspects=aspects, wspace=1.35, hspace=.35
)
# plot wigners
indxs = [(row, col) for row in range(1, 3) for col in range(2)]
for indx, wigner, color in zip(indxs, wigners, wigner_colors):
    ax = plt.subplot(gs[indx])
...
indxs = [(0, col) for col in range(2)]
for indx, color, traces in zip(indxs, trace_colors, tracess):
    ax = plt.subplot(gs[indx])
...
cax = plt.subplot(gs[1:3, -1])
```

Plot

Once you have axes with the `subplot()` call, it can be used as you are used to using `matplotlib.axes.Axes` objects (though some defaults, such as colormap, differ from bare `matplotlib`). However, you can also pass `WrightTools.data.Data` objects in directly (and there are some kwargs available when you do). These `WrightTools.artists.Axes` will extract out the proper arrays and plot the data.

```python
for indx, wigner, color in zip(indxs, wigners, wigner_colors):
    ax = plt.subplot(gs[indx])
    ax.pcolor(wigner, vmin=0, vmax=1)  # global colormap
    ax.contour(wigner)  # local contours
...
for indx, color, traces in zip(indxs, trace_colors, tracess):
    ax = plt.subplot(gs[indx])
    for trace, w_color in zip(traces, wigner_colors):
        ax.plot(trace, color=w_color, linewidth=1.5)
```

Beautify

Once the main data is plotted, additional information can be overlaid on the axes. Of course, standard `matplotlib` methods like `axhline()` or `set_xlim()` are all available. In addition, `WrightTools` defines some small helper functions for common tasks.

- **set_ax_spines()** Easily set color/width of the outline (spines) of an axis
  - Great for using color to connect different parts of a figure (or figures throughout a larger work)
- **corner_text()** Quick and easy plot labeling within a dense grid
  - Pairs well with `WrightTools.data.Constant.label`
- **plot_colorbar()** Add a colorbar in a single function call
• **set_fig_labels()** Label axes in a whole row/column of a figure
  
  – Allows the use of slice objects to limit range affected
  
  – Removes axis labels from other axes in the rectangle
  
  – Pairs well with `WrightTools.data.Axes.label`

```python
wigner_colors = ['C0', 'C1', 'C2', 'C3']
trace_colors = ['#FE4EDA', '#00B7EB']
...
for indx, wigner, color in zip(indxs, wigners, wigner_colors):
  ...
  ax.grid()
  wt.artists.set_ax_spines(ax=ax, c=color)
  # set title as value of w2
  wigner.constants[0].format_spec = '.2f'
  wigner.round_spec = -1
  wt.artists.corner_text(wigner.constants[0].label, ax=ax)
  # plot overlines
  for d2, t_color in zip(d2_vals, trace_colors):
    ax.axhline(d2, color=t_color, alpha=.5, linewidth=6)
    # plot w2 placement
    ax.axvline(wigner.w2.points, color='grey', alpha=.75, linewidth=6)
  ...
for indx, color, traces in zip(indxs, trace_colors, tracess):
  ...
  ax.set_xlim(traces.axes[0].min(), traces.axes[0].max())
  wt.artists.set_ax_spines(ax=ax, c=color)
  # plot colormap
  cax = plt.subplot(gs[1:3, -1])
  ticks = np.linspace(data.ai0.min(), data.ai0.max(), 11)
  wt.artists.plot_colorbar(cax=cax, label='amplitude', cmap='default', ticks=ticks)
  # set axis labels
  wt.artists.set_fig_labels(xlabel=data.w1__e__wm.label, ylabel=data.d2.label, col=slice(0, 1))
```

### Save

Saving figures is as easy as calling `savefig()`. This is a simple wrapper for `matplotlib.pyplot.savefig()` which allows us to override defaults so that figures created with `create_figure()` have proper margins and resolution. If you wish to change margin padding or transparancy settings, the matplotlib function will work just as well.

```python
# saving the figure as a png
wt.artists.savefig("custom_fig.png", fig=fig, close=False)
```
1.6 Units

WrightTools provides its own units system. You can use it directly, if you wish.

```python
>>> import WrightTools as wt
>>> wt.units.converter(2., 'eV', 'nm')
620.0
```

This same units system enables the units-aware properties throughout WrightTools.

In WrightTools, units are organized into kinds. It is always possible to convert between units of the same kind, and never possible to convert between kinds.

The units system also provides a symbol for each unit, enabling easy plotting.

The following table contains every unit in WrightTools.

<table>
<thead>
<tr>
<th>name</th>
<th>description</th>
<th>kind</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>rad</td>
<td>radian</td>
<td>angle</td>
<td>None</td>
</tr>
<tr>
<td>deg</td>
<td>degrees</td>
<td>angle</td>
<td>None</td>
</tr>
<tr>
<td>fs</td>
<td>femtoseconds</td>
<td>delay</td>
<td>τ</td>
</tr>
<tr>
<td>ps</td>
<td>picoseconds</td>
<td>delay</td>
<td>τ</td>
</tr>
<tr>
<td>ns</td>
<td>nanoseconds</td>
<td>delay</td>
<td>τ</td>
</tr>
<tr>
<td>mm_delay</td>
<td>mm</td>
<td>delay</td>
<td>τ</td>
</tr>
<tr>
<td>nm</td>
<td>nanometers</td>
<td>energy</td>
<td>λ</td>
</tr>
<tr>
<td>wn</td>
<td>wavenumbers</td>
<td>energy</td>
<td>¯ν</td>
</tr>
<tr>
<td>eV</td>
<td>electronvolts</td>
<td>energy</td>
<td>ℏω</td>
</tr>
<tr>
<td>meV</td>
<td>millielectronvolts</td>
<td>energy</td>
<td>E</td>
</tr>
<tr>
<td>Hz</td>
<td>hertz</td>
<td>energy</td>
<td>f</td>
</tr>
<tr>
<td>THz</td>
<td>terahertz</td>
<td>energy</td>
<td>f</td>
</tr>
<tr>
<td>GHz</td>
<td>gigahertz</td>
<td>energy</td>
<td>f</td>
</tr>
<tr>
<td>K</td>
<td>kelvin</td>
<td>temperature</td>
<td>T</td>
</tr>
<tr>
<td>deg_C</td>
<td>celsius</td>
<td>temperature</td>
<td>T</td>
</tr>
<tr>
<td>deg_F</td>
<td>fahrenheit</td>
<td>temperature</td>
<td>T</td>
</tr>
<tr>
<td>deg_R</td>
<td>rankine</td>
<td>temperature</td>
<td>T</td>
</tr>
<tr>
<td>fluence</td>
<td>uJ per sq. cm</td>
<td>intensity</td>
<td>ℱ</td>
</tr>
<tr>
<td>mOD</td>
<td>mOD</td>
<td>optical density</td>
<td>None</td>
</tr>
<tr>
<td>OD</td>
<td>OD</td>
<td>optical density</td>
<td>None</td>
</tr>
<tr>
<td>nm_p</td>
<td>nanometers</td>
<td>position</td>
<td>None</td>
</tr>
<tr>
<td>um</td>
<td>microns</td>
<td>position</td>
<td>None</td>
</tr>
<tr>
<td>mm</td>
<td>millimeters</td>
<td>position</td>
<td>None</td>
</tr>
<tr>
<td>cm</td>
<td>centimeters</td>
<td>position</td>
<td>None</td>
</tr>
<tr>
<td>in</td>
<td>inches</td>
<td>position</td>
<td>None</td>
</tr>
<tr>
<td>FWHM</td>
<td>full width half max</td>
<td>pulse width</td>
<td>σ</td>
</tr>
<tr>
<td>fs_t</td>
<td>femtoseconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>ps_t</td>
<td>picoseconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>ns_t</td>
<td>nanoseconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>us_t</td>
<td>microseconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>ms_t</td>
<td>milliseconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>s_t</td>
<td>seconds</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>m_t</td>
<td>minutes</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>h_t</td>
<td>hours</td>
<td>time</td>
<td>None</td>
</tr>
<tr>
<td>d_t</td>
<td>days</td>
<td>time</td>
<td>None</td>
</tr>
</tbody>
</table>
1.7 Datasets

A few example datasets are distributed within WrightTools. These make it easy to demonstrate and test data processing features. They’re also a lot of fun!

The following table contains every dataset distributed within WrightTools.

<table>
<thead>
<tr>
<th>dataset</th>
<th>axis expressions</th>
<th>shape</th>
<th>gallery links</th>
</tr>
</thead>
<tbody>
<tr>
<td>BrunoldrRaman. LDS821_514nm_80mW</td>
<td>('energy',)</td>
<td>(1340,)</td>
<td>Resonance Raman</td>
</tr>
<tr>
<td>Cary.CuPCtS_H2O_vis (collection)</td>
<td>('wavelength',)</td>
<td>(141,)</td>
<td></td>
</tr>
<tr>
<td>Cary.filters (collection)</td>
<td>('wavelength',)</td>
<td>multiple</td>
<td>Plotting Multiple Lines</td>
</tr>
<tr>
<td>COLORS.v0p2_d1_d2_diagonal</td>
<td>('d1', 'd2')</td>
<td>(21, 21)</td>
<td>Fill types</td>
</tr>
<tr>
<td>COLORS.v2p2_WL_wigner</td>
<td>('wm', 'd1')</td>
<td>(241, 51)</td>
<td></td>
</tr>
<tr>
<td>JASCO.PbSe_batch_1</td>
<td>('energy',)</td>
<td>(1801,)</td>
<td></td>
</tr>
<tr>
<td>JASCO.PbSe_batch_4_2012_02_21</td>
<td>('energy',)</td>
<td>(1251,)</td>
<td></td>
</tr>
<tr>
<td>JASCO.PbSe_batch_4_2012_03_15</td>
<td>('energy',)</td>
<td>(1251,)</td>
<td></td>
</tr>
<tr>
<td>KENT.LDS821_DOVE</td>
<td>('w2', 'w1')</td>
<td>(60, 60)</td>
<td>DOVE transform</td>
</tr>
<tr>
<td>KENT.LDS821_TRSF</td>
<td>('w2', 'w1')</td>
<td>(71, 71)</td>
<td>Quick 2D, Quick 1D</td>
</tr>
<tr>
<td>KENT.PbSe_2D_delay_B</td>
<td>('d2', 'd1')</td>
<td>(101, 101)</td>
<td></td>
</tr>
<tr>
<td>ocean_optics.tsunami</td>
<td>('energy',)</td>
<td>(2048,)</td>
<td></td>
</tr>
<tr>
<td>PyCMDS.d1_d2_000</td>
<td>('d1', 'd2')</td>
<td>(101, 101)</td>
<td>Label delay space</td>
</tr>
<tr>
<td>PyCMDS.d1_d2_001</td>
<td>('d1', 'd2')</td>
<td>(101, 101)</td>
<td>Label delay space</td>
</tr>
<tr>
<td>PyCMDS.w1_000</td>
<td>('w1',)</td>
<td>(51,)</td>
<td></td>
</tr>
<tr>
<td>PyCMDS.w1_wa_000</td>
<td>('wl=wm', 'wa')</td>
<td>(25, 256)</td>
<td>Tune test</td>
</tr>
<tr>
<td>PyCMDS.w2_w1_000</td>
<td>('w2', 'w1')</td>
<td>(81, 81)</td>
<td>Fringes transform</td>
</tr>
<tr>
<td>PyCMDS.wm_w2_w1_000</td>
<td>('wm', 'w2', 'w1')</td>
<td>(35, 11, 11)</td>
<td></td>
</tr>
<tr>
<td>PyCMDS.wm_w2_w1_001</td>
<td>('wm', 'w2', 'w1')</td>
<td>(29, 11, 11)</td>
<td></td>
</tr>
<tr>
<td>Shimadzu.MoS2_fromCzech2015</td>
<td>('energy',)</td>
<td>(819,)</td>
<td></td>
</tr>
<tr>
<td>Solis.wm_ypos_fluorescence_with_trilam</td>
<td>('wm', 'ypos')</td>
<td>(2560, 2160)</td>
<td></td>
</tr>
<tr>
<td>Solis.xpos_ypos_fluorescence</td>
<td>('xpos', 'ypos')</td>
<td>(2560, 2160)</td>
<td></td>
</tr>
<tr>
<td>spcm.test_data</td>
<td>('time',)</td>
<td>(1024,)</td>
<td></td>
</tr>
<tr>
<td>spcm.test_data_full_metadata</td>
<td>('time',)</td>
<td>(1024,)</td>
<td></td>
</tr>
<tr>
<td>Tensor27.CuPCtS_powder_ATR</td>
<td>('energy',)</td>
<td>(7259,)</td>
<td></td>
</tr>
<tr>
<td>wt5.vlp0p0_perovskite_TA</td>
<td>('wl=wm', 'w2', 'd2')</td>
<td>(52, 52, 13)</td>
<td>Quick 2D Signed</td>
</tr>
<tr>
<td>wt5.vlp0p1_MoS2_TrEE_movie</td>
<td>('w2', 'w1', 'd2')</td>
<td>(41, 41, 23)</td>
<td>Level, Colormaps</td>
</tr>
</tbody>
</table>


6 Multiresonant Coherent Multidimensional Electronic Spectroscopy of Colloidal PbSe Quantum Dots Lena A. Yurs, Stephen B.
1.8 Contributing

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

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

Are you interested in adding support for yet another data format? Please see write_from_function.

1.8.1 Preparing

1. fork the WrightTools repository (if you have push access to the main repository you can skip this step)
2. clone WrightTools 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

   Note: On *nix machines (unfortunately this does not work on Windows), the tests may be multiprocessed using pytest-mp:
   
   $ pip install pytest-mp
   $ pytest --mp

1.8.2 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

References:

$ 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!

1.8.3 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

WrightTools follows pep8, with the following modifications:

1. Maximum line length from 79 characters to 99 characters.
WrightTools also follows numpy Docstring Convention, which is a set of adjustments to pep257. WrightTools additionally ignores one guideline:

1. WrightTools 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.
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.

## 1.9 The wt5 File Format

WrightTools stores data in binary wt5 files.

wt5 is a sub-format of HDF5.

### 1.9.1 wt5

wt5 files are hdf5 files with particular structure and attributes defined. wt5 objects may appear embedded within a larger hdf5 file or vise-versa, however this is untested. At the root of a wt5 file, a `Collection` or `Data` object is found. `Collection` and `Data` are hdf5 groups. A `Collection` may have children consisting of `Collection` and/or `Data`. A `Data` may have children consisting of `Variable` and/or `Channel`. `Variable` and `Channel` are hdf5 datasets.

**Metadata**

The following metadata is handled within WrightTools and define the necessary attributes to be a wt5 file. It is recommended not to write over these attributes manually except at import time (e.g. `from_<x>` function).
### 1.9.2 HDF5

The HDF5 data model contains two primary objects: the group and the dataset. Groups are used to hierarchically organize content within the file. Each group is a container for datasets and other groups. Think of groups like folders in your computer's file system. Every HDF5 file contains a top-level root group, signified by `/`.

Datasets are specialty containers for raw data values. Think of datasets like multidimensional arrays, similar to the `numpy ndarray`. Each dataset has a specific data type, such as integer, float, or character.

Groups and datasets can contain additional metadata. This metadata is stored in a key: value pair system called `attrs`, similar to a python dictionary.

Much more information can be found on the HDF5 tutorial.

WrightTools relies upon the h5py package, a Pythonic interface to HDF5.
1.9.3 Access

wt5 is a binary format, so it cannot be interpreted with traditional text editors. Since wt5 is a sub-format of HDF5, WrightTools benefits from the ecosystem of HDF5 tools that already exists. This means that it is possible to import and interact with wt5 files without WrightTools, or even without python.

ASCII

Export an HDF5 file to a human-readable ASCII file using h5dump.
See also HDF to Excel.

Fortran

Use the official HDF5 Fortran Library.

Graphical

HDF COMPASS, a simple tool for navigating and viewing data within HDF5 files (no editing functionality).
HDF VIEW, a visual tool for browsing and editing HDF5 files.

MATLAB

MATLAB offers built-in high-level HDF5 functions including h5disp, h5read, and h5readatt.

Python (without WrightTools)

We recommend the amazing h5py package.

Shell

h5cli: bash-like interface to interacting with HDF5 files.
h5diff: compare two HDF5 files, reporting the differences.
h5ls: print information about one or more HDF5 files.
Complete list of official HDF5 tools

1.9.4 Changes

Version 1.0.0

Initial release of the format.
Version 1.0.1

Changes internal handling of strings. Bare strings are no longer required to call `encode()` before storing.

Version 1.0.2

Adds “constants” as a stored attribute in the attrs dictionary, a list of strings just like axes.

Version 1.0.3

Changed identity as stored in attrs dictionary (axis and constant) to use the expression including operators. Previous versions exhibited a bug where decimal points would be ignored when the expression was generated from the attrs (thus “2.0” would be stored as “2_0” and read in as “20”).

1.10 WrightTools API

1.10.1 WrightTools.artists module

Artists.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axes(fig, rect[, facecolor, frameon, ...])</td>
<td>Axes.</td>
</tr>
<tr>
<td>Figure([figsize, dpi, facecolor, edgecolor, ...])</td>
<td>Figure.</td>
</tr>
<tr>
<td>GridSpec(nrows, ncols[, figure, left, ...])</td>
<td>GridSpec.</td>
</tr>
<tr>
<td>add_sideplot(ax, along[, pad, grid, ...])</td>
<td>Add a sideplot to an axis.</td>
</tr>
<tr>
<td>apply_rcparams(kind)</td>
<td>Quickly apply rcparams for given purposes.</td>
</tr>
<tr>
<td>colormaps</td>
<td></td>
</tr>
<tr>
<td>corner_text(text[, distance, ax, corner, ...])</td>
<td>Place some text in the corner of the figure.</td>
</tr>
<tr>
<td>create_figure(*[, width, nrows, cols, ...])</td>
<td>Re-parameterization of matplotlib figure creation tools, exposing convenient variables.</td>
</tr>
<tr>
<td>diagonal_line((xi, yi, ax, c, ls, lw, zorder))</td>
<td>Plot a diagonal line.</td>
</tr>
<tr>
<td>get_color_cycle(n[, cmap, rotations])</td>
<td>Get a list of RGBA colors following a colormap.</td>
</tr>
<tr>
<td>get_scaled_bounds(ax, position, *[...])</td>
<td>Get scaled bounds.</td>
</tr>
<tr>
<td>grayify_cmap(cmap)</td>
<td>Return a grayscale version of the colormap.</td>
</tr>
<tr>
<td>interact2D(data[, xaxis, yaxis, channel, ...])</td>
<td>Interactive 2D plot of the dataset.</td>
</tr>
<tr>
<td>overline_colors</td>
<td>list() -&gt; new empty list list(iterable) -&gt; new list initialized from iterable’s items</td>
</tr>
<tr>
<td>pcolor_helper(xi, yi[, zi])</td>
<td>Prepare a set of arrays for plotting using <code>pcolor</code>.</td>
</tr>
<tr>
<td>plot_colorbar(lcax, cmap, ticks, clim, ...)</td>
<td>Easily add a colormap to an axis.</td>
</tr>
<tr>
<td>plot_colormap_components(cmap)</td>
<td>Plot the components of a given colormap.</td>
</tr>
<tr>
<td>plot_gridlines(ax, c, lw, diagonal, ...)</td>
<td>Plot dotted gridlines onto an axis.</td>
</tr>
<tr>
<td>plot_margins(*[, fig, inches, centers, edges])</td>
<td>Add lines onto a figure indicating the margins, centers, and edges.</td>
</tr>
<tr>
<td>quick1D(data[, axis, at, channel, local, ...])</td>
<td>Quickly plot 1D slice(s) of data.</td>
</tr>
<tr>
<td>quick2D(data[, xaxis, yaxis, at, channel, ...])</td>
<td>Quickly plot 2D slice(s) of data.</td>
</tr>
<tr>
<td>savefig(path[, fig, close])</td>
<td>Save a figure.</td>
</tr>
<tr>
<td>set_ax_labels((ax, xlabel, ylabel, xticks, ...))</td>
<td>Set all axis labels properties easily.</td>
</tr>
<tr>
<td>set_ax_spines((ax, c, lw, zorder))</td>
<td>Easily set the properties of all four axis spines.</td>
</tr>
<tr>
<td>set_fig_labels((fig, xlabel, ylabel, ...))</td>
<td>Set all axis labels of a figure simultaneously.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>stitch_to_animation</code></td>
<td>Stitch a series of images into an animation.</td>
</tr>
<tr>
<td><code>subplots_adjust</code></td>
<td>Enforce margin to be equal around figure, starting at subplots.</td>
</tr>
</tbody>
</table>

**WrightTools.artists.Axes**

```python
class WrightTools.artists.Axes(fig, rect, facecolor=None, frameon=True, sharex=None, sharey=None, label='', xscale=None, yscale=None, **kwargs)
```

Bases: `matplotlib.axes._axes.Axes`

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>add_sideplot</code></td>
<td>Add a side axis.</td>
</tr>
<tr>
<td><code>contour</code></td>
<td>Plot contours.</td>
</tr>
<tr>
<td><code>contourf</code></td>
<td>Plot contours.</td>
</tr>
<tr>
<td><code>legend</code></td>
<td>Add a legend.</td>
</tr>
<tr>
<td><code>pcolor</code></td>
<td>Create a pseudocolor plot of a 2-D array.</td>
</tr>
<tr>
<td><code>pcolormesh</code></td>
<td>Create a pseudocolor plot of a 2-D array.</td>
</tr>
<tr>
<td><code>plot</code></td>
<td>Plot lines and/or markers.</td>
</tr>
</tbody>
</table>

**WrightTools.artists.Axes.add_sideplot**

```python
Axes.add_sideplot(along[, pad, height, ymin, ymax])
```

Add a side axis.

Parameters

- `along` (`{'x', 'y'}`) – Axis to add along.
- `pad` (`float (optional)`) – Side axis pad. Default is 0.
- `height` (`float (optional)`) – Side axis height. Default is 0.

**WrightTools.artists.Axes.contour**

```python
Axes.contour(*args, **kwargs)
```

Plot contours.

If a 3D or higher Data object is passed, a lower dimensional channel can be plotted, provided the `squeeze` of the channel has `ndim==2` and the first two axes do not span dimensions other than those spanned by that channel.

Parameters

- `data` (`2D WrightTools.data.Data object`) – Data to plot.
- `channel` (`int or string (optional)`) – Channel index or name. Default is 0.
- `dynamic_range` (`boolean (optional)`) – Force plotting of all contours, overloading for major extent. Only applies to signed data. Default is False.
• **autolabel** ("none", 'both', 'x', 'y') (optional) – Parameterize application of labels directly from data object. Default is none.

• **xlabel** (string (optional)) – xlabel. Default is None.

• **ylabel** (string (optional)) – ylabel. Default is None.

• ****kwargs – matplotlib.axes.Axes.contour optional keyword arguments.

Returns

Return type: matplotlib.contour.QuadContourSet

**WrightTools.artists.Axes.contourf**

Axes.contourf(*args, **kwargs)

Plot contours.

If a 3D or higher Data object is passed, a lower dimensional channel can be plotted, provided the squeeze of the channel has ndim==2 and the first two axes do not span dimensions other than those spanned by that channel.

Parameters

• **data** (2D WrightTools.data.Data object) – Data to plot.

• **channel** (int or string (optional)) – Channel index or name. Default is 0.

• **dynamic_range** (boolean (optional)) – Force plotting of all contours, overloading for major extent. Only applies to signed data. Default is False.

• **autolabel** ("none", 'both', 'x', 'y') (optional) – Parameterize application of labels directly from data object. Default is none.

• **xlabel** (string (optional)) – xlabel. Default is None.

• **ylabel** (string (optional)) – ylabel. Default is None.

• ****kwargs – matplotlib.axes.Axes.contourf optional keyword arguments.

Returns

Return type: matplotlib.contour.QuadContourSet

**WrightTools.artists.Axes.legend**

Axes.legend(*args, **kwargs)

Add a legend.

Parameters

• **args** – matplotlib legend args.

• **kwargs** – matplotlib legend kwargs.

Returns

Return type: legend
### WrightTools.artists.Axes.pcolor

**Axes.pcolor(*args, **kwargs)**

Create a pseudocolor plot of a 2-D array.

If a 3D or higher Data object is passed, a lower dimensional channel can be plotted, provided the squeeze of the channel has `ndim==2` and the first two axes do not span dimensions other than those spanned by that channel.

Uses pcolor_helper to ensure that color boundaries are drawn bisecting point positions, when possible.

**Parameters**

- **data** *(2D WrightTools.data.Data object)* – Data to plot.
- **channel** *(int or string (optional)) – Channel index or name. Default is 0.*
- **dynamic_range** *(boolean (optional)) – Force plotting of all contours, over-loading for major extent. Only applies to signed data. Default is False.*
- **autolabel** *({'none', 'both', 'x', 'y'} (optional)) – Parameterize application of labels directly from data object. Default is none.*
- **xlabel** *(string (optional)) – xlabel. Default is None.*
- **ylabel** *(string (optional)) – ylabel. Default is None.*

**Returns**

- **Return type** *matplotlib.collections.PolyCollection*

### WrightTools.artists.Axes.pcolormesh

**Axes.pcolormesh(*args, **kwargs)**

Create a pseudocolor plot of a 2-D array.

If a 3D or higher Data object is passed, a lower dimensional channel can be plotted, provided the squeeze of the channel has `ndim==2` and the first two axes do not span dimensions other than those spanned by that channel.

Uses pcolor_helper to ensure that color boundaries are drawn bisecting point positions, when possible. Quicker than pcolor

**Parameters**

- **data** *(2D WrightTools.data.Data object)* – Data to plot.
- **channel** *(int or string (optional)) – Channel index or name. Default is 0.*
- **dynamic_range** *(boolean (optional)) – Force plotting of all contours, over-loading for major extent. Only applies to signed data. Default is False.*
- **autolabel** *({'none', 'both', 'x', 'y'} (optional)) – Parameterize application of labels directly from data object. Default is none.*
- **xlabel** *(string (optional)) – xlabel. Default is None.*
- **ylabel** *(string (optional)) – ylabel. Default is None.*

**Returns**

- **Return type** *matplotlib.collections.PolyCollection*
Return type  matplotlib.collections.QuadMesh

WrightTools.artists.Axes.plot

Axes.plot(*args, **kwargs)
Plot lines and/or markers.
If a 2D or higher Data object is passed, a lower dimensional channel can be plotted, provided the squeeze of the channel has ndim=1 and the first axis does not span dimensions other than that spanned by the channel.

Parameters

• data (1D WrightTools.data.Data object) – Data to plot.
• channel (int or string (optional)) – Channel index or name. Default is 0.
• dynamic_range (boolean (optional)) – Force plotting of all contours, overloading for major extent. Only applies to signed data. Default is False.
• autolabel ("none", "both", "x", "y") (optional) – Parameterize application of labels directly from data object. Default is none.
• xlabel (string (optional)) – xlabel. Default is None.
• ylabel (string (optional)) – ylabel. Default is None.
• **kwargs – matplotlib.axes.Axes.plot optional keyword arguments.

Returns  list of matplotlib.lines.line2D objects
Return type  list

WrightTools.artists.Figure

class WrightTools.artists.Figure(figsize=None, dpi=None, facecolor=None, edgecolor=None, linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)
Bases: matplotlib.figure.Figure
Figure.

WrightTools.artists.GridSpec

class WrightTools.artists.GridSpec(nrows, ncols, figure=None, left=None, bottom=None, right=None, top=None, wspace=None, hspace=None, width_ratios=None, height_ratios=None)
Bases: matplotlib.gridspec.GridSpec
GridSpec.
WrightTools.artists.add_sideplot

WrightTools.artists.add_sideplot(ax, along='x', pad=0.0, *, grid=True, zero_line=True, arrs_to_bin=None, normalize_bin=True, ymin=0, ymax=1.1, height=0.75, c='C0')

Add a sideplot to an axis. Sideplots share their corresponding axis.

Parameters

• ax (matplotlib AxesSubplot object) – The axis to add a sideplot along.
• along ({'x', 'y'}) – The dimension to add a sideplot along.
• pad (number (optional)) – Distance between axis and sideplot. Default is 0.
• grid (bool (optional)) – Toggle for plotting grid on sideplot. Default is True.
• zero_line (bool (optional)) – Toggle for plotting black line at zero signal. Default is True.
• arrs_to_bin (list [xi, yi, zi] (optional)) – Bins are plotted if arrays are supplied. Default is None.
• normalize_bin (bool (optional)) – Normalize bin by max value. Default is True.
• ymin (number (optional)) – Bin minimum extent. Default is 0.
• ymax (number (optional)) – Bin maximum extent. Default is 1.1
• c (string (optional)) – Line color. Default is C0.

Returns AxesSubplot object

Return type axCorr

WrightTools.artists.apply_rcparams

WrightTools.artists.apply_rcparams(kind='fast')

Quickly apply rcparams for given purposes.

Parameters kind ({'default', 'fast', 'publication'} (optional)) – Settings to use. Default is ‘fast’.

WrightTools.artists.colormaps

WrightTools.artists.colormaps = {'cube': <matplotlib.colors.LinearSegmentedColormap object>, ...

WrightTools.artists.corner_text

WrightTools.artists.corner_text(text, distance=0.075, *, ax=None, corner='UL', factor=200, bbox=True, fontsize=18, background_alpha=1, edgecolor=None)

Place some text in the corner of the figure.

Parameters

• text (str) – The text to use.
• distance (number (optional)) – Distance from the corner. Default is 0.05.
• `ax` (*axis (optional)*) – The axis object to label. If None, uses current axis. Default is None.

• `corner` (*{'UL', 'LL', 'UR', 'LR'} (optional)*) – The corner to label. Upper left, Lower left etc. Default is UL.

• `factor` (*number (optional)*) – Scaling factor. Default is 200.

• `bbox` (*boolean (optional)*) – Toggle bounding box. Default is True.

• `fontsize` (*number (optional)*) – Text fontsize. If None, uses the matplotlib default. Default is 18.

• `background_alpha` (*number (optional)*) – Opacity of background bounding box. Default is 1.

• `edgecolor` (*string (optional)*) – Frame edgecolor. Default is None (inherits from legend.edgecolor rcparam).

Returns The matplotlib text object.

Return type text

**WrightTools.artists.create_figure**

WrightTools.artists.create_figure(*, width='single', nrows=1, cols=[1], margin=1.0, hspace=0.25, wspace=0.25, cbar_width=0.25, aspects=[], default_aspect=1)

Re-parameterization of matplotlib figure creation tools, exposing convenient variables.

Figures are defined primarily by their width. Height is defined by the aspect ratios of the subplots contained within. hspace, wspace, and cbar_width are defined in inches, making it easier to make consistent plots. Margins are enforced to be equal around the entire plot, starting from the edges of the subplots.

Parameters

• `width` (*{'single', 'double', 'dissertation'} or float (optional)*) – The total width of the generated figure. Can be given in inches directly, or can be specified using keys. Default is ‘single’ (6.5 inches).

• `nrows` (*int (optional)*) – The number of subplot rows in the figure. Default is 1.

• `cols` (*list (optional)*) – A list of numbers, defining the number and width-ratios of the figure columns. May also contain the special string ‘cbar’, defining a column as a colorbar-containing column. Default is [1].

• `margin` (*float (optional)*) – Margin in inches. Margin is applied evenly around the figure, starting from the subplot boundaries (so that ticks and labels appear in the margin). Default is 1.

• `hspace` (*float (optional)*) – The ‘height space’ (space separating two subplots vertically), in inches. Default is 0.25.

• `wspace` (*float (optional)*) – The ‘width space’ (space separating two subplots horizontally), in inches. Default is 0.25.

• `cbar_width` (*float (optional)*) – The width of the colorbar in inches. Default is 0.25.

• `aspects` (*list of lists (optional)*) – Define the aspect ratio of individual subplots. List of lists, each sub-ist having the format [[row, col], aspect]. The figure will expand vertically to accommodate the defined aspect ratio. Aspects are V/H so aspects larger than 1 will
be taller than wide and vice-versa for aspects smaller than 1. You may only define the aspect for one subplot in each row. If no aspect is defined for a particular row, the leftmost subplot will have an aspect of `default_aspect`. Default is given by `default_aspect` kwarg.

- **default_aspect** *(number (optional)) – Default aspect of left-most column, if no aspect is defined for a given row.*

Returns

(WrightTools.artists.Figure, WrightTools.artists.GridSpec). GridSpec contains SubplotSpec objects that can have axes placed into them. The SubplotSpec objects can be accessed through indexing: [row, col]. Slicing works, for example `cax = plt.subplot(gs[i, -1])`. See matplotlib gridspec documentation for more information.

Return type  **tuple**

Notes

To ensure the margins work as expected, save the fig with the same margins (`pad_inches`) as specified in this function. Common savefig call: `plt.savefig(plt.savefig(output_path, dpi=300, transparent=True, pad_inches=1))`

See also:

- **wt.artists.plot_margins()** Plot lines to visualize the figure edges, margins, and centers. For debug and design purposes.
- **wt.artists.subplots_adjust()** Enforce margins for figure generated elsewhere.

WrightTools.artists.diagonal_line

WrightTools.artists.diagonal_line *(xi=None, yi=None, *, ax=None, c=None, ls=None, lw=None, zorder=3)*

Plot a diagonal line.

Parameters

- **xi** *(1D array-like (optional)) – The x axis points. If None, taken from axis limits. Default is None.*
- **yi** *(1D array-like) – The y axis points. If None, taken from axis limits. Default is None.*
- **ax** *(axis (optional)) – Axis to plot on. If none is supplied, the current axis is used.*
- **c** *(string (optional)) – Line color. Default derives from rcParams grid color.*
- **ls** *(string (optional)) – Line style. Default derives from rcParams linestyle.*
- **lw** *(float (optional)) – Line width. Default derives from rcParams linewidth.*
- **zorder** *(number (optional)) – Matplotlib zorder. Default is 3.*

Returns  **The plotted line.**

Return type  **matplotlib.lines.Line2D object**
WrightTools Documentation, Release 3.2.7

WrightTools.artists.get_color_cycle

WrightTools.artists.get_color_cycle(n, cmap='rainbow', rotations=3)
Get a list of RGBA colors following a colormap.
Useful for plotting lots of elements, keeping the color of each unique.

Parameters
- n (integer) – The number of colors to return.
- cmap (string (optional)) – The colormap to use in the cycle. Default is rainbow.
- rotations (integer (optional)) – The number of times to repeat the colormap over the cycle. Default is 3.

Returns List of RGBA lists.
Return type list

WrightTools.artists.get_scaled_bounds

WrightTools.artists.get_scaled_bounds(ax, position, *, distance=0.1, factor=200)
Get scaled bounds.

Parameters
- ax (Axes object) – Axes object.
- position ({'UL', 'LL', 'UR', 'LR'}) – Position.
- distance (number (optional)) – Distance. Default is 0.1.
- factor (number (optional)) – Factor. Default is 200.

Returns (h_scaled, v_scaled), [va, ha]
Return type ([h_scaled, v_scaled], [va, ha])

WrightTools.artists.grayify_cmap

WrightTools.artists.grayify_cmap(cmap)
Return a grayscale version of the colormap.

Source

WrightTools.artists.interact2D

WrightTools.artists.interact2D(data, xaxis=0, yaxis=1, channel=0, local=False, verbose=True)
Interactive 2D plot of the dataset. Side plots show x and y projections of the slice (shaded gray). Left clicks on the main axes draw 1D slices on side plots at the coordinates selected. Right clicks remove the 1D slices. For 3+ dimensional data, sliders below the main axes are used to change which slice is viewed.

Parameters
- data (WrightTools.Data object) – Data to plot.
- xaxis (string, integer, or data.Axis object (optional)) – Expression or index of x axis. Default is 0.
• **yaxis** *(string, integer, or data.Axis object (optional)) – Expression or index of y axis. Default is 1.*

• **channel** *(string, integer, or data.Channel object (optional)) – Name or index of channel to plot. Default is 0.*

• **local** *(boolean (optional)) – Toggle plotting locally. Default is False.*

• **verbose** *(boolean (optional)) – Toggle talkback. Default is True.*

**WrightTools.artists.overline_colors**

```python
WrightTools.artists.overline_colors = ['#CCFF00', '#FE4EDA', '#FF6600', '#00FFBF', '#00B7EB']
```

- `list()` -> new empty list
- `list(iterable)` -> new list initialized from iterable’s items

**WrightTools.artists.pcolor_helper**

```python
WrightTools.artists.pcolor_helper(xi, yi, zi=None)
```

Prepare a set of arrays for plotting using `pcolor`.

The return values are suitable for feeding directly into `matplotlib.pcolor` such that the pixels are properly centered.

**Parameters**

- **xi** *(1D or 2D array-like)* – Array of X-coordinates.
- **yi** *(1D or 2D array-like)* – Array of Y-coordinates.
- **zi** *(2D array (optional, deprecated)) – If zi is not None, it is returned unchanged in the output.*

**Returns**

- **X** *(2D ndarray)* – X dimension for pcolor
- **Y** *(2D ndarray)* – Y dimension for pcolor
- **zi** *(2D ndarray)* – if zi parameter is not None, returns zi parameter unchanged

**WrightTools.artists.plot_colorbar**

```python
WrightTools.artists.plot_colorbar(cax=None, cmap='default', ticks=None, clim=None, vlim=None, label=None, tick_fontsize=14, label_fontsize=18, decimals=None, orientation='vertical', ticklocation='auto', extend='neither', extendfrac=None, extendrect=False)
```

Easily add a colormap to an axis.

**Parameters**

- **cax** *(matplotlib axis (optional)) – The axis to plot the colorbar on. Finds the current axis if none is given.*
- **cmap** *(string or LinearSegmentedColormap (optional)) – The colormap to fill the colorbar with. Strings map as keys to the WrightTools colormaps dictionary. Default is `default`.*
- **ticks** *(1D array-like (optional)) – Ticks. Default is None.*
• **clim** *(two element list (optional, deprecated))* – The true limits of the colorbar, in the same units as ticks. If None, stretches the colorbar over the limits of ticks. Default is None. Deprecated: Use **vlim** directly instead.

• **vlim** *(two element list-like (optional))* – The limits of the displayed colorbar, in the same units as ticks. If None, displays over clim. Default is None.

• **label** *(str (optional))* – Label. Default is None.

• **tick_fontsize** *(number (optional))* – Fontsize. Default is 14.

• **label_fontsize** *(number (optional))* – Label fontsize. Default is 18.

• **decimals** *(integer (optional))* – Number of decimals to appear in tick labels. Default is None (best guess).

• **orientation** *({'vertical', 'horizontal'} (optional))* – Colorbar orientation. Default is vertical.

• **ticklocation** *({'auto', 'left', 'right', 'top', 'bottom'} (optional))* – Tick location. Default is auto.

• **extend** *({'neither', 'both', 'min', 'max'} (optional))* – If not ‘neither’, make pointed end(s) for out-of-range values. These are set for a given colormap using the colormap set_under and set_over methods.

• **extendfrac** *(None, 'auto', length, lengths) (optional))* – If set to None, both the minimum and maximum triangular colorbar extensions have a length of 5% of the interior colorbar length (this is the default setting). If set to ‘auto’, makes the triangular colorbar extensions the same lengths as the interior boxes (when spacing is set to ‘uniform’) or the same lengths as the respective adjacent interior boxes (when spacing is set to ‘proportional’). If a scalar, indicates the length of both the minimum and maximum triangular colorbar extensions as a fraction of the interior colorbar length. A two-element sequence of fractions may also be given, indicating the lengths of the minimum and maximum colorbar extensions respectively as a fraction of the interior colorbar length.

• **extendrect** *(bool (optional))* – If False the minimum and maximum colorbar extensions will be triangular (the default). If True the extensions will be rectangular.

**Returns** The created colorbar.

**Return type** matplotlib.colorbar.ColorbarBase object

---

**WrightTools.artists.plot_colormap_components**

WrightTools.artists.plot_colormap_components *(cmap)*

Plot the components of a given colormap.

**WrightTools.artists.plot_gridlines**

WrightTools.artists.plot_gridlines *(ax=None, c='grey', lw=1, diagonal=False, zorder=2, makegrid=True)*

Plot dotted gridlines onto an axis.

**Parameters**

• **ax** *(matplotlib AxesSubplot object (optional))* – Axis to add gridlines to. If None, uses current axis. Default is None.

• **c** *(matplotlib color argument (optional))* – Gridline color. Default is grey.
• `lw(number (optional))` – Gridline linewidth. Default is 1.
• `diagonal(boolean (optional))` – Toggle inclusion of diagonal gridline. Default is False.
• `zorder(number (optional))` – Zorder of plotted grid. Default is 2.

WrightTools.artists.plot_margins

WrightTools.artists.plot_margins(*, fig=None, inches=1.0, centers=True, edges=True)

Add lines onto a figure indicating the margins, centers, and edges.

Useful for ensuring your figure design scripts work as intended, and for laying out figures.

Parameters

• `fig(matplotlib.figure.Figure object (optional))` – The figure to plot onto. If None, gets current figure. Default is None.
• `inches(float (optional))` – The size of the figure margin, in inches. Default is 1.
• `centers(bool (optional))` – Toggle for plotting lines indicating the figure center. Default is True.
• `edges(bool (optional))` – Toggle for plotting lines indicating the figure edges. Default is True.

WrightTools.artists.quick1D

WrightTools.artists.quick1D(data, axis=0, at={}, channel=0, *, local=False, autosave=False, save_directory=None, fname=None, verbose=True)

Quickly plot 1D slice(s) of data.

Parameters

• `data(WrightTools.Data object)` – Data to plot.
• `axis(string or integer (optional))` – Expression or index of axis. Default is 0.
• `at(dictionary (optional))` – Dictionary of parameters in non-plotted dimensions. If not provided, plots will be made at each coordinate.
• `channel(string or integer (optional))` – Name or index of channel to plot. Default is 0.
• `local(boolean (optional))` – Toggle plotting locally. Default is False.
• `autosave(boolean (optional))` – Toggle autosave. Default is False.
• `save_directory(string (optional))` – Location to save image(s). Default is None (auto-generated).
• `fname(string (optional))` – File name. If None, data name is used. Default is None.
• `verbose(boolean (optional))` – Toggle talkback. Default is True.

Returns List of saved image files (if any).
Return type list of strings
WrightTools.artists.quick2D

WrightTools.artists.quick2D (data, xaxis=0, yaxis=1, at={}, channel=0, *, contours=0, pixelated=True, dynamic_range=False, local=False, contours_local=True, autosave=False, save_directory=None, fname=None, verbose=True)

Quickly plot 2D slice(s) of data.

Parameters

- **data** (WrightTools.Data object.) – Data to plot.
- **xaxis** (string or integer (optional)) – Expression or index of horizontal axis. Default is 0.
- **yaxis** (string or integer (optional)) – Expression or index of vertical axis. Default is 1.
- **at** (dictionary (optional)) – Dictionary of parameters in non-plotted dimension(s). If not provided, plots will be made at each coordinate.
- **channel** (string or integer (optional)) – Name or index of channel to plot. Default is 0.
- **contours** (integer (optional)) – The number of black contour lines to add to the plot. Default is 0.
- **pixelated** (boolean (optional)) – Toggle between pcolor and contourf (deulaney) plotting backends. Default is True (pcolor).
- **dynamic_range** (boolean (optional)) – Force the colorbar to use all of its colors. Only changes behavior for signed channels. Default is False.
- **local** (boolean (optional)) – Toggle plotting locally. Default is False.
- **contours_local** (boolean (optional)) – Toggle plotting black contour lines locally. Default is True.
- **autosave** (boolean (optional)) – Toggle autosave. Default is False.
- **save_directory** (string (optional)) – Location to save image(s). Default is None (auto-generated).
- **fname** (string (optional)) – File name. If None, data name is used. Default is None.
- **verbose** (boolean (optional)) – Toggle talkback. Default is True.

Returns List of saved image files (if any).

Return type list of strings
WrightTools.artists.savefig

WrightTools.artists.savefig(path, fig=None, close=True, **kwargs)

Save a figure.

Note, that this method defaults to transparent background (facecolor kwarg) and to 300 dpi.

Parameters

• path (str) – Path to save figure at.
• fig (matplotlib.figure.Figure object (optional)) – The figure to plot onto. If None, gets current figure. Default is None.
• close (bool (optional)) – Toggle closing of figure after saving. Default is True.

Parameters (Keyword) –

• ------------------
• kwags (any) – All additional parameters are passed to the underlying matplotlib savefig call

Returns The full path where the figure was saved.

Return type str

WrightTools.artists.set_ax_labels

WrightTools.artists.set_ax_labels(ax=None, xlabel=None, ylabel=None, xticks=None, yticks=None, label_fontsize=18)

Set all axis labels properties easily.

Parameters

• ax (matplotlib AxesSubplot object (optional)) – Axis to set. If None, uses current axis. Default is None.
• xlabel (None or string (optional)) – x axis label. Default is None.
• ylabel (None or string (optional)) – y axis label. Default is None.
• xticks (None or False or list of numbers) – xticks. If False, ticks are hidden. Default is None.
• yticks (None or False or list of numbers) – yticks. If False, ticks are hidden. Default is None.
• label_fontsize (number) – Fontsize of label. Default is 18.

See also:

set_fig_labels()
WrightTools.artists.set_ax_spines

WrightTools.artists.set_ax_spines(ax=None, *, c='k', lw=3, zorder=10)

Easily set the properties of all four axis spines.

Parameters

- **ax** (matplotlib AxesSubplot object (optional)) – Axis to set. If None, uses current axis. Default is None.
- **c** (any matplotlib color argument (optional)) – Spine color. Default is k.
- **lw** (number (optional)) – Spine linewidth. Default is 3.
- **zorder** (number (optional)) – Spine zorder. Default is 10.

WrightTools.artists.set_fig_labels

WrightTools.artists.set_fig_labels(fig=None, xlabel=None, ylabel=None, xticks=None, yticks=None, title=None, row=-1, col=0, label_fontsize=18, title_fontsize=20)

Set all axis labels of a figure simultaneously.

Only plots ticks and labels for edge axes.

Parameters

- **fig** (matplotlib.figure.Figure object (optional)) – Figure to set labels of. If None, uses current figure. Default is None.
- **xlabel** (None or string (optional)) – x axis label. Default is None.
- **ylabel** (None or string (optional)) – y axis label. Default is None.
- **xticks** (None or False or list of numbers (optional)) – xticks. If False, ticks are hidden. Default is None.
- **yticks** (None or False or list of numbers (optional)) – yticks. If False, ticks are hidden. Default is None.
- **title** (None or string (optional)) – Title of figure. Default is None.
- **row** (integer or slice (optional)) – Row to label. Default is -1. If slice, step is ignored.
- **col** (integer or slice (optional)) – col to label. Default is 0. If slice, step is ignored.
- **label_fontsize** (number (optional)) – Fontsize of label. Default is 18.
- **title_fontsize** (number (optional)) – Fontsize of title. Default is 20.

See also:

set_ax_labels()
**WrightTools.artists.stitch_to_animation**

WrightTools.artists.stitch_to_animation(images, outpath=None, *, duration=0.5, palettesize=256, verbose=True)

Stitch a series of images into an animation.

Currently supports animated gifs, other formats coming as needed.

**Parameters**

- **images** *(list of strings)* -- Filepaths to the images to stitch together, in order of appearance.

- **outpath** *(string (optional))* -- Path of output, including extension. If None, bases output path on path of first path in images. Default is None.

- **duration** *(number or list of numbers (optional))* -- Duration of (each) frame in seconds. Default is 0.5.

- **palettesize** *(int (optional))* -- The number of colors in the resulting animation. Input is rounded to the nearest power of 2. Default is 1024.

- **verbose** *(bool (optional))* -- Toggle talkback. Default is True.

**WrightTools.artists.subplots_adjust**

WrightTools.artists.subplots_adjust(fig=None, inches=1)

Enforce margin to be equal around figure, starting at subplots.

**Note:** You probably should be using wt.artists.create_figure instead.

**See also:**

- wt.artists.plot_margins() Visualize margins, for debugging / layout.
- wt.artists.create_figure() Convinience method for creating well-behaved figures.

### 1.10.2 WrightTools.collection package

Collection class and associated.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collection</td>
<td>Nestable Collection of Data objects.</td>
</tr>
<tr>
<td>from_Cary(filepath[, name, parent, verbose])</td>
<td>Create a collection object from a Cary UV VIS absorbance file.</td>
</tr>
<tr>
<td>from_directory(filepath, from_methods, *[, …])</td>
<td>Create a WrightTools Collection from a directory of source files.</td>
</tr>
</tbody>
</table>
**WrightTools.collection.Collection**

```python
class WrightTools.collection.Collection(file=None, parent=None, name=None, **kwargs):
    Bases: WrightTools._group.Group

    Nestable Collection of Data objects.

    __init__(file=None, parent=None, name=None, **kwargs)
        Create a new Group object by binding to a low-level GroupID.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close()</td>
<td>Close the file that contains the Group.</td>
</tr>
<tr>
<td>copy([parent, name, verbose])</td>
<td>Create a copy under parent.</td>
</tr>
<tr>
<td>create_collection([name, position])</td>
<td>Create a new child collection.</td>
</tr>
<tr>
<td>create_data([name, position])</td>
<td>Create a new child data.</td>
</tr>
<tr>
<td>flush()</td>
<td>Ensure contents are written to file.</td>
</tr>
<tr>
<td>print_tree([depth, verbose])</td>
<td>Print a ascii-formatted tree representation of the collection contents.</td>
</tr>
<tr>
<td>save([filepath, overwrite, verbose])</td>
<td>Save as root of a new file.</td>
</tr>
</tbody>
</table>

**WrightTools.collection.Collection.close**

`Collection.close()`

Close the file that contains the Group.

All groups which are in the file will be closed and removed from the _instances dictionaries. Tempfiles, if they exist, will be removed

**WrightTools.collection.Collection.copy**

`Collection.copy(parent=None, name=None, verbose=True)`

Create a copy under parent.

All children are copied as well.

**Parameters**

- `parent` (*WrightTools Collection (optional)*) – Parent to copy within. If None, copy is created in root of new tempfile. Default is None.
- `name` (*string (optional)*) – Name of new copy at destination. If None, the current natural name is used. Default is None.
- `verbose` (*boolean (optional)*) – Toggle talkback. Default is True.

**Returns**

Created copy.

**Return type**

Group
### WrightTools.collection.Collection.create_collection

**Function**

```
Collection.create_collection(name='collection', position=None, **kwargs)
```

**Description**

Create a new child collection.

**Parameters**

- **name** *(string)* – Unique identifier.
- **position** *(integer (optional))* – Location to insert. Default is None (append).
- **kwargs** – Additional arguments to child collection instantiation.

**Returns**

New child.

**Return type**

WrightTools Collection

### WrightTools.collection.Collection.create_data

**Function**

```
Collection.create_data(name='data', position=None, **kwargs)
```

**Description**

Create a new child data.

**Parameters**

- **name** *(string)* – Unique identifier.
- **position** *(integer (optional))* – Location to insert. Default is None (append).
- **kwargs** – Additional arguments to child data instantiation.

**Returns**

New child.

**Return type**

WrightTools Data

### WrightTools.collection.Collection.flush

**Function**

```
Collection.flush()
```

**Description**

Ensure contents are written to file.

### WrightTools.collection.Collection.print_tree

**Function**

```
Collection.print_tree(depth=9, *, verbose=False)
```

**Description**

Print an ascii-formatted tree representation of the collection contents.

**Parameters**

- **depth** *(integer (optional))* – Number of layers to include in the tree. Default is 9.
- **verbose** *(boolean (optional))* – Toggle inclusion of extra information. Default is True.
**WrightTools.collection.Collection.save**

`Collection.save(filepath=None, overwrite=False, verbose=True)`

Save as root of a new file.

**Parameters**

- **filepath** (*Path-like object (optional)*) – Filepath to write. If None, file is created using `natural_name`.
- **overwrite** (*boolean (optional)*) – Toggle overwrite behavior. Default is False.
- **verbose** (*boolean (optional)*) – Toggle talkback. Default is True

**Returns** Written filepath.

**Return type** `str`

**Attributes**

<table>
<thead>
<tr>
<th>attr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attrs</code></td>
<td>Attributes attached to this object</td>
</tr>
<tr>
<td><code>class_name</code></td>
<td><em>Collection</em></td>
</tr>
<tr>
<td><code>created</code></td>
<td>Return a File instance associated with this object</td>
</tr>
<tr>
<td><code>file</code></td>
<td>Filepath to write. If None, file is created using <code>natural_name</code></td>
</tr>
<tr>
<td><code>fullpath</code></td>
<td>Filepath to write. If None, file is created using <code>natural_name</code></td>
</tr>
<tr>
<td><code>item_names</code></td>
<td>Item names.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>Return the full name of this object.</td>
</tr>
<tr>
<td><code>natural_name</code></td>
<td>Natural name.</td>
</tr>
<tr>
<td><code>parent</code></td>
<td>Parent.</td>
</tr>
</tbody>
</table>

**WrightTools.collection.Collection.attrs**

`property Collection.attrs`

Attributes attached to this object

**WrightTools.collection.Collection.class_name**

`Collection.class_name = 'Collection'`
WrightTools.collection.Collection.created

Property Collection.created

WrightTools.collection.Collection.file

Property Collection.file

Return a File instance associated with this object

WrightTools.collection.Collection.fullpath

Property Collection.fullpath

file and internal structure.

Type Full path

WrightTools.collection.Collection.item_names

Property Collection.item_names

Item names.

WrightTools.collection.Collection.name

Property Collection.name

Return the full name of this object. None if anonymous.

WrightTools.collection.Collection.natural_name

Property Collection.natural_name

Natural name.

WrightTools.collection.Collection.parent

Property Collection.parent

Parent.

WrightTools.collection.from_Cary

WrightTools.collection.from_Cary(filepath, name=None, parent=None, verbose=True)

Create a collection object from a Cary UV VIS absorbance file.

We hope to support as many Cary instruments and datasets as possible. This function has been tested with data collected on a Cary50 UV/VIS spectrometer. If any alternate instruments are found not to work as expected, please submit a bug report on our issue tracker.
```python
>>> import WrightTools as wt
>>> from WrightTools import datasets
>>> p = datasets.Cary.CuPCtS_H2O_vis
>>> data = wt.collection.from_Cary(p)[0]
>>> wt.artists.quick1D(data)
```

```
Parameters

- **filepath** *(path-like)* – Path to Cary output file (.csv).
- **parent** *(WrightTools.Collection)* – A collection object in which to place a collection of Data objects.
- **verbose** *(boolean (optional)) –* Toggle talkback. Default is True.

Returns

New data object.

Return type  data
```
WrightTools.collection.from_directory

WrightTools.collection.from_directory(filepath, from_methods, *, name=None, parent=None, verbose=True)

Create a WrightTools Collection from a directory of source files.

Parameters

- **filepath** (*path-like*) – Path to the directory on the file system
- **from_methods** (*dict<str, callable>*) – Dictionary which maps patterns (using Unix-like glob wildcard patterns) to functions which take a filepath, plus the keyword arguments ['name', 'parent', and 'verbose']. (e.g. most from_<kind> methods within WrightTools) The value can be **None** which results in that item being ignored. The first matching pattern encountered will be used. Therefore, if multiple patterns will match the same file, use and **OrderedDict**. Patterns are matched on the file name level, not using the full path.

Keyword Arguments

- **name** (*str*) – Name to use for the root data object. Default is the directory name.
- **parent** (*Collection*) – Parent collection to insert the directory structure into. Default is a new collection in temp file.
- **verbose** (*bool*) – Print information as objects are created. Passed to the functions.

Examples

```python
>>> from_dict = {'*.data':wt.data.from_PyCMDS,
...               '*.csv':wt.collections.from_Cary,
...               'unused':None,
...               }
>>> col = wt.collection.from_directory('path/to/folder', from_dict)
```

1.10.3 WrightTools.data package

Data class and associated.

<table>
<thead>
<tr>
<th>Data(*args, **kwargs)</th>
<th>Multidimensional dataset.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Axis(parent, expression[, units])</td>
<td>Axis class.</td>
</tr>
<tr>
<td>Channel(parent, id, *[units, null, ...])</td>
<td>Channel.</td>
</tr>
<tr>
<td>Constant(parent, expression[, units, ...])</td>
<td>Constant class.</td>
</tr>
<tr>
<td>Variable(parent, id[, units])</td>
<td>Variable.</td>
</tr>
<tr>
<td>join(datas, *[atol, rtol, name, parent, ...])</td>
<td>Join a list of data objects together.</td>
</tr>
<tr>
<td>from_BrunoldrRaman(filepath[, name, parent, ...])</td>
<td>Create a data object from the Brunold rRaman instrument.</td>
</tr>
<tr>
<td>from_COLORS(filepaths[, name, cols, ...])</td>
<td>Create data object from COLORS file(s).</td>
</tr>
<tr>
<td>from_JASCO(filepath[, name, parent, verbose])</td>
<td>Create a data object from JASCO UV-Vis spectrometers.</td>
</tr>
<tr>
<td>from_KENT(filepaths[, name, ignore, ...])</td>
<td>Create data object from KENT file(s).</td>
</tr>
<tr>
<td>from_Aramis(filepath[, name, parent, verbose])</td>
<td>Create a data object from Horiba Aramis ngc binary file.</td>
</tr>
<tr>
<td>from_ocean_optics(filepath[, name, parent, ...])</td>
<td>Create a data object from an Ocean Optics brand spectrometer.</td>
</tr>
<tr>
<td>from_PyCMDS(filepath[, name, parent, ...])</td>
<td>Create a data object from a single PyCMDS output file.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 7 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>from_shimadzu(filepath[, name, parent, verbose])</code></td>
<td>Create a data object from Shimadzu .txt file.</td>
</tr>
<tr>
<td><code>from_Solis(filepath[, name, parent, verbose])</code></td>
<td>Create a data object from Andor Solis software (ascii exports).</td>
</tr>
<tr>
<td><code>from_spcm(filepath[, name, delimiter, ...])</code></td>
<td>Create a data object from a Becker &amp; Hickl spcm file (ASCII-exported, .asc).</td>
</tr>
<tr>
<td><code>from_Tensor27(filepath[, name, parent, verbose])</code></td>
<td>Create a data object from a Tensor27 FTIR file.</td>
</tr>
</tbody>
</table>

**WrightTools.data.Data**

```python
class WrightTools.data.Data(*args, **kwargs)

Bases: WrightTools._group.Group

Multidimensional dataset.
```

___init__(*args, **kwargs) Create a new Group object by binding to a low-level GroupID.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>bring_to_front(channel)</code></td>
<td>Bring a specific channel to the zero-indexed position in channels.</td>
</tr>
<tr>
<td><code>chop(*args[, at, parent, verbose])</code></td>
<td>Divide the dataset into its lower-dimensionality components.</td>
</tr>
<tr>
<td><code>clear()</code></td>
<td>Close the file that contains the Group.</td>
</tr>
<tr>
<td><code>close()</code></td>
<td>Close the file that contains the Group.</td>
</tr>
<tr>
<td><code>collapse(axis[, method])</code></td>
<td>Collapse the dataset along one axis, adding lower rank channels.</td>
</tr>
<tr>
<td><code>convert(destination_units, *[...])</code></td>
<td>Convert all compatible axes and constants to given units.</td>
</tr>
<tr>
<td><code>copy([parent, name, verbose])</code></td>
<td>Create a copy under parent.</td>
</tr>
<tr>
<td><code>create_channel(name[, values, shape, units, ...])</code></td>
<td>Append a new channel.</td>
</tr>
<tr>
<td><code>create_constant(expression, *[,...])</code></td>
<td>Append a constant to the stored list.</td>
</tr>
<tr>
<td><code>create_dataset(name[, shape, dtype, data])</code></td>
<td>Create a new HDF5 dataset</td>
</tr>
<tr>
<td><code>create_group(name[, track_order])</code></td>
<td>Create and return a new subgroup.</td>
</tr>
<tr>
<td><code>create_variable(name[, values, shape, ...])</code></td>
<td>Add new child variable.</td>
</tr>
<tr>
<td><code>flush()</code></td>
<td>Ensure contents are written to file.</td>
</tr>
<tr>
<td><code>get(name[, default, getclass, getlink])</code></td>
<td>Get the coordinates, in units, of the minimum in a channel.</td>
</tr>
<tr>
<td><code>get_nadir([channel])</code></td>
<td>Get the coordinates, in units, of the minimum in a channel.</td>
</tr>
<tr>
<td><code>get_zenith([channel])</code></td>
<td>Get the coordinates, in units, of the maximum in a channel.</td>
</tr>
<tr>
<td><code>gradient(axis[, ..., channel])</code></td>
<td>Compute the gradient along one axis.</td>
</tr>
<tr>
<td><code>heal([channel, method, fill_value, verbose])</code></td>
<td>Remove nans from channel using interpolation.</td>
</tr>
<tr>
<td><code>level(channel, axis, npts, *[,...])</code></td>
<td>Subtract the average value of npts at the edge of a given axis.</td>
</tr>
<tr>
<td><code>map_variable(variable, points[, ...])</code></td>
<td>Map points of an axis to new points using linear interpolation.</td>
</tr>
<tr>
<td><code>moment(axis[, channel, moment, resultant])</code></td>
<td>Take the nth moment the dataset along one axis, adding lower rank channels.</td>
</tr>
</tbody>
</table>

Continued on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>offset</code></td>
<td>Offset one axis based on another axis’ values.</td>
</tr>
<tr>
<td><code>print_tree</code></td>
<td>Print a ascii-formatted tree representation of the data contents.</td>
</tr>
<tr>
<td><code>prune</code></td>
<td>Remove unused variables and (optionally) channels from the Data object.</td>
</tr>
<tr>
<td><code>remove_channel</code></td>
<td>Remove channel from data.</td>
</tr>
<tr>
<td><code>remove_constant</code></td>
<td>Remove a constant from the stored list.</td>
</tr>
<tr>
<td><code>remove_variable</code></td>
<td>Remove variable from data.</td>
</tr>
<tr>
<td><code>rename_channels</code></td>
<td>Rename a set of channels.</td>
</tr>
<tr>
<td><code>rename_variables</code></td>
<td>Rename a set of variables.</td>
</tr>
<tr>
<td><code>save</code></td>
<td>Save as root of a new file.</td>
</tr>
<tr>
<td><code>set_constants</code></td>
<td>Set the constants associated with the data.</td>
</tr>
<tr>
<td><code>share_nans</code></td>
<td>Share not-a-numbers between all channels.</td>
</tr>
<tr>
<td><code>smooth</code></td>
<td>Smooth a channel using an n-dimensional kaiser window.</td>
</tr>
<tr>
<td><code>split</code></td>
<td>Split the data object along a given expression, in units.</td>
</tr>
<tr>
<td><code>transform</code></td>
<td>Transform the data.</td>
</tr>
<tr>
<td><code>update</code></td>
<td>If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v</td>
</tr>
<tr>
<td><code>zoom</code></td>
<td>Zoom the data array using spline interpolation of the requested order.</td>
</tr>
</tbody>
</table>

**WrightTools.data.Data.bring_to_front**

Data.bring_to_front(channel)

Bring a specific channel to the zero-indexed position in channels.

All other channels get pushed back but remain in order.

**Parameters**

- **channel** *(int or str)* – Channel index or name.

**WrightTools.data.Data.chop**

Data.chop(*args, at={}, parent=None, verbose=True) → WrightTools.collection._collection.Collection

Divide the dataset into its lower-dimensionality components.

**Parameters**

- **axis** *(str or int (args))* – Axes of the returned data objects. Strings refer to the names of axes in this object, integers refer to their index. Provide multiple axes to return multidimensional data objects.
- **at** *(dict (optional))* – Choice of position along an axis. Keys are axis names, values are lists [position, input units]. If exact position does not exist, the closest valid position is used.
- **parent** *(WrightTools Collection instance (optional))* – Collection to place the new “chop” collection within. Default is None (new parent).
• **verbose**(bool *(optional*)) – Toggle talkback. Default is True.

**Returns** Collection of chopped data objects.

**Return type** WrightTools Collection

**Examples**

```python
>>> data.axis_names
['d2', 'w1', 'w2']
```

Get all w1 wigners.

```python
>>> datas = data.chop('d2', 'w1')
>>> len(datas)
51
```

Get 2D frequency at d2=0 fs.

```python
>>> datas = data.chop('w1', 'w2', at={'d2': [0, 'fs']})
>>> len(datas)
0
>>> datas[0].axis_names
['w1', 'w2']
>>> datas[0].d2[:]
0.
```

**See also:**

- `collapse()` Collapse the dataset along one axis.
- `split()` Split the dataset while maintaining its dimensionality.

**WrightTools.data.Data.clear**

Data.**clear**() → None. Remove all items from D.

**WrightTools.data.Data.close**

Data.**close**()

Close the file that contains the Group.

All groups which are in the file will be closed and removed from the _instances dictionaries. Tempfiles, if they exist, will be removed
**Data**.**collapse**(axis, method='sum')

Collapse the dataset along one axis, adding lower rank channels.

- New channels have names `<channel name>_<axis name>_<method>`.

**Parameters**

- **axis** (*int or str*) – The axis to collapse along. If given as an integer, the axis in the underlying array is used. If given as a string, the axis must exist, and be a 1D array-aligned axis. (i.e. have a shape with a single value which is not 1) The axis to collapse along is inferred from the shape of the axis.

- **method** (*{'average', 'sum', 'max', 'min'} (optional)*) – The method of collapsing the given axis. Method may also be list of methods corresponding to the channels of the object. Default is sum. NaNs are ignored. Can also be a list, allowing for different treatment for varied channels. In this case, None indicates that no change to that channel should occur.

**See also:**

- **chop()** – Divide the dataset into its lower-dimensionality components.
- **split()** – Split the dataset while maintaining its dimensionality.
- **moment()** – Take the moment along a particular axis

---

**Data**.**convert**(destination_units, *, convert_variables=False, verbose=True)

Convert all compatible axes and constants to given units.

**Parameters**

- **destination_units** (*str*) – Destination units.

- **convert_variables** (*boolean (optional)*) – Toggle conversion of stored arrays. Default is False

- **verbose** (*bool (optional)*) – Toggle talkback. Default is True.

**See also:**

- **Axis**.**convert()** – Convert a single axis object to compatible units. Call on an axis object in data.axes.

---

**Data**.**copy**(parent=None, name=None, verbose=True)

Create a copy under parent.

All children are copied as well.

**Parameters**

- **parent** (*WrightTools Collection (optional)*) – Parent to copy within. If None, copy is created in root of new tempfile. Default is None.

- **name** (*string (optional)*) – Name of new copy at destination. If None, the current natural name is used. Default is None.
• **verbose** *(boolean (optional)) – Toggle talkback. Default is True.*

**Returns** Created copy.

**Return type** Group

### WrightTools.data.Data.create_channel

Data.create_channel *(name, values=None, *, shape=None, units=None, dtype=None, **kwargs) → WrightTools.data._channel.Channel*

Append a new channel.

**Parameters**

- **name** *(string) – Unique name for this channel.*
- **values** *(array (optional)) – Array. If None, an empty array equaling the data shape is created. Default is None.*
- **shape** *(tuple of int) – Shape to use. Must broadcast with the full shape. Only used if values is None. Default is the full shape of self.*
- **units** *(string (optional)) – Channel units. Default is None.*
- **dtype** *(numpy.dtype (optional)) – dtype to use for dataset, default is np.float64. Only used if values is None.*
- **kwargs** *(dict) – Additional keyword arguments passed to Channel instantiation.*

**Returns** Created channel.

**Return type** Channel

### WrightTools.data.Data.create_constant

Data.create_constant *(expression, *, verbose=True)*

Append a constant to the stored list.

**Parameters**

- **expression** *(str) – Expression for the new constant.*
- **verbose** *(boolean (optional)) – Toggle talkback. Default is True*

**See also:**

*set_constants() – Remove and replace all constants.*

*remove_constant() – Remove an individual constant.*
WrightTools.data.Data.create_dataset

Data.create_dataset (name, shape=None, dtype=None, data=None, **kwds)
Create a new HDF5 dataset

- **name** Name of the dataset (absolute or relative). Provide None to make an anonymous dataset.
- **shape** Dataset shape. Use "()" for scalar datasets. Required if "data" isn’t provided.
- **dtype** Numpy dtype or string. If omitted, dtype('f') will be used. Required if "data" isn’t provided; otherwise, overrides data array’s dtype.
- **data** Provide data to initialize the dataset. If used, you can omit shape and dtype arguments.

Keyword-only arguments:

- **chunks** (Tuple) Chunk shape, or True to enable auto-chunking.
- **maxshape** (Tuple) Make the dataset resizable up to this shape. Use None for axes you want to be unlim-
  ited.
- **compression** (String or int) Compression strategy. Legal values are ‘gzip’, ‘szip’, ‘lzf’. If an integer
  in range(10), this indicates gzip compression level. Otherwise, an integer indicates the number of a
  dynamically loaded compression filter.
- **compression_opts** Compression settings. This is an integer for gzip, 2-tuple for szip, etc. If specifying a
  dynamically loaded compression filter number, this must be a tuple of values.
- **scaleoffset** (Integer) Enable scale/offset filter for (usually) lossy compression of integer or floating-point
  data. For integer data, the value of scaleoffset is the number of bits to retain (pass 0 to let HDF5
  determine the minimum number of bits necessary for lossless compression). For floating point data,
  scaleoffset is the number of digits after the decimal place to retain; stored values thus have absolute
  error less than 0.5*10**(-scaleoffset).
- **shuffle** (T/F) Enable shuffle filter.
- **fletcher32** (T/F) Enable fletcher32 error detection. Not permitted in conjunction with the scale/offset
  filter.
- **fillvalue** (Scalar) Use this value for uninitialized parts of the dataset.
- **track_times** (T/F) Enable dataset creation timestamps.
- **track_order** (T/F) Track attribute creation order if True. If omitted use global default
  h5.get_config().track_order.
- **external** (List of tuples) Sets the external storage property, thus designating that the dataset will be stored
  in one or more non-HDF5 file(s) external to the HDF5 file. Adds each listed tuple of (file[, offset[, size]]) to the dataset’s list of external files.

WrightTools.data.Data.create_group

Data.create_group (name, track_order=None)
Create and return a new subgroup.

Name may be absolute or relative. Fails if the target name already exists.

- **track_order** Track dataset/group/attribute creation order under this group if True. If None use global
  default h5.get_config().track_order.
WrightTools Documentation, Release 3.2.7

WrightTools.data.Data.create_variable

Data.create_variable(name, values=None, *, shape=None, units=None, dtype=None, **kwargs) → WrightTools.data._variable.Variable

Add new child variable.

Parameters

- **name** (string) – Unique identifier.
- **values** (array-like (optional)) – Array to populate variable with. If None, an variable will be filled with NaN. Default is None.
- **shape** (tuple of int) – Shape to use. must broadcast with the full shape. Only used if values is None. Default is the full shape of self.
- **units** (string (optional)) – Variable units. Default is None.
- **dtype** (numpy.dtype (optional)) – dtype to use for dataset, default is np.float64. Only used if values is None.
- **kwargs** – Additional kwargs to variable instantiation.

Returns

New child variable.

Return type

WrightTools Variable

WrightTools.data.Data.flush

Data.flush()

Ensure contents are written to file.

WrightTools.data.Data.get

Data.get(name, default=None, getclass=False, getlink=False)

Retrieve an item or other information.

“name” given only:  Return the item, or “default” if it doesn’t exist

“getclass” is True: Return the class of object (Group, Dataset, etc.), or “default” if nothing with that name exists

“getlink” is True: Return HardLink, SoftLink or ExternalLink instances. Return “default” if nothing with that name exists.

“getlink” and “getclass” are True: Return HardLink, SoftLink and ExternalLink classes. Return “default” if nothing with that name exists.

Example:

```python
>>> cls = group.get('foo', getclass=True)
>>> if cls == SoftLink:
```

64 Chapter 1. Contents
WrightTools.data.Data.get_nadir

Data.get_nadir(channel=0) → tuple
Get the coordinates, in units, of the minimum in a channel.

Parameters channel (int or str (optional)) – Channel. Default is 0.

Returns Coordinates in units for each axis.

Return type generator of numbers

WrightTools.data.Data.get_zenith

Data.get_zenith(channel=0) → tuple
Get the coordinates, in units, of the maximum in a channel.

Parameters channel (int or str (optional)) – Channel. Default is 0.

Returns Coordinates in units for each axis.

Return type generator of numbers

WrightTools.data.Data.gradient

Data.gradient(axis, *, channel=0)
Compute the gradient along one axis.
New channels have names <channel name>_<axis name>_gradient.

Parameters

• axis (int or str) – The axis to differentiate along. If given as an integer, the axis in
the underlying array is used, and unitary spacing is assumed. If given as a string, the axis
must exist, and be a 1D array-aligned axis. (i.e. have a shape with a single value which is
not 1) The axis to collapse along is inferred from the shape of the axis.

• channel (int or str) – The channel to differentiate. Default is the first channel.

WrightTools.data.Data.heal

Data.heal(channel=0, method='linear', fill_value=nan, verbose=True)
Remove nans from channel using interpolation.

Parameters

• channel (int or str (optional)) – Channel to heal. Default is 0.

• method (['linear', 'nearest', 'cubic'] (optional)) – The interpolation
method. Note that cubic interpolation is only possible for 1D and 2D data. See
griddata for more information. Default is linear.

• fill_value (number-like (optional)) – The value written to pixels that can-
not be filled by interpolation. Default is nan.

• verbose (bool (optional)) – Toggle talkback. Default is True.
Note: Healing may take several minutes for large datasets. Interpolation time goes as nearest, linear, then cubic.

**WrightTools.data.Data.level**

Data.level(*channel, axis, npts, *, verbose=True*)

Subtract the average value of npts at the edge of a given axis.

Parameters
- **channel (int or str)** – Channel to level.
- **axis (int)** – Axis to level along.
- **npts (int)** – Number of points to average for each slice. Positive numbers take points at leading indicies and negative numbers take points at trailing indicies.
- **verbose (bool (optional))** – Toggle talkback. Default is True.

**WrightTools.data.Data.map_variable**

Data.map_variable(*variable, points, input_units='same', *, name=None, parent=None, verbose=True*) → WrightTools.data._data.Data

Map points of an axis to new points using linear interpolation. Out-of-bounds points are written nan.

Parameters
- **variable (string)** – The variable to map onto.
- **points (array-like or int)** – If array, the new points. If int, new points will have the same limits, with int defining the number of evenly spaced points between.
- **input_units (str (optional))** – The units of the new points. Default is same, which assumes the new points have the same units as the axis.
- **name (string (optional))** – The name of the new data object. If None, generated from natural_name. Default is None.
- **parent (WrightTools.Collection (optional))** – Parent of new data object. If None, data is made at root of a new temporary file.
- **verbose (bool (optional))** – Toggle talkback. Default is True.

Returns New data object.

Return type WrightTools.Data
**WrightTools.data.Data.moment**

```
Data.moment(axis, channel=0, moment=1, *, resultant=None)
```

Take the nth moment the dataset along one axis, adding lower rank channels.

New channels have names `<channel name>_<axis name>_moment_<moment num>`.

Moment 0 is the integral of the slice. Moment 1 is the weighted average or "Center of Mass", normalized by the integral Moment 2 is the variance, the central moment about the center of mass, normalized by the integral Moments 3+ are central moments about the center of mass, normalized by the integral and by the standard deviation to the power of the moment.

Moments, especially higher order moments, are susceptible to noise and baseline. It is recommended when used with real data to use `WrightTools.data.Channel.clip()` in conjunction with moments to reduce effects of noise.

**Parameters**

- **axis** *(int or str)* – The axis to take the moment along. If given as an integer, the axis with that index is used. If given as a string, the axis with that name is used. The axis must exist, and be a 1D array-aligned axis. (i.e. have a shape with a single value which is not 1) The collapsed axis must be monotonic to produce correct results. The axis to collapse along is inferred from the shape of the axis.

- **channel** *(int or str)* – The channel to take the moment. If given as an integer, the channel with that index is used. If given as a string, the channel with that name is used. The channel must have values along the axis (i.e. its shape must not be 1 in the dimension for which the axis is not 1) Default is 0, the first channel.

- **moment** *(int or tuple of int)* – The moments to take. One channel will be created for each number given. Default is 1, the center of mass.

- **resultant** *(tuple of int)* – The resultant shape after the moment operation. By default, it is intuited by the axis along which the moment is being taken. This default only works if that axis is 1D, so resultant is required if a multidimensional axis is passed as the first argument. The requirement of monotonicity applies on a per pixel basis.

**See also:**

- `collapse()` Reduce dimensionality by some mathematical operation
- `clip()` Set values above/below a threshold to a particular value
- `WrightTools.kit.joint_shape()` Useful for setting resultant kwarg based off of axes not collapsed.

**WrightTools.data.Data.offset**

```
Data.offset(points, offsets, along, offset_axis, units='same', offset_units='same', mode='valid', method='linear', verbose=True)
```

Offset one axis based on another axis’ values.

Useful for correcting instrumental artifacts such as zerotune.

**Parameters**

- **points** *(1D array-like)* – Points.
- **offsets** *(1D array-like)* – Offsets.
- **along** *(str or int)* – Axis that points array lies along.
• `offset_axis` *(str or int)* – Axis to offset using offsets.

• `units` *(str (optional))* – Units of points array.

• `offset_units` *(str (optional))* – Units of offsets array.

• `mode` *({'valid', 'full', 'old'} (optional))* – Define how far the new axis will extend. Points outside of valid interpolation range will be written nan.

• `method` *({'linear', 'nearest', 'cubic'} (optional))* – The interpolation method. Note that cubic interpolation is only possible for 1D and 2D data. See `griddata` for more information. Default is linear.

• `verbose` *(bool (optional))* – Toggle talkback. Default is True.

```python
>>> points  # an array of wl points
>>> offsets # an array of dl corrections
>>> data.offset(points, offsets, 'wl', 'dl')
```

**WrightTools.data.Data.print_tree**

Data.

`print_tree` *(*, `verbose=True`)*

Print a ascii-formatted tree representation of the data contents.

**WrightTools.data.Data.prune**

Data.

`prune` *(keep_channels=True*, *, `verbose=True`)*

Remove unused variables and (optionally) channels from the Data object.

Unused variables are those that are not included in either axes or constants. Unused channels are those not specified in keep_channels, or the first channel.

**Parameters**

• `keep_channels` *(boolean or int or str or tuple)* – If False, removes all but the first channel. If int or str, removes all but that index/name channel. If tuple, removes all channels except those in the tuple by index or name. Default is True: do not delete channels

• `verbose` *(boolean)* – Toggle talkback. Default is True.

**WrightTools.data.Data.remove_channel**

Data.

`remove_channel` *(channel, *, `verbose=True`)*

Remove channel from data.

**Parameters**

• `channel` *(int or str)* – Channel index or name to remove.

• `verbose` *(boolean (optional))* – Toggle talkback. Default is True.
WrightTools.data.Data.remove_constant

```python
Data.remove_constant(constant, *, verbose=True)
```

Remove a constant from the stored list.

**Parameters**

- `constant` *(str or Constant or int)* – Expression for the new constant.
- `verbose` *(boolean (optional)) – Toggle talkback. Default is True

**See also:**

- `set_constants()`  Remove and replace all constants.
- `create_constant()`  Add an individual constant.

WrightTools.data.Data.remove_variable

```python
Data.remove_variable(variable, *, implied=True, verbose=True)
```

Remove variable from data.

**Parameters**

- `variable` *(int or str)* – Variable index or name to remove.
- `implied` *(boolean (optional)) – Toggle deletion of other variables that start with the same name. Default is True.
- `verbose` *(boolean (optional)) – Toggle talkback. Default is True.

WrightTools.data.Data.rename_channels

```python
Data.rename_channels(*, verbose=True, **kwargs)
```

Rename a set of channels.

**Parameters**

- `kwargs` – Keyword arguments of the form `current:'new'`.
- `verbose` *(boolean (optional)) – Toggle talkback. Default is True.

WrightTools.data.Data.rename_variables

```python
Data.rename_variables(*, implied=True, verbose=True, **kwargs)
```

Rename a set of variables.

**Parameters**

- `kwargs` – Keyword arguments of the form `current:'new'`.
- `implied` *(boolean (optional)) – Toggle inclusion of other variables that start with the same name. Default is True.
- `verbose` *(boolean (optional)) – Toggle talkback. Default is True.
WrightTools.data.Data.save

\texttt{Data.save(filepath=None, overwrite=False, verbose=True)}

Save as root of a new file.

**Parameters**

- \texttt{filepath} (Path-like object (optional)) – Filepath to write. If None, file is created using natural_name.
- \texttt{overwrite} (boolean (optional)) – Toggle overwrite behavior. Default is False.
- \texttt{verbose} (boolean (optional)) – Toggle talkback. Default is True

**Returns** Written filepath.

**Return type** str

WrightTools.data.Data.set_constants

\texttt{Data.set_constants(*constants, verbose=True)}

Set the constants associated with the data.

**Parameters**

- \texttt{constants} (str) – Expressions for the new set of constants.
- \texttt{verbose} (boolean (optional)) – Toggle talkback. Default is True

**See also:**

- \texttt{transform()}  Similar method except for axes.
- \texttt{create_constant()}  Add an individual constant.
- \texttt{remove_constant()}  Remove an individual constant.

WrightTools.data.Data.share_nans

\texttt{Data.share_nans()}

Share not-a-numbers between all channels.

If any channel is nan at a given index, all channels will be nan at that index after this operation.

Uses the share_nans method found in wt.kit.

WrightTools.data.Data.smooth

\texttt{Data.smooth(factors, channel=None, verbose=True) \rightarrow WrightTools.data._data.Data}

Smooth a channel using an n-dimensional kaiser window.

Note, all arrays are loaded into memory.

For more info see Kaiser_window wikipedia entry.

**Parameters**

- \texttt{factors} (int or list of int) – The smoothing factor. You may provide a list of smoothing factors for each axis.
• **channel** *(int or str or None (optional)) – The channel to smooth. If None, all channels will be smoothed. Default is None.*

• **verbose** *(bool (optional)) – Toggle talkback. Default is True.*

### WrightTools.data.Data.split

**Data.split**(expression, positions, *, units=None, parent=None, verbose=True) → WrightTools.collection._collection.Collection

Split the data object along a given expression, in units.

**Parameters**

- **expression** *(int or str)* – The expression to split along. If given as an integer, the axis at that index is used.
- **positions** *(number-type or 1D array-type)* – The position(s) to split at, in units.
- **units** *(str (optional)) – The units of the given positions. Default is same, which assumes input units are identical to first variable units.*
- **parent** *(WrightTools.Collection (optional)) – The parent collection in which to place the ‘split’ collection. Default is a new Collection.*
- **verbose** *(bool (optional)) – Toggle talkback. Default is True.*

**Returns** A Collection of data objects. The order of the objects is such that the axis points retain their original order.

**Return type** WrightTools.collection.Collection

**See also:**

chop() Divide the dataset into its lower-dimensionality components.
collapse() Collapse the dataset along one axis.

### WrightTools.data.Data.transform

**Data.transform**(axes, verbose=True)

Transform the data.

**Parameters**

- **axes** *(strings)* – Expressions for the new set of axes.
- **verbose** *(boolean (optional)) – Toggle talkback. Default is True*

**See also:**

set_constants() Similar method except for constants
### WrightTools.data.Data.update

Data.update([E], **F) → None. Update D from mapping/iterable E and F.

If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k, v in F.items(): D[k] = v

### WrightTools.data.Data.zoom

Data.zoom(factor, order=1, verbose=True)

Zoom the data array using spline interpolation of the requested order.

The number of points along each axis is increased by factor. See scipy ndimage for more info.

#### Parameters

- **factor** (float) – The number of points along each axis will increase by this factor.
- **order** (int (optional)) – The order of the spline used to interpolate onto new points.
- **verbose** (bool (optional)) – Toggle talkback. Default is True.

#### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrs</td>
<td>Attributes attached to this object</td>
</tr>
<tr>
<td>axes</td>
<td></td>
</tr>
<tr>
<td>axis_expressions</td>
<td>Axis expressions.</td>
</tr>
<tr>
<td>axis_names</td>
<td>Axis names.</td>
</tr>
<tr>
<td>channel_names</td>
<td>Channel names.</td>
</tr>
<tr>
<td>channels</td>
<td>Channels.</td>
</tr>
<tr>
<td>class_name</td>
<td></td>
</tr>
<tr>
<td>constant_expressions</td>
<td>Axis expressions.</td>
</tr>
<tr>
<td>constant_names</td>
<td>Axis names.</td>
</tr>
<tr>
<td>constant_units</td>
<td>All constant units.</td>
</tr>
<tr>
<td>constants</td>
<td></td>
</tr>
<tr>
<td>created</td>
<td></td>
</tr>
<tr>
<td>datasets</td>
<td>Datasets.</td>
</tr>
<tr>
<td>file</td>
<td>Return a File instance associated with this object</td>
</tr>
<tr>
<td>fullpath</td>
<td>file and internal structure.</td>
</tr>
<tr>
<td>id</td>
<td>Low-level identifier appropriate for this object</td>
</tr>
<tr>
<td>item_names</td>
<td>Item names.</td>
</tr>
<tr>
<td>kind</td>
<td>Kind.</td>
</tr>
<tr>
<td>name</td>
<td>Return the full name of this object.</td>
</tr>
<tr>
<td>natural_name</td>
<td>Natural name.</td>
</tr>
<tr>
<td>ndim</td>
<td>Get number of dimensions.</td>
</tr>
<tr>
<td>parent</td>
<td>Parent.</td>
</tr>
<tr>
<td>ref</td>
<td>An (opaque) HDF5 reference to this object</td>
</tr>
<tr>
<td>regionref</td>
<td>Create a region reference (Datasets only).</td>
</tr>
<tr>
<td>shape</td>
<td>Shape.</td>
</tr>
<tr>
<td>size</td>
<td>Size.</td>
</tr>
<tr>
<td>source</td>
<td>Source.</td>
</tr>
<tr>
<td>units</td>
<td>All axis units.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 9 – continued from previous page

<table>
<thead>
<tr>
<th>variable_names</th>
<th>Variable names.</th>
</tr>
</thead>
<tbody>
<tr>
<td>variables</td>
<td>Variables.</td>
</tr>
</tbody>
</table>

**WrightTools.data.Data.attrs**

```python
@property Data.attrs
Attributes attached to this object
```

**WrightTools.data.Data.axes**

```python
@property Data.axes
```

**WrightTools.data.Data.axis_expressions**

```python
@property Data.axis_expressions
Axis expressions.
```

**WrightTools.data.Data.axis_names**

```python
@property Data.axis_names
Axis names.
```

**WrightTools.data.Data.channel_names**

```python
@property Data.channel_names
Channel names.
```

**WrightTools.data.Data.channels**

```python
@property Data.channels
Channels.
```

**WrightTools.data.Data.class_name**

```python
Data.class_name = 'Data'
```
WrightTools.data.Data.constant_expressions

**property** Data.constant_expressions

Axis expressions.

WrightTools.data.Data.constant_names

**property** Data.constant_names

Axis names.

WrightTools.data.Data.constant_units

**property** Data.constant_units

All constant units.

WrightTools.data.Data.constants

**property** Data.constants

WrightTools.data.Data.created

**property** Data.created

WrightTools.data.Data.datasets

**property** Data.datasets

Datasets.

WrightTools.data.Data.file

**property** Data.file

Return a File instance associated with this object

WrightTools.data.Data.fullpath

**property** Data.fullpath

file and internal structure.

  Type Full path
**WrightTools.data.Data.id**

*property* Data.id

Low-level identifier appropriate for this object.

**WrightTools.data.Data.item_names**

*property* Data.item_names

Item names.

**WrightTools.data.Data.kind**

*property* Data.kind

Kind.

**WrightTools.data.Data.name**

*property* Data.name

Return the full name of this object. None if anonymous.

**WrightTools.data.Data.natural_name**

*property* Data.natural_name

Natural name.

**WrightTools.data.Data.ndim**

*property* Data.ndim

Get number of dimensions.

**WrightTools.data.Data.parent**

*property* Data.parent

Parent.

**WrightTools.data.Data.ref**

*property* Data.ref

An (opaque) HDF5 reference to this object.
WrightTools.data.Data.regionref

**property** Data.regionref
Create a region reference (Datasets only).

The syntax is regionref[{slices}]. For example, dset.regionref[...] creates a region reference in which the whole dataset is selected.

Can also be used to determine the shape of the referenced dataset (via .shape property), or the shape of the selection (via the .selection property).

WrightTools.data.Data.shape

**property** Data.shape
Shape.

WrightTools.data.Data.size

**property** Data.size
Size.

WrightTools.data.Data.source

**property** Data.source
Source.

WrightTools.data.Data.units

**property** Data.units
All axis units.

WrightTools.data.Data.variable_names

**property** Data.variable_names
Variable names.

WrightTools.data.Data.variables

**property** Data.variables
Variables.
WrightTools.data.Axis

class WrightTools.data.Axis (parent, expression, units=None)
    Bases: object

    Axis class.

    __init__ (parent, expression, units=None)
        Data axis.

        Parameters

        • parent (WrightTools.Data) – Parent data object.
        • expression (string) – Axis expression.
        • units (string (optional)) – Axis units. Default is None.

    Methods

    convert (destination_units, *[...])
        Convert axis to destination_units.

    max ()
        Axis max.

    min ()
        Axis min.

WrightTools.data.Axis.convert

Axis.convert (destination_units, *, convert_variables=False)
    Convert axis to destination_units.

    Parameters

    • destination_units (string) – Destination units.
    • convert_variables (boolean (optional)) – Toggle conversion of stored arrays. Default is False.

WrightTools.data.Axis.max

Axis.max ()
    Axis max.

WrightTools.data.Axis.min

Axis.min ()
    Axis min.
Attributes

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>full</strong></td>
<td>Axis expression evaluated and repeated to match the shape of the parent data object.</td>
</tr>
<tr>
<td><strong>identity</strong></td>
<td>Complete identifier written to disk in data.attrs['axes'].</td>
</tr>
<tr>
<td><strong>label</strong></td>
<td>A latex formatted label representing axis expression.</td>
</tr>
<tr>
<td><strong>masked</strong></td>
<td>Axis expression evaluated, and masked with NaN shared from data channels.</td>
</tr>
<tr>
<td><strong>natural_name</strong></td>
<td>Valid python identifier representation of the expression.</td>
</tr>
<tr>
<td><strong>ndim</strong></td>
<td>Get number of dimensions.</td>
</tr>
<tr>
<td><strong>points</strong></td>
<td>Squeezed array.</td>
</tr>
<tr>
<td><strong>shape</strong></td>
<td>Shape.</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>Size.</td>
</tr>
<tr>
<td><strong>units_kind</strong></td>
<td>Units kind.</td>
</tr>
<tr>
<td><strong>variables</strong></td>
<td>Variables.</td>
</tr>
</tbody>
</table>

**WrightTools.data.Axis.full**

**property** `Axis.full`

Axis expression evaluated and repeated to match the shape of the parent data object.

**WrightTools.data.Axis.identity**

**property** `Axis.identity`

Complete identifier written to disk in data.attrs['axes'].

**WrightTools.data.Axis.label**

**property** `Axis.label`

A latex formatted label representing axis expression.

**WrightTools.data.Axis.masked**

**property** `Axis.masked`

Axis expression evaluated, and masked with NaN shared from data channels.

**WrightTools.data.Axis.natural_name**

**property** `Axis.natural_name`

Valid python identifier representation of the expression.
WrightTools.data.Axis.ndim

**property** `Axis.ndim`
Get number of dimensions.

WrightTools.data.Axis.points

**property** `Axis.points`
Squeezed array.

WrightTools.data.Axis.shape

**property** `Axis.shape`
Shape.

WrightTools.data.Axis.size

**property** `Axis.size`
Size.

WrightTools.data.Axis.units_kind

**property** `Axis.units_kind`
Units kind.

WrightTools.data.Axis.variables

**property** `Axis.variables`
Variables.

WrightTools.data.Channel

class WrightTools.data.Channel(parent, id, *, units=None, null=None, signed=None, label=None, label_seed=None, **kwargs)
Bases: WrightTools._dataset.Dataset
Channel.

__init__(parent, id, *, units=None, null=None, signed=None, label=None, label_seed=None, **kwargs)
Construct a channel object.

**Parameters**

- **values** (`array-like`) – Values.
- **name** (`string`) – Channel name.
- **units** (`string (optional)`) – Channel units. Default is None.
- **null** (`number (optional)`) – Channel null. Default is None (0).
- **signed** *(bool, optional)* – Channel signed flag. Default is None (guess).
- **label** *(string)* – Label. Default is None.
- **label_seed** *(list of strings)* – Label seed. Default is None.
- ****kwargs** – Additional keyword arguments are added to the attrs dictionary and to the natural namespace of the object (if possible).

### Methods

```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argmax()</td>
<td>Index of the maximum, ignoring nans.</td>
</tr>
<tr>
<td>argmin()</td>
<td>Index of the minimum, ignoring nans.</td>
</tr>
<tr>
<td>chunkwise(func, *args, **kwargs)</td>
<td>Execute a function for each chunk in the dataset.</td>
</tr>
<tr>
<td>clip([min, max, replace])</td>
<td>Clip values outside of a defined range.</td>
</tr>
<tr>
<td>convert(destination_units)</td>
<td>Convert units.</td>
</tr>
<tr>
<td>log([base, floor])</td>
<td>Take the log of the entire dataset.</td>
</tr>
<tr>
<td>log10([floor])</td>
<td>Take the log base 10 of the entire dataset.</td>
</tr>
<tr>
<td>log2([floor])</td>
<td>Take the log base 2 of the entire dataset.</td>
</tr>
<tr>
<td>mag()</td>
<td>Channel magnitude (maximum deviation from null).</td>
</tr>
<tr>
<td>max()</td>
<td>Maximum, ignoring nans.</td>
</tr>
<tr>
<td>min()</td>
<td>Minimum, ignoring nans.</td>
</tr>
<tr>
<td>normalize([mag])</td>
<td>Normalize a Channel, set null to 0 and the mag to given value.</td>
</tr>
<tr>
<td>slices()</td>
<td>Returns a generator yielding tuple of slice objects.</td>
</tr>
<tr>
<td>symmetric_root([root])</td>
<td></td>
</tr>
<tr>
<td>trim(neighborhood[, method, factor, ...])</td>
<td>Remove outliers from the dataset.</td>
</tr>
</tbody>
</table>
```

#### WrightTools.data.Channel.argmax

Channel.argmax()  
Index of the maximum, ignoring nans.

#### WrightTools.data.Channel.argmin

Channel.argmin()  
Index of the minimum, ignoring nans.

#### WrightTools.data.Channel.chunkwise

Channel.chunkwise(func, *args, **kwargs)  
Execute a function for each chunk in the dataset.  
Order of execution is not guaranteed.  

**Parameters**

- **func** *(function)* – Function to execute. First two arguments must be dataset, slices.
- **(optional) (kwargs)** – Additional (unchanging) arguments passed to func.
- **(optional)** – Additional (unchanging) keyword arguments passed to func.
Returns  Dictionary of index: function output. Index is to lowest corner of each chunk.

Return type  collections OrderedDict

**WrightTools.data.Channel.clip**

Channel.clip(min=None, max=None, replace=nan)
Clip values outside of a defined range.

Parameters
- **min** (number (optional)) – New channel minimum. Default is None.
- **max** (number (optional)) – New channel maximum. Default is None.
- **replace** (number or 'value' (optional)) – Replace behavior. Default is nan.

**WrightTools.data.Channel.convert**

Channel.convert(destination_units)
Convert units.

Parameters **destination_units** (string (optional)) – Units to convert into.

**WrightTools.data.Channel.log**

Channel.log(base=2.718281828459045, floor=None)
Take the log of the entire dataset.

Parameters
- **base** (number (optional)) – Base of log. Default is e.
- **floor** (number (optional)) – Clip values below floor after log. Default is None.

**WrightTools.data.Channel.log10**

Channel.log10(floor=None)
Take the log base 10 of the entire dataset.

Parameters **floor** (number (optional)) – Clip values below floor after log. Default is None.

**WrightTools.data.Channel.log2**

Channel.log2(floor=None)
Take the log base 2 of the entire dataset.

Parameters **floor** (number (optional)) – Clip values below floor after log. Default is None.
WrightTools.data.Channel.mag

Channel.mag() → complex
Channel magnitude (maximum deviation from null).

WrightTools.data.Channel.max

Channel.max()
Maximum, ignoring nans.

WrightTools.data.Channel.min

Channel.min()
Minimum, ignoring nans.

WrightTools.data.Channel.normalize

Channel.normalize(mag=1.0)
Normalize a Channel, set null to 0 and the mag to given value.

Parameters

- **mag** (float (optional)) – New value of mag. Default is 1.

WrightTools.data.Channel.slices

Channel.slices()
Returns a generator yielding tuple of slice objects.
Order is not guaranteed.

WrightTools.data.Channel.symmetric_root

Channel.symmetric_root(root=2)

WrightTools.data.Channel.trim

Channel.trim(neighborhood, method='ztest', factor=3, replace='nan', verbose=True)
Remove outliers from the dataset.
Identifies outliers by comparing each point to its neighbors using a statistical test.

Parameters

- **neighborhood** (list of integers) – Size of the neighborhood in each dimension. Length of the list must be equal to the dimensionality of the channel.
- **method** ({'ztest'} (optional)) – Statistical test used to detect outliers. Default is ztest.
  - **ztest** Compare point deviation from neighborhood mean to neighborhood standard deviation.
- **factor** (number (optional)) – Tolerance factor. Default is 3.
• **replace** ({'**nan**', 'mean', 'exclusive_mean', number} (optional)) – Behavior of outlier replacement. Default is **nan**.

  **nan** Outliers are replaced by numpy nans.

  **mean** Outliers are replaced by the mean of its neighborhood, including itself.

  **exclusive_mean** Outliers are replaced by the mean of its neighborhood, not including itself.

  **number** Array becomes given number.

**Returns** Indicies of trimmed outliers.

**Return type** list of tuples

**See also:**

*clip()* Remove pixels outside of a certain range.

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>attrs</strong></td>
<td>Attributes attached to this object</td>
</tr>
<tr>
<td><strong>class_name</strong></td>
<td>Numpy dtype representing the datatype</td>
</tr>
<tr>
<td><strong>file</strong></td>
<td>Return a File instance associated with this object</td>
</tr>
<tr>
<td><strong>fillvalue</strong></td>
<td>Fill value for this dataset (0 by default)</td>
</tr>
<tr>
<td><strong>flush</strong></td>
<td>Flush the dataset data and metadata to the file.</td>
</tr>
<tr>
<td><strong>full</strong></td>
<td>File and internal structure.</td>
</tr>
<tr>
<td><strong>major_extent</strong></td>
<td>Maximum deviation from null.</td>
</tr>
<tr>
<td><strong>minor_extent</strong></td>
<td>Minimum deviation from null.</td>
</tr>
<tr>
<td><strong>name</strong></td>
<td>Return the full name of this object.</td>
</tr>
<tr>
<td><strong>natural_name</strong></td>
<td>Natural name of the dataset.</td>
</tr>
<tr>
<td><strong>ndim</strong></td>
<td>Numpy-style attribute giving the number of dimensions</td>
</tr>
<tr>
<td><strong>null</strong></td>
<td>Parent.</td>
</tr>
<tr>
<td><strong>parent</strong></td>
<td>Squeezed array.</td>
</tr>
<tr>
<td><strong>shape</strong></td>
<td>Numpy-style shape tuple giving dataset dimensions</td>
</tr>
<tr>
<td><strong>signed</strong></td>
<td>Numpy-style attribute giving the total dataset size</td>
</tr>
<tr>
<td><strong>size</strong></td>
<td>Units.</td>
</tr>
<tr>
<td><strong>value</strong></td>
<td>Alias for dataset[()]</td>
</tr>
</tbody>
</table>
WrightTools.data.Channel.attrs

**property** Channel.attrs
Attributes attached to this object

WrightTools.data.Channel.class_name

Channel.class_name = 'Channel'

WrightTools.data.Channel.dtype

**property** Channel.dtype
Numpy dtype representing the datatype

WrightTools.data.Channel.file

**property** Channel.file
Return a File instance associated with this object

WrightTools.data.Channel.fillvalue

**property** Channel.fillvalue
Fill value for this dataset (0 by default)

WrightTools.data.Channel.flush

Channel.flush
Flush the dataset data and metadata to the file. If the dataset is chunked, raw data chunks are written to the file.

This is part of the SWMR features and only exist when the HDF5 library version >=1.9.178

WrightTools.data.Channel.full

**property** Channel.full

WrightTools.data.Channel.fullpath

**property** Channel.fullpath
file and internal structure.

   Type  Full path
**WrightTools.data.Channel.major_extent**

**property** Channel.major_extent
Maximum deviation from null.

**WrightTools.data.Channel.minor_extent**

**property** Channel.minor_extent
Minimum deviation from null.

**WrightTools.data.Channel.name**

**property** Channel.name
Return the full name of this object. None if anonymous.

**WrightTools.data.Channel.natural_name**

**property** Channel.natural_name
Natural name of the dataset. May be different from name.

**WrightTools.data.Channel.ndim**

**property** Channel.ndim
Numpy-style attribute giving the number of dimensions.

**WrightTools.data.Channel.null**

**property** Channel.null

**WrightTools.data.Channel.parent**

**property** Channel.parent
Parent.

**WrightTools.data.Channel.points**

**property** Channel.points
Squeezed array.
WrightTools.data.Channel.shape

property Channel.shape
Numpy-style shape tuple giving dataset dimensions

WrightTools.data.Channel.signed

property Channel.signed

WrightTools.data.Channel.size

property Channel.size
Numpy-style attribute giving the total dataset size

WrightTools.data.Channel.units

property Channel.units
Units.

WrightTools.data.Channel.value

property Channel.value
Alias for dataset[()]

WrightTools.data.Constant

class WrightTools.data.Constant (parent, expression, units=None, format_spec='0.3g', round_spec=None)
Bases: WrightTools.data._axis.Axis
Constant class.

__init__ (parent, expression, units=None, format_spec='0.3g', round_spec=None)
Data constant.

Parameters

• parent (WrightTools.Data) – Parent data object.
• expression (string) – Constant expression.
• units (string (optional)) – Constant units. Default is None.
• format_spec (string (optional)) – Format string specification, as passed to format(). Default is “0.3g”
• round_spec (int or None (optional)) – Decimal digits to round to before formatting, as passed to round(). Default is None (no rounding).
## Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>convert(destination_units, *[...])</code></td>
<td>Convert axis to destination_units.</td>
</tr>
<tr>
<td><code>max()</code></td>
<td>Axis max.</td>
</tr>
<tr>
<td><code>min()</code></td>
<td>Axis min.</td>
</tr>
</tbody>
</table>

### WrightTools.data.Constant.convert

Constant.[`convert(destination_units, *, convert_variables=False)`](#)  
Convert axis to destination_units.

**Parameters**

- `destination_units (string)` – Destination units.
- `convert_variables (boolean (optional))` – Toggle conversion of stored arrays. Default is False.

### WrightTools.data.Constant.max

Constant.[`max()`](#)  
Axis max.

### WrightTools.data.Constant.min

Constant.[`min()`](#)  
Axis min.

## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>full</code></td>
<td>Axis expression evaluated and repeated to match the shape of the parent data object.</td>
</tr>
<tr>
<td><code>identity</code></td>
<td>Complete identifier written to disk in data.attrs['axes'].</td>
</tr>
<tr>
<td><code>label</code></td>
<td>A latex formatted label representing constant expression and united value.</td>
</tr>
<tr>
<td><code>masked</code></td>
<td>Axis expression evaluated, and masked with NaN shared from data channels.</td>
</tr>
<tr>
<td><code>natural_name</code></td>
<td>Valid python identifier representation of the expression.</td>
</tr>
<tr>
<td><code>ndim</code></td>
<td>Get number of dimensions.</td>
</tr>
<tr>
<td><code>points</code></td>
<td>Squeezed array.</td>
</tr>
<tr>
<td><code>shape</code></td>
<td>Shape.</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Size.</td>
</tr>
<tr>
<td><code>std</code></td>
<td>The standard deviation of the constant.</td>
</tr>
<tr>
<td><code>units_kind</code></td>
<td>Units kind.</td>
</tr>
<tr>
<td><code>value</code></td>
<td>The value of the constant.</td>
</tr>
<tr>
<td><code>variables</code></td>
<td>Variables.</td>
</tr>
</tbody>
</table>
**WrightTools.data.Constant.full**

*property* `Constant.full`  
Axis expression evaluated and repeated to match the shape of the parent data object.

**WrightTools.data.Constant.identity**

*property* `Constant.identity`  
Complete identifier written to disk in `data.attrs['axes']`.

**WrightTools.data.Constant.label**

*property* `Constant.label`  
A latex formatted label representing constant expression and united value.

**WrightTools.data.Constant.masked**

*property* `Constant.masked`  
Axis expression evaluated, and masked with NaN shared from data channels.

**WrightTools.data.Constant.natural_name**

*property* `Constant.natural_name`  
Valid python identifier representation of the expression.

**WrightTools.data.Constant.ndim**

*property* `Constant.ndim`  
Get number of dimensions.

**WrightTools.data.Constant.points**

*property* `Constant.points`  
Squeezed array.

**WrightTools.data.Constant.shape**

*property* `Constant.shape`  
Shape.
WrightTools.data.Constant.size

property Constant.size
Size.

WrightTools.data.Constant.std

property Constant.std
The standard deviation of the constant.

WrightTools.data.Constant.units_kind

property Constant.units_kind
Units kind.

WrightTools.data.Constant.value

property Constant.value
The value of the constant.

WrightTools.data.Constant.variables

property Constant.variables
Variables.

WrightTools.data.Variable

class WrightTools.data.Variable(parent, id, units=None, **kwargs)
   Bases: WrightTools._dataset.Dataset
   Variable.
   __init__(parent, id, units=None, **kwargs)
   Variable.

   Parameters
   • parent (WrightTools.Data) – Parent data object.
   • id (h5py DatasetID) – Dataset ID.
   • units (string (optional)) – Variable units. Default is None.
   • kwargs – Additional keys and values to be written into dataset attrs.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argmax()</td>
<td>Index of the maximum, ignoring nans.</td>
</tr>
<tr>
<td>argmin()</td>
<td>Index of the minimum, ignoring nans.</td>
</tr>
<tr>
<td>chunkwise(func, *args, **kwargs)</td>
<td>Execute a function for each chunk in the dataset. Order of execution is not guaranteed.</td>
</tr>
<tr>
<td>clip([min, max, replace])</td>
<td>Clip values outside of a defined range.</td>
</tr>
<tr>
<td>convert(destination_units)</td>
<td>Convert units.</td>
</tr>
<tr>
<td>log([base, floor])</td>
<td>Take the log of the entire dataset.</td>
</tr>
<tr>
<td>log10([floor])</td>
<td>Take the log base 10 of the entire dataset.</td>
</tr>
<tr>
<td>log2([floor])</td>
<td>Take the log base 2 of the entire dataset.</td>
</tr>
<tr>
<td>max()</td>
<td>Maximum, ignoring nans.</td>
</tr>
<tr>
<td>min()</td>
<td>Minimum, ignoring nans.</td>
</tr>
<tr>
<td>slices()</td>
<td>Returns a generator yielding tuple of slice objects.</td>
</tr>
<tr>
<td>symmetric_root([root])</td>
<td></td>
</tr>
<tr>
<td>write_direct(source[, source_sel, dest_sel])</td>
<td>Write data directly to HDF5 from a NumPy array.</td>
</tr>
</tbody>
</table>

WrightTools.data.Variable.argmax

Variable.argmax()
Index of the maximum, ignoring nans.

WrightTools.data.Variable.argmin

Variable.argmin()
Index of the minimum, ignoring nans.

WrightTools.data.Variable.chunkwise

Variable.chunkwise(func, *args, **kwargs)
Execute a function for each chunk in the dataset.

Order of execution is not guaranteed.

Parameters

- **func (function)** – Function to execute. First two arguments must be dataset, slices.
- **(optional) (kwargs)** – Additional (unchanging) arguments passed to func.
- **(optional)** – Additional (unchanging) keyword arguments passed to func.

Returns Dictionary of index: function output. Index is to lowest corner of each chunk.

Return type collections.OrderedDict
**WrightTools.data.Variable.clip**

Variable.clip(min=None, max=None, replace=nan)

Clip values outside of a defined range.

**Parameters**

- **min (number (optional))** – New channel minimum. Default is None.
- **max (number (optional))** – New channel maximum. Default is None.
- **replace (number or 'value' (optional))** – Replace behavior. Default is nan.

**WrightTools.data.Variable.convert**

Variable.convert(destination_units)

Convert units.

**Parameters**

- **destination_units (string (optional))** – Units to convert into.

**WrightTools.data.Variable.log**

Variable.log(base=2.718281828459045, floor=None)

Take the log of the entire dataset.

**Parameters**

- **base (number (optional))** – Base of log. Default is e.
- **floor (number (optional))** – Clip values below floor after log. Default is None.

**WrightTools.data.Variable.log10**

Variable.log10(floor=None)

Take the log base 10 of the entire dataset.

**Parameters**

- **floor (number (optional))** – Clip values below floor after log. Default is None.

**WrightTools.data.Variable.log2**

Variable.log2(floor=None)

Take the log base 2 of the entire dataset.

**Parameters**

- **floor (number (optional))** – Clip values below floor after log. Default is None.
WrightTools.data.Variable.max

Variable\_max()  
Maximum, ignoring nans.

WrightTools.data.Variable.min

Variable\_min()  
Minimum, ignoring nans.

WrightTools.data.Variable.slices

Variable\_slices()  
Returns a generator yielding tuple of slice objects.  
Order is not guaranteed.

WrightTools.data.Variable.symmetric_root

Variable\_symmetric\_root(root=2)

WrightTools.data.Variable.write_direct

Variable\_write\_direct(source, source\_sel=None, dest\_sel=None)  
Write data directly to HDF5 from a NumPy array.  
The source array must be C-contiguous. Selections must be the output of numpy.s\_[<args>].  
Broadcasting is supported for simple indexing.

Attributes

<table>
<thead>
<tr>
<th>attr</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attrs</td>
<td>Attributes attached to this object</td>
</tr>
<tr>
<td>class_name</td>
<td>Numpy dtype representing the datatype</td>
</tr>
<tr>
<td>dtype</td>
<td>Return a File instance associated with this object</td>
</tr>
<tr>
<td>file</td>
<td>Fill value for this dataset (0 by default)</td>
</tr>
<tr>
<td>flush</td>
<td>Flush the dataset data and metadata to the file.</td>
</tr>
<tr>
<td>full</td>
<td>file and internal structure.</td>
</tr>
<tr>
<td>fullpath</td>
<td>Return the full name of this object.</td>
</tr>
<tr>
<td>label</td>
<td>Natural name of the dataset.</td>
</tr>
<tr>
<td>name</td>
<td>Numpy-style attribute giving the number of dimensions</td>
</tr>
<tr>
<td>natural_name</td>
<td>Parent.</td>
</tr>
<tr>
<td>points</td>
<td>Squeezed array.</td>
</tr>
<tr>
<td>shape</td>
<td>Numpy-style shape tuple giving dataset dimensions</td>
</tr>
</tbody>
</table>

Continued on next page


Table 17 – continued from previous page

<table>
<thead>
<tr>
<th>size</th>
<th>Numpy-style attribute giving the total dataset size</th>
</tr>
</thead>
<tbody>
<tr>
<td>units</td>
<td>Units.</td>
</tr>
</tbody>
</table>

**WrightTools.data.Variable.attrs**

**property** `Variable.attrs`

Attributes attached to this object

**WrightTools.data.Variable.class_name**

`Variable.class_name = 'Variable'`

**WrightTools.data.Variable.dtype**

**property** `Variable.dtype`

Numpy dtype representing the datatype

**WrightTools.data.Variable.file**

**property** `Variable.file`

Return a File instance associated with this object

**WrightTools.data.Variable.fillvalue**

**property** `Variable.fillvalue`

Fill value for this dataset (0 by default)

**WrightTools.data.Variable.flush**

`Variable.flush`

Flush the dataset data and metadata to the file. If the dataset is chunked, raw data chunks are written to the file.

This is part of the SWMR features and only exist when the HDF5 library version >=1.9.178

**WrightTools.data.Variable.full**

**property** `Variable.full`


WrightTools.data.Variable.fullpath

property Variable.fullpath
file and internal structure.
    Type Full path

WrightTools.data.Variable.label

property Variable.label

WrightTools.data.Variable.name

property Variable.name
    Return the full name of this object. None if anonymous.

WrightTools.data.Variable.natural_name

property Variable.natural_name
    Natural name of the dataset. May be different from name.

WrightTools.data.Variable.ndim

property Variable.ndim
    Numpy-style attribute giving the number of dimensions

WrightTools.data.Variable.parent

property Variable.parent
    Parent.

WrightTools.data.Variable.points

property Variable.points
    Squeezed array.

WrightTools.data.Variable.shape

property Variable.shape
    Numpy-style shape tuple giving dataset dimensions
**WrightTools.data.Variable.size**

**property** Variable.size

Numpy-style attribute giving the total dataset size

**WrightTools.data.Variable.units**

**property** Variable.units

Units.

**WrightTools.data.join**

WrightTools.data.join(datas, *, atol=None, rtol=None, name='join', parent=None, method='first', verbose=True) → WrightTools.data._data.Data

Join a list of data objects together.

Joined datas must have the same transformation applied. This transformation will define the array order for the joined dataset. All axes in the applied transformation must be a single variable, the result will have sorted numbers.

Join does not perform any interpolation. For that, look to Data.map_variable or Data.heal

**Parameters**

- **datas** *(list of data or WrightTools.Collection)* – The list or collection of data objects to join together.
- **atol** *(numeric or list of numeric)* – The absolute tolerance to use (in np.isclose) to consider points overlapped. If given as a single number, applies to all axes. If given as a list, must have same length as the data transformation. None in the list invokes default behavior. Default is 10% of the minimum spacing between consecutive points in any input data file.
- **rtol** *(numeric or list of numeric)* – The relative tolerance to use (in np.isclose) to consider points overlapped. If given as a single number, applies to all axes. If given as a list, must have same length as the data transformation. None in the list invokes default behavior. Default is 4 * np.finfo(dtype).resolution for floating point types, 0 for integer types.
- **name** *(str (optional))* – The name for the data object which is created. Default is ‘join’.
- **parent** *(WrightTools.Collection (optional))* – The location to place the joined data object. Default is new temp file at root.
- **method** *({'first', 'last', 'min', 'max', 'sum', 'mean'})* – Mode to use for merged points in the joined space. Default is ‘first’.
- **verbose** *(bool (optional))* – Toggle talkback. Default is True.

**Returns** A new Data instance.

**Return type** WrightTools.data.Data
WrightTools.data.from_BrunoldRaman

WrightTools.data.from_BrunoldRaman(filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data

Create a data object from the Brunold rRaman instrument.
Expects one energy (in wavenumbers) and one counts value.

Parameters

- filepath (path-like) – Path to .txt file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- name (string (optional)) – Name to give to the created data object. If None, filename is used. Default is None.
- parent (WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.
- verbose (boolean (optional)) – Toggle talkback. Default is True.

Returns New data object(s).
Return type data

WrightTools.data.from_COLORS

WrightTools.data.from_COLORS(filepaths, name=None, cols=None, invert_d1=True, ignore=['w3', 'wa', 'dref', 'm0', 'm1', 'm2', 'm3', 'm4', 'm5', 'm6'], parent=None, verbose=True)

Create data object from COLORS file(s).

Parameters

- filepaths (path-like or list of path-like) – Filepath(s). Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- name (string (optional)) – Unique dataset identifier. If None (default), autogenerated.
- cols({'v0', 'v1', 'v2'} (optional)) – Format of COLORS dat file. If None, autorecognized. Default is None.
- invert_d1 (boolean (optional)) – Toggle inversion of D1 at import time. Default is True.
- ignore (list of strings (optional)) – Columns to ignore.
- parent (WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.
- verbose (bool (optional)) – Toggle talkback. Default is True.

Returns Data from COLORS.
Return type WrightTools.Data
WrightTools.data.from_JASCO

WrightTools.data.from_JASCO(filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data
Create a data object from JASCO UV-Vis spectrometers.

Parameters

• filepath(path-like) – Path to .txt file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.

• name(string (optional)) – Name to give to the created data object. If None, file-name is used. Default is None.

• parent(WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.

• verbose(boolean (optional)) – Toggle talkback. Default is True.

Returns New data object(s).

Return type data

WrightTools.data.from_KENT

WrightTools.data.from_KENT(filepaths, name=None, ignore=['wm'], delay_tolerance=0.1, frequency_tolerance=0.5, parent=None, verbose=True) → WrightTools.data._data.Data
Create data object from KENT file(s).

Parameters

• filepaths(path-like or list of path-like) – Filepath(s). Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.

• name(string (optional)) – Unique dataset identifier. If None (default), autogenerated.

• ignore(list of strings (optional)) – Columns to ignore. Default is ['wm'].

• delay_tolerance(float (optional)) – Tolerance below-which to ignore delay changes (in picoseconds). Default is 0.1.

• frequency_tolerance(float (optional)) – Tolerance below-which to ignore frequency changes (in wavenumbers). Default is 0.5.

• parent(WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.

• verbose(bool (optional)) – Toggle talkback. Default is True.

Returns Data from KENT.

Return type WrightTools.Data
WrightTools.data.from_Aramis

WrightTools.data.from_Aramis(filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data
Create a data object from Horiba Aramis ngc binary file.

Parameters

- **filepath** (path-like) – Path to .ngc file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- **name** (string (optional)) – Name to give to the created data object. If None, name is extracted from file. Default is None.
- **parent** (WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.
- **verbose** (boolean (optional)) – Toggle talkback. Default is True.

Returns New data object(s).

Return type data

WrightTools.data.from_ocean_optics

WrightTools.data.from_ocean_optics(filepath, name=None, *, parent=None, verbose=True) → WrightTools.data._data.Data
Create a data object from an Ocean Optics brand spectrometer.

Parameters

- **filepath** (path-like) – Path to an ocean optics output file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- **name** (str or None (optional)) – The name to be applied to the new data object. If None, name is read from file.
- **parent** (WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.
- **verbose** (boolean (optional)) – Toggle talkback. Default is True.

Returns New data object.

Return type data

WrightTools.data.from_PyCMDS

WrightTools.data.from_PyCMDS(filepath, name=None, parent=None, verbose=True, *, collapse=True) → WrightTools.data._data.Data
Create a data object from a single PyCMDS output file.

Parameters

- **filepath** (path-like) – Path to the .data file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- **name** (str or None (optional)) – The name to be applied to the new data object. If None, name is read from file.
- **parent** (WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.
• **verbose** *(bool (optional)) – Toggle talkback. Default is True.*

**Returns** A Data instance.

**Return type** data

### WrightTools.data.from_shimadzu

WrightTools.data.from_shimadzu *(filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data*

Create a data object from Shimadzu .txt file.

**Parameters**

• **filepath** *(path-like)* – Path to .txt file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.

• **name** *(string (optional)) – Name to give to the created data object. If None, filename is used. Default is None.*

• **parent** *(WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.*

• **verbose** *(boolean (optional)) – Toggle talkback. Default is True.*

**Returns** New data object.

**Return type** data

### WrightTools.data.from_Solis

WrightTools.data.from_Solis *(filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data*

Create a data object from Andor Solis software (ascii exports).

**Parameters**

• **filepath** *(path-like)* – Path to .txt file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.

• **name** *(string (optional)) – Name to give to the created data object. If None, filename is used. Default is None.*

• **parent** *(WrightTools.Collection (optional)) – Collection to place new data object within. Default is None.*

• **verbose** *(boolean (optional)) – Toggle talkback. Default is True.*

**Returns** New data object.

**Return type** data
WrightTools.data.from_spcm

WrightTools.data.from_spcm (filepath, name=None, *, delimiter='\', parent=None, verbose=True) → WrightTools.data._data.Data

Create a Data object from a Becker & Hickl spcm file (ASCII-exported, .asc).

If provided, setup parameters are stored in the attrs dictionary of the Data object.

See the spcm software hompage for more info.

Parameters

- **filepath (path-like)** – Path to SPC-xxx .asc file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- **name (string (optional))** – Name to give to the created data object. If None, filename is used. Default is None.
- **delimiter (string (optional))** – The string used to separate values. Default is '\'.
- **parent (WrightTools.Collection (optional))** – Collection to place new data object within. Default is None.
- **verbose (boolean (optional))** – Toggle talkback. Default is True.

Returns

Return type WrightTools.data.Data object

WrightTools.data.from_Tensor27

WrightTools.data.from_Tensor27 (filepath, name=None, parent=None, verbose=True) → WrightTools.data._data.Data

Create a data object from a Tensor27 FTIR file.

```python
>>> import WrightTools as wt
>>> import matplotlib.pyplot as plt
>>> from WrightTools import datasets
>>> p = datasets.Tensor27.CuPcTs_powder_ATR
>>> data = wt.data.from_Tensor27(p)
>>> artist = wt.artists.quick1D(data)
>>> plt.xlim(1300,1700)
>>> plt.ylim(-0.005,.02)
```

Parameters

- **filepath (path-like)** – Path to Tensor27 output file (.dpt). Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.
- **name (string (optional))** – Name to give to the created data object. If None, filename is used. Default is None.
- **parent (WrightTools.Collection (optional))** – Collection to place new data object within. Default is None.
- **verbose (boolean (optional))** – Toggle talkback. Default is True.

Returns

New data object.

Return type data
1.10.4 WrightTools.diagrams package

Diagrams.

WrightTools.diagrams.WMEL module

WMEL diagrams.

class WrightTools.diagrams.WMEL.Artist(size, energies, state_names=None, number_of_interactions=4, virtual=[None], state_font_size=8, state_text_buffer=0.5)

Bases: object

Dedicated WMEL figure artist.

__init__(size, energies, state_names=None, number_of_interactions=4, virtual=[None], state_font_size=8, state_text_buffer=0.5)

Initialize.

Parameters

- size([rows, columns]) – Layout.
- energies(list of numbers) – State energies.
- state_names(list of strings (optional)) – State names. Default is None.
- number_of_interactions(integer (optional)) – Number of interactions. Default is 4.
• **virtual** *(list of integers (optional)) – Indices of states which are virtual. Default is [None].*

• **state_font_size** *(number (optional)) – State font size. Default is 8.*

• **state_text_buffer** *(number (optional)) – Size of buffer around state text. Default is 0.5.*

**add_arrow** *(diagram, number, between, kind, label='', head_length=0.075, font_size=7, color='k')*

Add arrow.

**Parameters**

• **diagram** *(column, row) – Diagram position.*

• **number** *(integer) – Arrow position.*

• **between** *(start, stop) – Arrow span.*

• **kind** *(‘ket’, ‘bra’, ‘out’) – Arrow style.*

• **label** *(string (optional)) – Arrow label. Default is ‘’.*

• **head_length** *(number (optional)) – Arrow head length. Default 0.075.*

• **font_size** *(number (optional)) – Font size. Default is 7.*

• **color** *(matplotlib color) – Arrow color. Default is ‘k’.*

**Returns** *[line, arrow_head, text]*

**Return type** *list*

**clear_diagram** *(diagram)*

Clear diagram.

**Parameters**

• **diagram** *(column, row) – Diagram to clear.*

**label_columns** *(labels, font_size=15, text_buffer=1.15)*

Label columns.

**Parameters**

• **labels** *(list of strings) – Labels.*

• **font_size** *(number (optional)) – Font size. Default is 15.*

• **text_buffer** *(number) – Buffer around text. Default is 1.5.*

**label_rows** *(labels, font_size=15, text_buffer=1.5)*

Label rows.

**Parameters**

• **labels** *(list of strings) – Labels.*

• **font_size** *(number (optional)) – Font size. Default is 15.*

• **text_buffer** *(number) – Buffer around text. Default is 1.5.*

**plot** *(save_path=None, close=False, bbox_inches='tight', pad_inches=1)*

Plot figure.

**Parameters**

• **save_path** *(string (optional)) – Save path. Default is None.*

• **close** *(boolean (optional)) – Toggle automatic figure closure after plotting. Default is False.*
• **bbox_inches** *(number (optional)) – Bounding box size, in inches. Default is 'tight'.*

• **pad_inches** *(number (optional)) – Pad inches. Default is 1.*

class WrightTools.diagrams.WMEL.Subplot *(ax, energies, number_of_interactions=4, title='', title_font_size=16, state_names=None, virtual=[None], state_font_size=14, state_text_buffer=0.5, label_side='left')*

Bases: object

Subplot containing WMEL.

__init__(ax, energies, number_of_interactions=4, title='', title_font_size=16, state_names=None, virtual=[None], state_font_size=14, state_text_buffer=0.5, label_side='left')

Subplot.

Parameters

• **ax** *(matplotlib axis) – The axis.*

• **energies** *(1D array-like) – Energies (scaled between 0 and 1)*

• **number_of_interactions** *(integer) – Number of interactions in diagram.*

• **title** *(string (optional)) – Title of subplot. Default is empty string.*

• **state_names** *(list of str (optional)) – list of the names of the states*

• **virtual** *(list of ints (optional)) – list of indexes of any virtual energy states*

• **state_font_size** *(numtype (optional)) – font size for the state labels*

• **state_text_buffer** *(numtype (optional)) – space between the energy level bars and the state labels*

add_arrow *(index, between, kind, label='', head_length=10, head_aspect=1, font_size=14, color='k')*

Add an arrow to the WMEL diagram.

Parameters

• **index** *(integer) – The interaction, or start and stop interaction for the arrow.*

• **between** *(2-element iterable of integers) – The initial and final state of the arrow*

• **kind** *(['ket', 'bra']) – The kind of interaction.*

• **label** *(string (optional)) – Interaction label. Default is empty string.*

• **head_length** *(number (optional)) – size of arrow head*

• **font_size** *(number (optional)) – Label font size. Default is 14.*

• **color** *(matplotlib color (optional)) – Arrow color. Default is black.*

Returns

Return type  **[line,arrow_head,text]**
WrightTools Documentation, Release 3.2.7

WrightTools.diagrams.delay module

Delay space.

WrightTools.diagrams.delay.label_sectors(*, labels=['I', 'II', 'IV', 'VI', 'V', 'III'], ax=None, lw=2, lc='k', cs=None, c_zlevel=2, c_alpha=0.5, fontsize=40)

Label the six time-orderings in a three-pulse experiment.

Parameters

- `labels (list of strings)` – Labels to place within sectors, starting in the upper left and proceeding clockwise. Default is ['I', 'II', 'IV', 'VI', 'V', 'III'].
- `ax (matplotlib axis object (optional))` – Axis to label. If None, uses current axis. Default is None.
- `cs (list of matplotlib colors (optional))` – Color to label sectors. If None, sectors are not colored. Default is None.
- `c_zlevel (number (optional))` – Matplotlib zlevel of color. Default is 2.
- `c_alpha (number between 0 and 1.)` – Transparency of color. Default is 0.5

1.10.5 WrightTools.exceptions module

Custom exception types.

<table>
<thead>
<tr>
<th>Exception Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimensionalityError</td>
<td>Expected, received</td>
</tr>
<tr>
<td>EntireDatasetInMemoryWarning</td>
<td>Warn when an entire dataset is taken into memory at once.</td>
</tr>
<tr>
<td>FileExistsError</td>
<td>Raised when trying to create a file or directory which already exists.</td>
</tr>
<tr>
<td>MultidimensionalAxisError</td>
<td>Error for when operation does not support Multidimensional Axes.</td>
</tr>
<tr>
<td>NameNotUniqueError</td>
<td>(name)</td>
</tr>
<tr>
<td>ObjectExistsWarning</td>
<td>Warn that an HDF5 object already exists when a new one is requested.</td>
</tr>
<tr>
<td>TypeError</td>
<td>Raised when an operation or function is applied to an object of inappropriate type.</td>
</tr>
<tr>
<td>UnitsError</td>
<td>Expected, received</td>
</tr>
<tr>
<td>ValueError</td>
<td>Raised when an argument has the right type but an inappropriate value.</td>
</tr>
<tr>
<td>VisibleDeprecationWarning</td>
<td></td>
</tr>
<tr>
<td>WrightToolsException</td>
<td>WrightTools Base Exception.</td>
</tr>
<tr>
<td>WrightToolsWarning</td>
<td>WrightTools Base Warning.</td>
</tr>
<tr>
<td>WrongFileTypeWarning</td>
<td></td>
</tr>
</tbody>
</table>
WrightTools.exceptions.DimensionalityError

exception WrightTools.exceptions.DimensionalityError (expected, received) DimensionalityError.

WrightTools.exceptions.EntireDatasetInMemoryWarning

exception WrightTools.exceptions.EntireDatasetInMemoryWarning
   Warn when an entire dataset is taken into memory at once.
   Such operations may lead to memory overflow errors for large datasets.
   Warning ignored by default.

WrightTools.exceptions.FileExistsError

exception WrightTools.exceptions.FileExistsError
   Raised when trying to create a file or directory which already exists.
   Corresponds to errno EEXIST.

WrightTools.exceptions.MultidimensionalAxisError

exception WrightTools.exceptions.MultidimensionalAxisError (axis, operation) MultidimensionalAxisError
   Error for when operation does not support Multidimensional Axes.

WrightTools.exceptions.NameNotUniqueError

exception WrightTools.exceptions.NameNotUniqueError (name=None) NameNotUniqueError.

WrightTools.exceptions.ObjectExistsWarning

exception WrightTools.exceptions.ObjectExistsWarning
   Warn that an HDF5 object already exists when a new one is requested.

WrightTools.exceptions>TypeError

exception WrightTools.exceptions.TypeError
   Raised when an operation or function is applied to an object of inappropriate type.
   The associated value is a string giving details about the type mismatch.
WrightTools.exceptions.UnitsError

exception WrightTools.exceptions.UnitsError (expected, received)
Units Error.

WrightTools.exceptions.ValueError

exception WrightTools.exceptions.ValueError
Raised when an argument has the right type but an inappropriate value.

WrightTools.exceptions.VisibleDeprecationWarning

exception WrightTools.exceptions.VisibleDeprecationWarning
VisibleDepreciationWarning.

WrightTools.exceptions.WrightToolsException

exception WrightTools.exceptions.WrightToolsException
WrightTools Base Exception.

WrightTools.exceptions.WrightToolsWarning

exception WrightTools.exceptions.WrightToolsWarning
WrightTools Base Warning.

WrightTools.exceptions.WrongFileTypeWarning

exception WrightTools.exceptions.WrongFileTypeWarning
WrongFileTypeWarning.

1.10.6 WrightTools.kit module

General-purpose tool kit.

INI(filepath) Handle communication with an INI file.
Spline(xi, yi[, k, s, ignore_nans]) Spline.
TimeStamp([at, timezone]) Class for representing a moment in time.
Timer([verbose]) Context manager for timing code.
closest_pair(arr[, give]) Find the pair of indices corresponding to the closest elements in an array.
diff(xi, yi[, order]) Take the numerical derivative of a 1D array.
discover_dimensions(arr, cols) Discover the dimensions of a flattened multidimensional array.
enforce_mask_shape(mask, shape) Reduce a boolean mask to fit a given shape.
fft(xi, yi[, axis]) Take the 1D FFT of an N-dimensional array and return “sensible” properly shifted arrays.
flatten_list(items[, seqtypes, in_place]) Flatten an irregular sequence.
fluence(power_mW, color, beam_radius, ...[, ...]) Calculate the fluence of a beam.

Continued on next page
### Table 19 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_index(lis, argument)</code></td>
<td>Find the index of an item, given either the item or index as an argument.</td>
</tr>
<tr>
<td><code>get_path_matching(name)</code></td>
<td>Get path matching a name.</td>
</tr>
<tr>
<td><code>glob_handler(extension[, folder, identifier])</code></td>
<td>Return a list of all files matching specified inputs.</td>
</tr>
<tr>
<td><code>intersperse(lis, value)</code></td>
<td>Put value between each existing item in list.</td>
</tr>
<tr>
<td><code>joint_shape(*args)</code></td>
<td>Given a set of arrays, return the joint shape.</td>
</tr>
<tr>
<td><code>leastsqfitter(p0, datax, datay, function[, ...])</code></td>
<td>Conveniently call scipy.optimize.leastsq().</td>
</tr>
<tr>
<td><code>mask_reduce(mask)</code></td>
<td>Reduce a boolean mask, removing all false slices in any dimension.</td>
</tr>
<tr>
<td><code>mono_resolution(grooves_per_mm, slit_width, ...)</code></td>
<td>Calculate the resolution of a monochromator.</td>
</tr>
<tr>
<td><code>nm_width(center, width[, units])</code></td>
<td>Given a center and width, in energy units, get back a width in nm.</td>
</tr>
<tr>
<td><code>orthogonal(*args)</code></td>
<td>Determine if a set of arrays are orthogonal.</td>
</tr>
<tr>
<td><code>pairwise(iterable)</code></td>
<td>s -&gt; (s0,s1), (s1,s2), (s2, s3), ...</td>
</tr>
<tr>
<td><code>remove_nans_1D(*args)</code></td>
<td>Remove nans in a set of 1D arrays.</td>
</tr>
<tr>
<td><code>share_nans(*arrs)</code></td>
<td>Take a list of nD arrays and return a new list of nD arrays.</td>
</tr>
<tr>
<td><code>smooth_1D(arr[, n, smooth_type])</code></td>
<td>Smooth 1D data using a window function.</td>
</tr>
<tr>
<td><code>string2identifier(s)</code></td>
<td>Turn a string into a valid python identifier.</td>
</tr>
<tr>
<td><code>symmetric_sqrt(x[, out])</code></td>
<td>Compute the ‘symmetric’ square root: sign(x) * sqrt(abs(x)).</td>
</tr>
<tr>
<td><code>timestamp_from_RFC3339(RFC3339)</code></td>
<td>Generate a Timestamp object from a RFC3339 formatted string.</td>
</tr>
<tr>
<td><code>unique(arr[, tolerance])</code></td>
<td>Return unique elements in 1D array, within tolerance.</td>
</tr>
<tr>
<td><code>valid_index(index, shape)</code></td>
<td>Get a valid index for a broadcastable shape.</td>
</tr>
<tr>
<td><code>zoom2D(xi, yi, zi[, xi_zoom, yi_zoom, ...])</code></td>
<td>Zoom a 2D array, with axes.</td>
</tr>
</tbody>
</table>

### WrightTools.kit.INI

**class WrightTools.kit.INI (filepath)**

Bases: object

Handle communication with an INI file.

```python
__init__(filepath)
```

Create an INI handler object.

**Parameters**

filepath *(path-like)* – Filepath.

```python
add_section(section)
```

Add section.

**Parameters**

section *(string)* – Section to add.

```python
clear()
```

Remove all contents from file. Use with extreme caution.

**Warning:** This is a destructive action.

**property dictionary**

Get a python dictionary of contents.

---

1.10. WrightTools API 107
get_options \((section) \rightarrow \text{list}\)

List the options in a section.

**Parameters**

- **section** \((\text{string})\) – The section to investigate.

**Returns**

The options within the given section.

**Return type**

list of strings

has_option \((section, option) \rightarrow \text{bool}\)

Test if file has option.

**Parameters**

- **section** \((\text{string})\) – Section.
- **option** \((\text{string})\) – Option.

**Returns**

**Return type**

boolean

has_section \((section) \rightarrow \text{bool}\)

Test if file has section.

**Parameters**

- **section** \((\text{string})\) – Section.

**Returns**

**Return type**

boolean

read \((section, option)\)

Read from file.

**Parameters**

- **section** \((\text{string})\) – Section.
- **option** \((\text{string})\) – Option.

**Returns**

Value.

**Return type**

string

property sections

List of sections.

write \((section, option, value)\)

Write to file.

**Parameters**

- **section** \((\text{string})\) – Section.
- **option** \((\text{string})\) – Option.
- **value** \((\text{string})\) – Value.
WrightTools.kit.Spline

class WrightTools.kit.Spline(xi, yi, k=3, s=1000, ignore_nans=True)

Bases: object

Spline.

__call__(*args, **kwargs)
Evaluate.

__init__(xi, yi, k=3, s=1000, ignore_nans=True)
Initialize.

Parameters

• \textbf{xi} (1D array) – x points.
• \textbf{yi} (1D array) – y points.
• \textbf{k} (integer (optional)) – Degree of smoothing. Must be between 1 and 5 (inclusive). Default is 3.
• \textbf{s} (integer (optional)) – Positive smoothing factor used to choose the number of knots. Number of knots will be increased until the smoothing condition is satisfied:

\[
\sum((w[i] * (y[i]-spl(x[i])))**2, axis=0) <= s
\]

If 0, spline will interpolate through all data points. Default is 1000.
• \textbf{ignore_nans} (boolean (optional)) – Toggle removal of nans. Default is True.

Note: Use \textbf{k=1} and \textbf{s=0} for a linear interpolation.

WrightTools.kit.TimeStamp

class WrightTools.kit.TimeStamp(at=None, timezone='local')

Bases: object

Class for representing a moment in time.

property RFC3339
RFC339.

Link to RFC3339.

property RFC5322
RFC5322.

Link to RFC5322.

__init__(at=None, timezone='local')
Create a TimeStamp object.

Parameters

• \textbf{at} (float (optional)) – Seconds since epoch (unix time). If None, current time will be used. Default is None.
• \textbf{timezone} (string or integer (optional)) – String (one in \{‘local’, ‘utc’\}) or seconds offset from UTC. Default is local.
unix
   Seconds since epoch (unix time).
   Type float

date
   Date.
   Type string

hms
   Hours, minutes, seconds.
   Type string

human
   Representation of the timestamp meant to be human readable.
   Type string

legacy
   Legacy WrightTools timestamp representation.
   Type string

RFC3339
   RFC3339 representation (recommended for most applications).
   Type string

RFC5322
   RFC5322 representation.
   Type string

path
   Representation of the timestamp meant for inclusion in filepaths.
   Type string

property date
   year-month-day.

property hms
   Get time formatted.
   HH:MM:SS

property human
   Human-readable timestamp.

property path
   Timestamp for placing into filepaths.

WrightTools.kit.Timer

class WrightTools.kit.Timer(verbosetrue)
   Bases: object

   Context manager for timing code.

   >>> with Timer():
   ...     your_code()
interval
    Timedelta for end - start in seconds.

__enter__(progress=None)

__exit__(type, value, traceback)

__init__(verbose=True)
    Initialize self. See help(type(self)) for accurate signature.

WrightTools.kit.closest_pair

WrightTools.kit.closest_pair(arr, give='indices')
    Find the pair of indices corresponding to the closest elements in an array.
    If multiple pairs are equally close, both pairs of indicies are returned. Optionally returns the closest distance itself.
    I am sure that this could be written as a cheaper operation. I wrote this as a quick and dirty method because I need it now to use on some relatively small arrays. Feel free to refactor if you need this operation done as fast as possible. - Blaise 2016-02-07

Parameters

• arr (numpy.ndarray) – The array to search.
• give ({'indicies', 'distance'} (optional)) – Toggle return behavior. If 'distance', returns a single float - the closest distance itself. Default is indicies.

Returns

List containing lists of two tuples: indicies the nearest pair in the array.

>>> arr = np.array([0, 1, 2, 3, 3, 4, 5, 6, 1])
>>> closest_pair(arr)

Return type  list of lists of two tuples

WrightTools.kit.diff

WrightTools.kit.diff(xi, yi, order=1) → numpy.ndarray
    Take the numerical derivative of a 1D array.
    Output is mapped onto the original coordinates using linear interpolation. Expects monotonic xi values.

Parameters

• xi (1D array-like) – Coordinates.
• yi (1D array-like) – Values.
• order (positive integer (optional)) – Order of differentiation.

Returns  Numerical derivative. Has the same shape as the input arrays.

Return type  1D numpy array
WrightTools.kit.discover_dimensions

WrightTools.kit.discover_dimensions(arr, cols) → collections.OrderedDict
Discover the dimensions of a flattened multidimensional array.

Parameters

• **arr** (*2D numpy ndarray*) – Array in [col, value].
• **cols** (*dictionary*) – Dictionary with column names as keys, and idx, tolerance and units as values.

Returns expression: points
Return type dictionary

WrightTools.kit.enforce_mask_shape

WrightTools.kit.enforce_mask_shape(mask, shape)
Reduce a boolean mask to fit a given shape.

Parameters

• **mask** (*ndarray with bool dtype*) – The mask which is to be reduced
• **shape** (*tuple of int*) – Shape which broadcasts to the mask shape.

Returns

Return type A boolean mask, collapsed along axes where the shape given has one element.

WrightTools.kit.fft

WrightTools.kit.fft(xi, yi, axis=0) → tuple
Take the 1D FFT of an N-dimensional array and return “sensible” properly shifted arrays.

Parameters

• **xi** (*numpy.ndarray*) – 1D array over which the points to be FFT’ed are defined
• **yi** (*numpy.ndarray*) – ND array with values to FFT
• **axis** (*int*) – axis of yi to perform FFT over

Returns

• **xi** (*1D numpy.ndarray*) – 1D array. Conjugate to input xi. Example: if input xi is in the time domain, output xi is in frequency domain.
• **yi** (*ND numpy.ndarray*) – FFT. Has the same shape as the input array (yi).
**WrightTools.kit.flatten_list**

`WrightTools.kit.flatten_list(items, seqtypes=(<class 'list'>, <class 'tuple'>), in_place=True)`

Flatten an irregular sequence.

Works generally but may be slower than it could be if you can make assumptions about your list.

**Source**

**Parameters**

- `items (iterable)` – The irregular sequence to flatten.
- `seqtypes (iterable of types (optional))` – Types to flatten. Default is (list, tuple).
- `in_place (boolean (optional))` – Toggle in_place flattening. Default is True.

**Returns** Flattened list.

**Return type** list

**Examples**

```python
>>> l = [[[1, 2, 3], [4, 5]], 6]
>>> wt.kit.flatten_list(l)
[1, 2, 3, 4, 5, 6]
```

**WrightTools.kit.fluence**

`WrightTools.kit.fluence(power_mW, color, beam_radius, reprate_Hz, pulse_width, color_units='wn', beam_radius_units='mm', pulse_width_units='fs_t', area_type='even') → tuple`

Calculate the fluence of a beam.

**Parameters**

- `power_mW (number)` – Time integrated power of beam.
- `color (number)` – Color of beam in units.
- `beam_radius (number)` – Radius of beam in units.
- `reprate_Hz (number)` – Laser repetition rate in inverse seconds (Hz).
- `pulse_width (number)` – Pulsewidth of laser in units
- `color_units (string (optional))` – Valid wt.units color unit identifier. Default is wn.
- `beam_radius_units (string (optional))` – Valid wt.units distance unit identi-
  fier. Default is mm.
- `pulse_width_units (number)` – Valid wt.units time unit identifier. Default is fs.
- `area_type (string (optional))` – Type of calculation to accomplish for Gaussian
  area. even specifies a flat-top calculation average specifies a Gaussian average within the
  FWHM Default is even.

**Returns** Fluence in uJ/cm², photons/cm², and peak intensity in GW/cm²

**Return type** tuple
**WrightTools.kit.get_index**

WrightTools.kit.get_index(lis, argument)

Find the index of an item, given either the item or index as an argument.

Particularly useful as a wrapper for arguments like channel or axis.

**Parameters**
- lis (list) – List to parse.
- argument (int or object) – Argument.

**Returns** Index of chosen object.

**Return type** int

**WrightTools.kit.get_path_matching**

WrightTools.kit.get_path_matching(name)

Get path matching a name.

**Parameters** name (string) – Name to search for.

**Returns** Full filepath.

**Return type** string

**WrightTools.kit.glob_handler**

WrightTools.kit.glob_handler(extension, folder=None, identifier=None)

Return a list of all files matching specified inputs.

**Parameters**
- extension (string) – File extension.
- folder (string (optional)) – Folder to search within. Default is None (current working directory).
- identifier (string) – Unique identifier. Default is None.

**Returns** Full path of matching files.

**Return type** list of strings

**WrightTools.kit.intersperse**

WrightTools.kit.intersperse(lis, value)

Put value between each existing item in list.

**Parameters**
- lis (list) – List to intersperse.
- value (object) – Value to insert.

**Returns** interspersed list

**Return type** list
WrightTools.kit.joint_shape

WrightTools.kit.joint_shape(*args) → tuple
Given a set of arrays, return the joint shape.

Parameters
  args (array-likes) –

Returns
  Joint shape.

Return type
tuple of int

WrightTools.kit.leastsqfitter

WrightTools.kit.leastsqfitter(p0, datax, datay, function, verbose=False, cov_verbose=False)
Conveniently call scipy.optimize.leastsq().

Returns fit parameters and their errors.

Parameters
  • p0 (list) – list of guess parameters to pass to function
  • datax (array) – array of independent values
  • datay (array) – array of dependent values
  • function (function) – function object to fit data to. Must be of the callable form
    function(p, x)
  • verbose (bool) – toggles printing of fit time, fit params, and fit param errors
  • cov_verbose (bool) – toggles printing of covariance matrix

Returns
  • pfit_leastsq (list) – list of fit parameters. s.t. the error between datay and function(p, datax)
    is minimized
  • perr_leastsq (list) – list of fit parameter errors (1 std)

WrightTools.kit.mask_reduce

WrightTools.kit.mask_reduce(mask)
Reduce a boolean mask, removing all false slices in any dimension.

Parameters
  mask (ndarray with bool dtype) – The mask which is to be reduced

Returns

Return type
  A boolean mask with no all False slices.
WrightTools.kit.mono_resolution

WrightTools.kit.mono_resolution(grooves_per_mm, slit_width, focal_length, output_color, output_units='wn') → float

Calculate the resolution of a monochromator.

Parameters

- **grooves_per_mm**(number) – Grooves per millimeter.
- **slit_width**(number) – Slit width in microns.
- **focal_length**(number) – Focal length in mm.
- **output_color**(number) – Output color in nm.
- **output_units**(string (optional)) – Output units. Default is wn.

Returns Resolution.
Return type float

WrightTools.kit.nm_width

WrightTools.kit.nm_width(center, width, units='wn') → float

Given a center and width, in energy units, get back a width in nm.

Parameters

- **center**(number) – Center (in energy units).
- **width**(number) – Width (in energy units).
- **units**(string (optional)) – Input units. Default is wn.

Returns Width in nm.
Return type number

WrightTools.kit.orthogonal

WrightTools.kit.orthogonal(*args) → bool

Determine if a set of arrays are orthogonal.

Parameters **args**(array-likes or array shapes)–

Returns Array orthogonality condition.
Return type bool

WrightTools.kit.pairwise

WrightTools.kit.pairwise(iterable)

s \rightarrow (s_0, s_1), (s_1, s_2), (s_2, s_3), …

Originally from itertools docs

Parameters **iterable**(iterable) – Iterable from which to produce pairs

Returns Generator which produces pairwise tuples
Return type generator
**WrightTools.kit.remove_nans_1D**

WrightTools.kit.remove_nans_1D(*args) → tuple

Remove nans in a set of 1D arrays.

Removes indices in all arrays if any array is nan at that index. All input arrays must have the same size.

- **Parameters**
  - *args (1D arrays)*

- **Returns**
  - Tuple of 1D arrays in same order as given, with nan indices removed.

- **Return type**
  - tuple

**WrightTools.kit.share_nans**

WrightTools.kit.share_nans(*arrs) → tuple

Take a list of nD arrays and return a new list of nD arrays.

The new list is in the same order as the old list. If one indexed element in an old array is nan then every element for that index in all new arrays in the list is then nan.

- **Parameters**
  - *arrs (nD arrays.)*

- **Returns**
  - List of nD arrays in same order as given, with nan indices synchronized.

- **Return type**
  - list

**WrightTools.kit.smooth_1D**

WrightTools.kit.smooth_1D(arr, n=10, smooth_type='flat') → numpy.ndarray

Smooth 1D data using a window function.

Edge effects will be present.

- **Parameters**
  - *arr (array_like)* – Input array, 1D.
  - *n (int (optional))* – Window length.
  - *smooth_type (flat, hanning, hamming, bartlett, blackman) (optional)* – Type of window function to convolve data with. ‘flat’ window will produce a moving average smoothing.

- **Returns**
  - Smoothed 1D array.

- **Return type**
  - array_like

**WrightTools.kit.string2identifier**

WrightTools.kit.string2identifier(s)

Turn a string into a valid python identifier.

Currently only allows ASCII letters and underscore. Illegal characters are replaced with underscore. This is slightly more opinionated than python 3 itself, and may be refactored in future (see PEP 3131).

- **Parameters**
  - *s (string)* – string to convert

- **Returns**
  - valid python identifier.

- **Return type**
  - str
WrightTools.kit.svd

WrightTools.kit.svd(a, i=None) → tuple
Singular Value Decomposition.
Factors the matrix \( a \) as \( u * \text{np.diag}(s) * v \), where \( u \) and \( v \) are unitary and \( s \) is a 1D array of \( a \)'s singular values.

**Parameters**
- **a** (*array_like*) – Input array.
- **i** (*int or slice (optional)*) – What singular value “slice” to return. Default is None which returns unitary 2D arrays.

**Returns** Decomposed arrays in order \( u, v, s \)

**Return type** tuple

---

WrightTools.kit.symmetric_sqrt

WrightTools.kit.symmetric_sqrt(x, out=None)
Compute the `symmetric` square root: \( \text{sign}(x) * \sqrt{\text{abs}(x)} \).

**Parameters**
- **x** (*array_like or number*) – Input array.
- **out** (*ndarray, None, or tuple of ndarray and None (optional)*) – A location into which the result is stored. If provided, it must have a shape that the inputs broadcast to. If not provided or None, a freshly-allocated array is returned. A tuple (possible only as a keyword argument) must have length equal to the number of outputs.

**Returns** Symmetric square root of arr.

**Return type** np.ndarray

---

WrightTools.kit.timestamp_from_RFC3339

WrightTools.kit.timestamp_from_RFC3339(RFC3339)
Generate a Timestamp object from a RFC3339 formatted string.

Link to RFC3339

**Parameters** RFC3339 (*string*) – RFC3339 formatted string.

**Returns**

**Return type** WrightTools.kit.TimeStamp
WrightTools.kit.unique

WrightTools.kit.unique(arr, tolerance=1e-06) → numpy.ndarray

Return unique elements in 1D array, within tolerance.

Parameters

- arr (array_like) – Input array. This will be flattened if it is not already 1D.
- tolerance (number (optional)) – The tolerance for uniqueness.

Returns

The sorted unique values.

Return type

array

WrightTools.kit.valid_index

WrightTools.kit.valid_index(index, shape) → tuple

Get a valid index for a broadcastable shape.

Parameters

- index (tuple) – Given index.
- shape (tuple of int) – Shape.

Returns

Valid index.

Return type

tuple

WrightTools.kit.zoom2D

WrightTools.kit.zoom2D(xi, yi, zi, xi_zoom=3.0, yi_zoom=3.0, order=3, mode='nearest', cval=0.0)

Zoom a 2D array, with axes.

Parameters

- xi (1D array) – x axis points.
- yi (1D array) – y axis points.
- zi (2D array) – array values. Shape of (x, y).
- xi_zoom (float (optional)) – Zoom factor along x axis. Default is 3.
- yi_zoom (float (optional)) – Zoom factor along y axis. Default is 3.
- order (int (optional)) – The order of the spline interpolation, between 0 and 5. Default is 3.
- mode ({'constant', 'nearest', 'reflect', or 'wrap'}) – Points outside the boundaries of the input are filled according to the given mode. Default is nearest.
- cval (scalar (optional)) – Value used for constant mode. Default is 0.0.
1.10.7 WrightTools.open

WrightTools.open(filepath, edit_local=False)
Open any wt5 file, returning the top-level object (data or collection).

Parameters

• filepath (path-like) – Path to file. Can be either a local or remote file (http/ftp). Can be compressed with gz/bz2, decompression based on file name.

• edit_local (boolean (optional)) – If True, the file itself will be opened for editing. Otherwise, a copy will be created. Default is False.

Returns Root-level object in file.
Return type WrightTools Collection or Data

1.10.8 WrightTools.units module

Unit and label handling in WrightTools.

WrightTools.units.converter

WrightTools.units.converter(val, current_unit, destination_unit)
Convert from one unit to another.

Parameters

• val (number) – Number to convert.

• current_unit (string) – Current unit.

• destination_unit (string) – Destination unit.

Returns Converted value.
Return type number

WrightTools.units.get_symbol

WrightTools.units.get_symbol(units) → str
Get default symbol type.

Parameters units_str (string) – Units.

Returns LaTeX formatted symbol.
Return type string
WrightTools.units.get_valid_conversions

WrightTools.units.get_valid_conversions(units) → tuple

WrightTools.units.is_valid_conversion

WrightTools.units.is_valid_conversion(a, b) → bool

WrightTools.units.kind

WrightTools.units.kind(units)
  Find the kind of given units.

  Parameters
  units (string) – The units of interest

  Returns
  The kind of the given units. If no match is found, returns None.

  Return type
  string

1.11 Gallery

Note: Click here to download the full example code

1.11.1 Plot colormap components

Quickly plot the RGB components of a colormap.

![WrightTools](image)

1.11. Gallery
import WrightTools as wt

cmap = wt.artists.colormaps["default"]
wt.artists.plot_colormap_components(cmap)

Total running time of the script: 0.241 seconds

Note: Click here to download the full example code

1.11.2 Quick 2D Signed

A quick 2D plot of a signed channel.

perovskite_TA
\[ \tau_{21} = 0.821 \text{ fs} \]

Out:
import WrightTools as wt
from WrightTools import datasets

p = datasets.wt5.vlp0p0_perovskite_TA
data = wt.open(p)
wt.artists.quick2D(data, "w1=wm", "w2", at="d2": [0, "fs"], verbose=False)

Total running time of the script: ( 0 minutes 1.309 seconds)

Note: Click here to download the full example code

1.11.3 Quick 2D

A quick 2D plot.
```python
import WrightTools as wt
from WrightTools import datasets

ps = datasets.KENT.LDS821_TRSF
data = wt.data.from_KENT(ps, ignore=['d1', 'd2', 'wm'], verbose=False)
wt.artists.quick2D(data, verbose=False)

Total running time of the script: ( 0 minutes 2.505 seconds)
```
1.11.4 Resonance Raman

A Resonance Raman plot.

LDS821_514nm_80mW

![Resonance Raman plot]

Out:

data created at /tmp/4lve2jdo.wt5::/
    range: 786.04325 to 2059.0301 (wn)
    size: 1340

```python
import WrightTools as wt
from WrightTools import datasets
p = datasets.BrunoldRaman.LDS821_514nm_80mW
data = wt.data.from_BrunoldRaman(p)
data.convert("wn", verbose=False)
```

(continues on next page)
wt.artists.quick1D(data)

**Total running time of the script:** ( 0 minutes 0.194 seconds)

**Note:**  Click *here* to download the full example code

### 1.11.5 Quick 1D

A quick 1D plot.

\[
\tilde{v}_2 = 1.52 \times 10^3 \text{ cm}^{-1}
\]

![Graph showing a plot with signal on the y-axis and \(\tilde{v}_1 (\text{cm}^{-1})\) on the x-axis.

Out:

[]

```python
import WrightTools as wt
from WrightTools import datasets

ps = datasets.KENT.LDS821_TRSF
```

(continues on next page)
1.11.6 Gradient

Demonstration of the gradient method.
```python
import numpy as np
import WrightTools as wt

data = wt.data.Data()
data.create_variable("w1", np.linspace(-10, 10, 100))
data.create_channel("sig", 1 / (np.pi * (1 + (data.w1[:]) - 1) ** 2))
data.transform("w1")
data.gradient("w1")

wt.artists.quick1D(data)
wt.artists.quick1D(data, channel="sig_wl_gradient")
```

Total running time of the script: ( 0 minutes 0.379 seconds)

Note: Click here to download the full example code
1.11.7 Plotting Multiple Lines

A quick demonstration of how to plot multiple lines on the same set of axes, using `create_figure()` to have a set of axes which can plot data objects directly.

The dataset is a set of optical filters transmission spectra.

```
import WrightTools as wt
from WrightTools import datasets
from matplotlib import pyplot as plt
```

Out:

```
11 data objects successfully created from Cary file:
  0: 600LP
  1: 600LP1
  2: 600LP2
  3: 550LP
  4: 600SP800N
  5: 600SP800N1
  6: 530SP
  7: GSBS
  8: 550LP2
  9: 530SP2
 10: 530SP_HI
<matplotlib.legend.Legend object at 0x7f050e3e9898>
```
p = datasets.Cary.filters
col = wt.collection.from_Cary(p)

fig, gs = wt.artists.create_figure(width="double", default_aspect=.5)
ax = plt.subplot(gs[0])

for data in col.values():
    if data.natural_name in ("600LP", "550LP2"):
        continue
    data.convert("wn", verbose=False)
    ax.plot(data, label=data.natural_name)

ax.set_ylabel("%T")
ax.set_xlabel("Frequency (cm$^{-1}$)"
ax.legend()

Total running time of the script: ( 0 minutes 0.492 seconds)

Note: Click here to download the full example code

1.11.8 Tune test

An example of transform on a tune test.
import matplotlib.pyplot as plt

import WrightTools as wt
from WrightTools import datasets

p = datasets.PyCMDS.w1_wa_000
data = wt.data.from_PyCMDS(p)

fig, gs = wt.artists.create_figure(width="double", cols=[1, 0.25, 1, "cbar"])

# as taken
ax = plt.subplot(gs[0, 0])
apolor(data)
wt.artists.set_ax_labels(xlabel=data.w1__e__wm.label, ylabel=data.wa.label)
ax.grid()
ax.set_title("as taken", fontsize=20)

# transformed
ax = plt.subplot(gs[0, 2])
data.transform("w1", "wa-w1")
apolor(data)
wt.artists.set_ax_labels(xlabel=data.w1.label, ylabel=data.wa__m__w1.label)
ax.grid()
ax.set_title("transformed", fontsize=20)

# colorbar
cax = plt.subplot(gs[0, -1])
wt.artists.plot_colorbar(cax, label="intensity")

Total running time of the script: ( 0 minutes 1.390 seconds)

Note: Click here to download the full example code
1.11.9 DOVE transform

An example of transform on a dataset from a DOVE experiment.

```python
import matplotlib.pyplot as plt
import WrightTools as wt
from WrightTools import datasets

p = datasets.KENT.LDS821_DOVE
data = wt.data.from_KENT(p, ignore=['d1', 'd2', 'wm'], verbose=False)

fig, gs = wt.artists.create_figure(width='double', cols=[1, 1, 'cbar'], wspace=0.7)

# as taken
ax = plt.subplot(gs[0, 0])
data.transform('w2', 'w1')
ax.pcolor(data)
wt.artists.set_ax_labels(xlabel=data.w2.label, ylabel=data.w1.label)
ax.grid()
ax.set_title('as taken', fontsize=20)

# transformed
ax = plt.subplot(gs[0, 1])
data.transform('w2', 'w1-w2')
ax.pcolor(data)
```

Out:

```
<matplotlib.colorbar.ColorbarBase object at 0x7f050cf4a8d0>
```
wt.artists.set_ax_labels(xlabel=data.w2.label)
ax.grid()
ax.set_title("transformed", fontsize=20)

# colorbar
cax = plt.subplot(gs[0, -1])
wt.artists.plot_colorbar(cax, label="Intensity")

**Total running time of the script:** ( 0 minutes 1.882 seconds)

**Note:** Click [here](#) to download the full example code

### 1.11.10 Fringes transform

An example of transform on a dataset containing fringes.

Out:

Correction factor applied to d1
Correction factor applied to d2
data created at /tmp/558ixt84.wt5::/
    axes: ('w2', 'w1')
    shape: (81, 81)
axis w2 converted from wn to wn
axis w1 converted from wn to wn
axis wm converted from nm to wn
axis w1 converted from wn to wn

<matplotlib.colorbar.ColorbarBase object at 0x7f050db50128>
```python
import matplotlib.pyplot as plt

import WrightTools as wt
from WrightTools import datasets

p = datasets.PyCMDS.w2_w1_000
data = wt.data.from_PyCMDS(p)

# to amplitude level
data.signal_mean.symmetric_root(2)

# as taken
fig, gs = wt.artists.create_figure(width="double", cols=[1, 1, "cbar"])

ax = plt.subplot(gs[0, 0])
ax.pcolor(data)
wt.artists.set_ax_labels(xlabel=data.w2.label, ylabel=data.w1.label)
ax.grid()
ax.set_title("as taken", fontsize=20)

# transformed
ax = plt.subplot(gs[0, 1])
data.transform("wm", "w1")
data.convert("wn")
ax.pcolor(data)
wt.artists.set_ax_labels(xlabel=data.wm.label, yticks=False)
ax.grid()
ax.set_title("transformed", fontsize=20)

# colorbar
cax = plt.subplot(gs[0, -1])
wt.artists.plot_colorbar(cax, label="amplitude")
```

Total running time of the script: ( 0 minutes 1.547 seconds)

Note: Click [here](#) to download the full example code

### 1.11.11 Level

Leveling a dataset.
import matplotlib.pyplot as plt

import WrightTools as wt
from WrightTools import datasets

fig, gs = wt.artists.create_figure(width="double", cols=[1, 1,"cbar"])

p = datasets.wt5.v1p0p1_MoS2_TrEE_movie
data = wt.open(p)
data.convert("eV")
data.ai0.symmetric_root(2)

# as taken
ax = plt.subplot(gs[0, 0])
chop = data.chop("w1=wm", "d2", at={"w2": [1.7, "eV"]})[0]
chop.ai0.null = chop.ai0.min()  # only for example
ax.pcolor(chop)
ax.contour(chop)

(continues on next page)
# leveled

```python
ax = plt.subplot(gs[0, 1])
data.level(0, 2, -3)
chop = data.chop("w1=wm", "d2", at={"w2": [1.7, "eV"]})[0]
chop.ai0.clip(min=0, replace='value')
ax.pcolor(chop, vmin=0)
ax.contour(chop)
```

# label

```python
wt.artists.set_fig_labels(xlabel=data.w1__e__wm.label, ylabel=data.d2.label)
```

# colorbar

```python
cax = plt.subplot(gs[0, -1])
wt.artists.plot_colorbar(cax=cax, label="amplitude")
wt.artists.set_ax_labels(cax, yticks=False)
```

Total running time of the script: (0 minutes 0.951 seconds)

Note: Click [here](#) to download the full example code

### 1.11.12 Heal

An example of how heal works.

Out:

```
  warnings.warn("heal", category=wt_exceptions.EntireDatasetInMemoryWarning)
channel /healed healed in 0.026 seconds
```
```python
import numpy as np
from matplotlib import pyplot as plt
import WrightTools as wt

# create original arrays
x = np.linspace(-3, 3, 31)[None, :]
y = np.linspace(-3, 3, 31)[None, :]
arr = np.exp(-1 * (x ** 2 + y ** 2))

# create damaged array
arr2 = arr.copy()
np.random.seed(11)  # set seed for reproducibility
arr2[np.random.random(arr2.shape) < .2] = np.nan

# create data object
d = wt.data.Data()
d.create_variable("x", values=x)
d.create_variable("y", values=y)
d.create_channel("original", arr)
d.create_channel("damaged", arr2)
d.create_channel("healed", arr2)
d.transform("x", "y")

# heal
d.heal(channel="healed")

# create figure
fig, gs = wt.artists.create_figure(cols=[1, 1, 1])
for i in range(3):
    ax = plt.subplot(gs[i])
    ax.pcolor(d, channel=i)
    ax.set_title(d.channel_names[i])
    ticks = [-2, 0, 2]
wt.artists.set_fig_labels(
    xlabel=d.axes[0].label, ylabel=d.axes[1].label, xticks=ticks, yticks=ticks
)
```

**Total running time of the script:** ( 0 minutes 0.456 seconds)

**Note:** Click [here](#) to download the full example code

### 1.11.13 Split

Some examples of how splitting works.
Correction factor applied to d1
Correction factor applied to d2
data created at /tmp/cvor20z8.wt5://
    axes: ('w2', 'w1')
    shape: (81, 81)
axis w2 converted from wn to wn
axis w1 converted from wn to wn
variable w1 converted from wn to wn
variable w2 converted from wn to wn
variable wm converted from nm to wn
split data into 3 pieces along <w2>:
  0 : -inf to 7000.00 wn (26, 1)
  1 : 7000.00 to 8000.00 wn (32, 1)
(continues on next page)
from matplotlib import pyplot as plt
import WrightTools as wt
from WrightTools import datasets

d = wt.data.from_PyCMDS(datasets.PyCMDS.w2_w1_000)

d.convert("wn", convert_variables=True)

# A simple split along an axis
a = d.split("w2", [7000, 8000])

# A more complicated split along some diagonal
b = d.split("w1+w2+7000", [20400, 23000], units="wn")

# A particularly strange split

c = d.split("strange", [.2, .4])

# Plot the splits in columns
fig, gs = wt.artists.create_figure(nrows=len(c), cols=[1, 1, 1])
for j, (col, title) in enumerate(zip([a, b, c], ["Simple", "Medium", "Advanced"])):
    for i, da in enumerate(col.values()):
        ax = plt.subplot(gs[i, j])
        if i == 0:
            ax.set_title(title)
        ax.pcolor(da)
        ax.set_xlim(d.axes[0].min(), d.axes[0].max())
        ax.set_ylim(d.axes[1].min(), d.axes[1].max())

wt.artists.set_fig_labels(xlabel=d.axes[0].label, ylabel=d.axes[1].label)

Total running time of the script: ( 0 minutes 4.653 seconds)

Note: Click here to download the full example code
1.11.14 Colormaps

Different colormaps.
1.11. Gallery
import matplotlib.pyplot as plt
from matplotlib import cm

import WrightTools as wt
from WrightTools import datasets

fig, gs = wt.artists.create_figure(width="double", cols=[1, 1, "cbar"], nrows=3)

p = datasets.wt5.vlp0p1_MoS2_TrEE_movie
data = wt.open(p)
data.level(0, 2, -3)
data.convert("eV")
data.ai0.symmetric_root(2)
data = data.chop("w1=wm", "w2", at="d2": [-600, "fs"])[0]
data.ai0.normalize()
data.ai0.clip(min=0, replace="value")

def fill_row(row, cmap):
    # greyscale
    ax = plt.subplot(gs[row, 0])
    ax.pcolor(data, cmap=wt.artists.grayify_cmap(cmap))
    # color
    ax = plt.subplot(gs[row, 1])
    ax.pcolor(data, cmap=cmap)
    # cbar
    cax = plt.subplot(gs[row, 2])
    wt.artists.plot_colorbar(cax=cax, label=cmap.name, cmap=cmap)
    wt.artists.set_ax_labels(cax, yticks=False)

cmap = wt.artists.colormaps["default"]
fill_row(0, cmap)
cmap = wt.artists.colormaps["wright"]
fill_row(1, cmap)
cmap = cm.viridis
fill_row(2, cmap)

# label
wt.artists.set_fig_labels(
    xlabel=data.w1__e__wm.label, ylabel=data.w2.label, col=slice(0, 1)
)
1.11.15 Map-Variable

An example of how map-variable works.

```python
import numpy as np
from matplotlib import pyplot as plt
import WrightTools as wt

# create original arrays
x = np.linspace(1, 5, 11)[:, None]
y = np.linspace(1, 5, 11)[None, :]
arr = np.exp(-1 * (((x - 3) / .6) ** 2 + ((y - 3) / .6) ** 2))

# create data object
d = wt.data.Data(name="original")
d.create_variable("x", values=x)
d.create_variable("y", values=y)
d.create_channel("z", arr)
d.transform("x", "y")

# create new data objects
pointsx = np.linspace(x.min(), x.max(), 31) # linear spacing
pointsy = points = np.logspace(0, .7, 11) # log spacing
dx = d.map_variable("x", points=pointsx) # just linear along x
dy = d.map_variable("y", points=pointsy) # just log along y
dxy = dx.map_variable("y", points=pointsy) # linear along x and log along y
```

Out:

data mapped from (11, 11) to (31, 11)
data mapped from (11, 11) to (11, 11)
data mapped from (31, 11) to (31, 11)
ds = [d, dx, dy, dxy]
# create figure
fig, gs = wt.artists.create_figure(width="double", cols=[1, 1, 1, 1])
for i, d in enumerate(ds):
    ax = plt.subplot(gs[i])
    ax.pcolor(d)
    ax.set_title(d.natural_name)
    ax.set_xlim(1, 5)
    ax.set_ylim(1, 5)
ticks = [1, 3, 5]
wt.artists.set_fig_labels(
    xlabel=d.axes[0].label, ylabel=d.axes[1].label, xticks=ticks, yticks=ticks)

Total running time of the script: ( 0 minutes 0.528 seconds)

Note: Click here to download the full example code

1.11.16 Label delay space

Using WrightTools to label delay space.

Out:
Correction factor applied to d1
Correction factor applied to d2
Correction factor applied to d1_points
Correction factor applied to d2_points
data created at /tmp/r1pykgki.wt5://
axes: ('d1', 'd2')
import matplotlib.pyplot as plt
import WrightTools as wt
from WrightTools import datasets

fig, gs = wt.artists.create_figure(width="double", cols=[1, 1, "cbar"])  
def set_lim(ax):
    ax.set_xlim(-175, 175)
    ax.set_ylim(-175, 175)

# traditional delay space
ax = plt.subplot(gs[0, 0])
p = datasets.PyCMDS.d1_d2_000
data = wt.data.from_PyCMDS(p)
data.convert("fs")
data.channels[0].symmetric_root(2)
data.channels[0].normalize()
data.channels[0].clip(min=0, replace="value")
ax.pcolor(data)
w.t.diagrams.delay.label_sectors(ax=ax)  # using default labels
set_lim(ax)
ax.set_title(r"\mathsf{\vec{k}_1 - \vec{k}_2 + \vec{k}_{2'}}", fontsize=20)

# conjugate delay space
ax = plt.subplot(gs[0, 1])
p = datasets.PyCMDS.d1_d2_001
data = wt.data.from_PyCMDS(p)
data.convert("fs")
data.channels[0].symmetric_root(2)
data.channels[0].normalize()
data.channels[0].clip(min=0, replace="value")
ax.pcolor(data)
lables = ["\text{II}", "\text{I}", "\text{III}", "\text{V}", "\text{VI}", "\text{IV}"]
wt.diagrams.delay.label_sectors(ax=ax, labels=labels)
set_lim(ax)
ax.set_title(r'$\mathsf{\vec{k}_1 + \vec{k}_2 - \vec{k}_{2^\prime}}$', fontsize=20)

# label
wt.artists.set_fig_labels(xlabel=data.d1.label, ylabel=data.d2.label)

# colorbar
cax = plt.subplot(gs[:, -1])
wt.artists.plot_colorbar(cax=cax, label="amplitude")

**Total running time of the script:** (0 minutes 2.644 seconds)

**Note:** Click [here](#) to download the full example code

### 1.11.17 Join

Some examples of how joining works.
```python
warnings.warn("join", category=wt_exceptions.EntireDatasetInMemoryWarning)
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 5.8039471957616176 (0.0% NaN)
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 6.0 (0.0% NaN)
rep_new = new > vals
rep_vals = vals > new
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 5.8039471957616176 (0.0% NaN)
rep_new = new < vals
rep_vals = vals < new
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 6.0 (0.0% NaN)
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 6.845681782746989 (0.0% NaN)
2 datas joined to create new data:
   axes:
   x : 76 points from 0.0 to 15.0 None
   y : 51 points from 0.0 to 10.0 None
   channels:
   /z : 0.0007332240373458809 to 5.8039471957616176 (0.0% NaN)
```
```python
import numpy as np
from matplotlib import pyplot as plt
import WrightTools as wt

a = wt.data.Data(name="a")
b = wt.data.Data(name="b")
a.create_variable("x", np.linspace(0, 10, 51)[\], None)
b.create_variable("x", np.linspace(5, 15, 51)[\], None)
a.create_variable("y", np.linspace(0, 10, 51)[None, :])
b.create_variable("y", np.linspace(0, 10, 51)[None, :])
a.create_channel("z", np.sin(a.x[:]) * np.cos(a.y[:]) + 1)
b.create_channel("z", 5 * np.exp(-(b.x[:] - 10)**2) * np.exp(-(b.y[:] - 5)**2) + 1)
a.transform("x", "y")
b.transform("x", "y")

first = wt.data.join([a, b], name="first")
last = wt.data.join([a, b], method="last", name="last")
min = wt.data.join([a, b], method="min", name="min")
max = wt.data.join([a, b], method="max", name="max")
sum = wt.data.join([a, b], method="sum", name="sum")
mean = wt.data.join([a, b], method="mean", name="mean")

# Plot the splits in columns
fig, gs = wt.artists.create_figure(nrows=4, cols=[1, 1])
for i, da in enumerate([a, b, first, last, min, max, sum, mean]):
    ax = plt.subplot(gs[i])
    ax.pcolor(da, vmin=0, vmax=6)
    wt.artists.corner_text(da.natural_name, ax=ax)
    ax.set_xlim(first.axes[0].min(), first.axes[0].max())
    ax.set_ylim(first.axes[1].min(), first.axes[1].max())
wt.artists.set_fig_labels(xlabel=a.axes[0].label, ylabel=a.axes[1].label)
```

**Total running time of the script:** (0 minutes 2.958 seconds)

**Note:** Click [here](#) to download the full example code
### 1.11.18 Fill types

Different ways to plot 2D data.

```python
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import WrightTools as wt
from WrightTools import datasets
cmap = wt.artists.colormaps['default']
fig, gs = wt.artists.create_figure(width="double", nrows=2, cols=[1, 1, 1, 1, "cbar"])
```

Out:

cols recognized as v0 (19)
data created at /tmp/o5p3b0ow.wt5::/
  axes: ('d1', 'd2')
  shape: (21, 21)
˓→6/site-packages/WrightTools/data/_data.py:1055: EntireDatasetInMemoryWarning: level
  warnings.warn("level", category=wt_exceptions.EntireDatasetInMemoryWarning)
channel ai0 leveled along axis 0
<matplotlib.colorbar.ColorbarBase object at 0x7f04fcc0fef0>
```python
# get data
p = datasets.COLORS.v0p2_d1_d2_diagonal
data = wt.data.from_COLORS(p, invert_d1=False)
data.level(0, 0, 1)
data.ai0.symmetric_root(2)
data.ai0.normalize()
data.ai0.clip(min=0, replace="value")

def dot_pixel_centers(ax, xi, yi):
    for x in xi:
        ax.scatter([x] * len(xi), yi, edgecolor=None, s=5, color="k")

def decorate(ax):
    ax.set_xlim(-150, 150)
    ax.set_ylim(-150, 150)

# pcolor
ax = plt.subplot(gs[0, 0])
ax.pcolor(data, cmap=cmap)
ad.set_title("pcolor", fontsize=20)
declare(ax)
ax = plt.subplot(gs[1, 0])
ax.pcolor(data, cmap=cmap, edgecolors="k")
dot_pixel_centers(ax, data.d1.points, data.d2.points)
declare(ax)

# tripcolor
xi = data.d1.points
yi = data.d2.points
zi = data.channels[0][:].T
ax = plt.subplot(gs[0, 1])
points = [xi, yi]
x, y = tuple(np.meshgrid(xi, yi, indexing="ij"))
ax.tripcolor(x.flatten(), y.flatten(), zi.T.flatten(), cmap=cmap, vmin=0, vmax=1)
declare(ax)
ax.set_title("tripcolor", fontsize=20)
ax = plt.subplot(gs[1, 1])
ax.tripcolor(x.flatten(), y.flatten(), zi.T.flatten(), edgecolor="k", cmap=cmap,
            vmin=0, vmax=1)
declare(ax)
dot_pixel_centers(ax, xi, yi)

def plot_delaunay_edges(ax, xi, yi, zi):
    x, y = tuple(np.meshgrid(xi, yi, indexing="ij"))
    tri = matplotlib.tri.Triangulation(x.flatten(), y.flatten())
    for i, j in tri.edges:
        plt.plot([x.flatten()[i], x.flatten()[j]], [y.flatten()[i], y.flatten()[j]], c="k",
                  lw=0.25)
    ax.set_xlim(-125, 125)
    ax.set_ylim(-125, 125)
```

(continues on next page)
```python
# contourf
ax = plt.subplot(gs[0, 2])
ax.contourf(data, vmin=-1e-3)
decorate(ax)
ax.set_title("contourf", fontsize=20)
ax = plt.subplot(gs[1, 2])
ax.contourf(data, vmin=-1e-3)
plot_delaunay_edges(ax, xi, yi, zi)
dot_pixel_centers(ax, xi, yi)
decorate(ax)

# contour
ax = plt.subplot(gs[0, 3])
ax.contour(data)
decorate(ax)
ax.set_title("contour", fontsize=20)
ax = plt.subplot(gs[1, 3])
ax.contour(data)
plot_delaunay_edges(ax, xi, yi, zi)
dot_pixel_centers(ax, xi, yi)
decorate(ax)

# label
 ticks = [-100, 0, 100]
wt.artists.set_fig_labels(xlabel=data.d1.label, ylabel=data.d2.label, xticks=ticks, yticks=ticks)

# colorbar
cax = plt.subplot(gs[:, -1])
wt.artists.plot_colorbar(cax=cax, label="amplitude")
```

Total running time of the script: ( 0 minutes 5.638 seconds)

Note: Click here to download the full example code

1.11.19 Custom Figure

Example of custom figure layout, beautification, and saving.
Out:

warnings.warn("level", category=wt_exceptions.EntireDatasetInMemoryWarning)
channel ai0 leveled along axis 2
warnings.warn("smooth", category=wt_exceptions.EntireDatasetInMemoryWarning)
smoothed data

(continues on next page)
chopped data into 1 piece(s) in ['w1=wm', 'd2']
chopped data into 1 piece(s) in ['w1=wm', 'd2']
chopped data into 1 piece(s) in ['w1=wm', 'd2']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']
chopped data into 1 piece(s) in ['w1=wm']

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.
return super().add_subplot(*args, **kwargs)

'/home/docs/checkouts/readthedocs.org/user_builds/wrighttools/checkouts/stable/_examples/custom_fig.png'

import matplotlib.pyplot as plt
import numpy as np
import WrightTools as wt
from WrightTools import datasets

# obtain and process data
p = datasets.wt5.v1p0p1_MoS2_TrEE_movie
data = wt.open(p)
data.level(0, 2, -3)
data.convert("eV", convert_variables=True, verbose=False)
data.smooth([2, 2, 2])
data.ai0.symmetric_root(2)
data.ai0.normalize()
data.ai0.clip(min=0, replace="value")
#
# chop out data of interest
#d2_vals = [-50, -500]
w2_vals = [1.7, 1.8, 1.9, 2.0]
wigners = [data.chop("w1=wm", "d2", at={"w2": [w2, "eV"]})[0] for w2 in w2_vals]
traces1 = [
    data.chop("w1=wm", at={"w2": [w2, "eV"], "d2": [d2_vals[0], "fs"]})[0] for w2 in w2_vals]
traces2 = [
    data.chop("w1=wm", at={"w2": [w2, "eV"], "d2": [d2_vals[1], "fs"]})[0] for w2 in w2_vals]
```python
tracess = [traces1, traces2]

# prepare spine colors
wigner_colors = ["C0", "C1", "C2", "C3"]

# prepare figure gridspec
cols = [1, 1, "cbar"]
aspects = [[0, 0], .3]

fig, gs = wt.artists.create_figure(
    width="double", cols=cols, nrows=3, aspects=aspects, wspace=.35, hspace=.35
)

# plot wigners
indxs = [(row, col) for row in range(1, 3) for col in range(2)]
for indx, wigner, color in zip(indxs, wigners, wigner_colors):
    ax = plt.subplot(gs[indx])
    ax.pcolor(wigner, vmin=0, vmax=1)
    ax.contour(wigner)
    ax.grid()
    wt.artists.set_ax_spines(ax=ax, c=color)
    wigner.constants[0].format_spec = ".2f"
    wigner.round_spec = -1
    wt.artists.corner_text(wigner.constants[0].label, ax=ax)
    for d2, t_color in zip(d2_vals, trace_colors):
        ax.axhline(d2, color=t_color, alpha=.5, linewidth=6)

# plot traces
indxs = [(0, col) for col in range(2)]
for indx, color, traces in zip(indxs, trace_colors, tracess):
    ax = plt.subplot(gs[indx])
    for trace, w_color in zip(traces, wigner_colors):
        ax.plot(trace, color=w_color, linewidth=1.5)
    ax.grid()
    ax.set_xlim(trace.axes[0].min(), trace.axes[0].max())
    wt.artists.set_ax_spines(ax=ax, c=color)

# plot colormap
cax = plt.subplot(gs[1:3, -1])
ticks = np.linspace(data.ai0.min(), data.ai0.max(), 11)
wt.artists.plot_colorbar(cax=cax, label="amplitude", cmap="default", ticks=ticks)

# set axis labels
wt.artists.set_fig_labels(xlabel=data.w1__e__wm.label, ylabel=data.d2.label, col=slice(0, 1))

# ylabel of zeroth row
ax = plt.subplot(gs[0, 0])
ax.set_ylabel("amplitude")

# saving the figure as a png
wt.artists.savefig("custom_fig.png", fig=fig, close=False)
```

**Total running time of the script:** (0 minutes 7.930 seconds)

**Note:** Click [here](#) to download the full example code
Draw WMELs for TRIVE off diagonal.

```python
import matplotlib.pyplot as plt
import WrightTools.diagrams.WMEL as WMEL

artist = WMEL.Artist(
    size=[6, 2],
    energies=[0., 0.43, 0.57, 1.],
    state_names=['g', 'a', 'b', 'a+b']
)

artist.label_rows(['$\alpha$', '$\beta$', '$\gamma$'])
artist.label_columns(['I', 'II', 'III', 'IV', 'V', 'VI'])

# pw1 alpha
artist.add_arrow([0, 0], 0, [0, 1], 'ket', '1')
artist.add_arrow([0, 0], 1, [0, 2], 'bra', '-2')
artist.add_arrow([0, 0], 2, [2, 0], 'bra', '2')
artist.add_arrow([0, 0], 3, [1, 0], 'out')

# pw1 beta
artist.add_arrow([0, 1], 0, [0, 1], 'ket', '1')
artist.add_arrow([0, 1], 1, [0, 2], 'bra', '-2')
```

(continues on next page)
artist.add_arrow([0, 1], 2, [1, 3], "ket", "2')
artist.add_arrow([0, 1], 3, [3, 2], "out")

# pw2 alpha
artist.add_arrow([1, 0], 0, [0, 1], "ket", "1")
artist.add_arrow([1, 0], 1, [1, 3], "ket", "2')")
artist.add_arrow([1, 0], 2, [3, 1], "ket", "-2")
artist.add_arrow([1, 0], 3, [1, 0], "out")

# pw2 beta
artist.add_arrow([1, 1], 0, [0, 1], "ket", "1")
artist.add_arrow([1, 1], 1, [1, 3], "ket", "2')")
artist.add_arrow([1, 1], 2, [0, 2], "bra", "-2")
artist.add_arrow([1, 1], 3, [3, 2], "out")

# pw3 alpha
artist.add_arrow([2, 0], 0, [0, 2], "bra", "-2")
artist.add_arrow([2, 0], 1, [0, 1], "ket", "1")
artist.add_arrow([2, 0], 2, [2, 0], "bra", "2')")
artist.add_arrow([2, 0], 3, [1, 0], "out")

# pw3 beta
artist.add_arrow([2, 1], 0, [0, 2], "ket", "-2")
artist.add_arrow([2, 1], 1, [0, 1], "ket", "1")
artist.add_arrow([2, 1], 2, [1, 3], "bra", "2')")
artist.add_arrow([2, 1], 3, [3, 2], "out")

# pw4 alpha
artist.add_arrow([3, 0], 0, [0, 2], "ket", "2')")
artist.add_arrow([3, 0], 1, [2, 3], "ket", "1")
artist.add_arrow([3, 0], 2, [3, 1], "ket", "-2")
artist.add_arrow([3, 0], 3, [1, 0], "out")

# pw4 beta
artist.add_arrow([3, 1], 0, [0, 2], "ket", "2')")
artist.add_arrow([3, 1], 1, [2, 3], "ket", "1")
artist.add_arrow([3, 1], 2, [0, 2], "bra", "-2")
artist.add_arrow([3, 1], 3, [3, 2], "out")

# pw5 alpha
artist.add_arrow([4, 0], 0, [0, 2], "bra", "-2")
artist.add_arrow([4, 0], 1, [2, 0], "bra", "2')")
artist.add_arrow([4, 0], 2, [0, 1], "ket", "1")
artist.add_arrow([4, 0], 3, [1, 0], "out")

# pw5 beta
artist.add_arrow([4, 1], 0, [0, 2], "bra", "-2")
artist.add_arrow([4, 1], 1, [0, 2], "ket", "2')")
artist.add_arrow([4, 1], 2, [2, 3], "ket", "1")
artist.add_arrow([4, 1], 3, [3, 2], "out")

# pw6 alpha
artist.add_arrow([5, 0], 0, [0, 2], "ket", "2')")
artist.add_arrow([5, 0], 1, [2, 0], "ket", "-2")
artist.add_arrow([5, 0], 2, [0, 1], "ket", "1")
artist.add_arrow([5, 0], 3, [1, 0], "out")

(continues on next page)
1.11.21 WMELs: TRIVE population transfer

Draw WMELs for TRIVE population transfer.
```python
import matplotlib.pyplot as plt
import WrightTools.diagrams.WMEL as WMEL

artist = WMEL.Artist(
    size=[4, 3],
    energies=[0., 0.4, 0.5, 0.8, 0.9, 1.],
    number_of_interactions=6,
)
```
state_names=['g', '1S', '1P', '2x 1S', '1S+1P', '2x 1P'],
)
artist.label_rows([r'$\text{\mathbf{\alpha}}$', r'$\text{\mathbf{\beta}}$', r'$\text{\mathbf{\gamma}}$'])
artist.label_columns(['diag before', 'cross before', 'diag after', 'cross after'], font_size=8)
artist.clear_diagram([1, 2])
artist.clear_diagram([2, 2])

# diag before alpha
artist.add_arrow([0, 0], 0, [0, 2], 'ket', '-2')
artist.add_arrow([0, 0], 1, [2, 0], 'ket', '2\prime')
artist.add_arrow([0, 0], 2, [0, 2], 'ket', '1')
artist.add_arrow([0, 0], 3, [2, 0], 'out')

# diag before beta
artist.add_arrow([0, 1], 0, [0, 2], 'ket', '-2')
artist.add_arrow([0, 1], 1, [0, 2], 'bra', '2\prime')
artist.add_arrow([0, 1], 2, [2, 5], 'ket', '1')
artist.add_arrow([0, 1], 3, [5, 2], 'out')

# diag before gamma
artist.add_arrow([0, 2], 0, [0, 2], 'ket', '-2')
artist.add_arrow([0, 2], 1, [0, 2], 'bra', '2\prime')
artist.add_arrow([0, 2], 2, [2, 0], 'bra', '1')
artist.add_arrow([0, 2], 3, [2, 0], 'out')

# cross before alpha
artist.add_arrow([1, 0], 0, [0, 2], 'ket', '-2')
artist.add_arrow([1, 0], 1, [2, 0], 'ket', '2\prime')
artist.add_arrow([1, 0], 2, [0, 1], 'ket', '1')
artist.add_arrow([1, 0], 3, [1, 0], 'out')

# cross before beta
artist.add_arrow([1, 1], 0, [0, 2], 'ket', '-2')
artist.add_arrow([1, 1], 1, [0, 2], 'bra', '2\prime')
artist.add_arrow([1, 1], 2, [2, 4], 'ket', '1')
artist.add_arrow([1, 1], 3, [2, 1], 'bra')
artist.add_arrow([1, 1], 4, [2, 4], 'ket', '1')
artist.add_arrow([1, 1], 5, [4, 1], 'out')

# diag after alpha
artist.add_arrow([2, 0], 0, [0, 2], 'ket', '-2')
artist.add_arrow([2, 0], 1, [2, 0], 'ket', '2\prime')
artist.add_arrow([2, 0], 4, [0, 2], 'ket', '1')
artist.add_arrow([2, 0], 5, [2, 0], 'out')

# diag after beta
artist.add_arrow([2, 1], 0, [0, 2], 'ket', '-2')
artist.add_arrow([2, 1], 1, [0, 2], 'bra', '2\prime')
artist.add_arrow([2, 1], 2, [2, 1], 'ket')
artist.add_arrow([2, 1], 3, [2, 1], 'bra')
artist.add_arrow([2, 1], 4, [1, 4], 'ket', '1')
artist.add_arrow([2, 1], 5, [4, 1], 'out')

# cross after alpha
artist.add_arrow([3, 0], 0, [0, 2], 'ket', '-2')
artist.add_arrow([3, 0], 1, [2, 0], 'ket', '2\prime')
```python
artist.add_arrow([3, 0], 4, [0, 1], "ket", "1")
artist.add_arrow([3, 0], 5, [1, 0], "out")

# cross after beta
artist.add_arrow([3, 1], 0, [0, 2], "ket", "-2")
artist.add_arrow([3, 1], 1, [0, 2], "bra", "2'")
artist.add_arrow([3, 1], 2, [2, 1], "ket")
artist.add_arrow([3, 1], 3, [2, 1], "bra")
artist.add_arrow([3, 1], 4, [1, 3], "ket", "1")
artist.add_arrow([3, 1], 5, [3, 1], "out")

# cross after gamma
artist.add_arrow([3, 2], 0, [0, 2], "ket", "-2")
artist.add_arrow([3, 2], 1, [0, 2], "bra", "2'")
artist.add_arrow([3, 2], 2, [2, 1], "ket")
artist.add_arrow([3, 2], 3, [2, 1], "bra")
artist.add_arrow([3, 2], 4, [1, 0], "bra", "1")
artist.add_arrow([3, 2], 5, [1, 0], "out")

artist.plot()
plt.show()
```

**Note:** Click [here](#) to download the full example code

### 1.11.22 WMELs: TRIVE on diagonal

Draw WMELs for TRIVE on diagonal.
```python
import matplotlib.pyplot as plt
import WrightTools.diagrams.WMEL as WMEL

artist = WMEL.Artist(size=[6, 3], energies=[0., .5, 1.], state_names=['g', 'a', 'b', '-a+b'])

artist.label_rows([r'$\alpha$', r'$\beta$', r'$\gamma$'])
```

(continues on next page)
artist.label_columns(['I', 'II', 'III', 'IV', 'V', 'VI'])

artist.clear_diagram([1, 2])
artist.clear_diagram([3, 2])

# pw1 alpha
artist.add_arrow([0, 0], 0, [0, 1], "ket", "1")
artist.add_arrow([0, 0], 1, [0, 1], "bra", "-2")
artist.add_arrow([0, 0], 2, [1, 0], "bra", "2'")
artist.add_arrow([0, 0], 3, [1, 0], "out")

# pw1 beta
artist.add_arrow([0, 1], 0, [0, 1], "ket", "1")
artist.add_arrow([0, 1], 1, [0, 1], "bra", "-2")
artist.add_arrow([0, 1], 2, [1, 2], "ket", "2'")
artist.add_arrow([0, 1], 3, [2, 1], "out")

# pw1 gamma
artist.add_arrow([0, 2], 0, [0, 1], "ket", "1")
artist.add_arrow([0, 2], 1, [1, 0], "ket", "-2")
artist.add_arrow([0, 2], 2, [0, 1], "ket", "2'")
artist.add_arrow([0, 2], 3, [1, 0], "out")

# pw2 alpha
artist.add_arrow([1, 0], 0, [0, 1], "ket", "1")
artist.add_arrow([1, 0], 1, [1, 2], "ket", "2'")
artist.add_arrow([1, 0], 2, [2, 1], "ket", "-2")
artist.add_arrow([1, 0], 3, [1, 0], "out")

# pw2 beta
artist.add_arrow([1, 1], 0, [0, 1], "ket", "1")
artist.add_arrow([1, 1], 1, [1, 2], "ket", "2'")
artist.add_arrow([1, 1], 2, [0, 1], "bra", "-2")
artist.add_arrow([1, 1], 3, [2, 1], "out")

# pw3 alpha
artist.add_arrow([2, 0], 0, [0, 1], "bra", "-2")
artist.add_arrow([2, 0], 1, [0, 1], "ket", "1")
artist.add_arrow([2, 0], 2, [1, 0], "bra", "2'")
artist.add_arrow([2, 0], 3, [1, 0], "out")

# pw3 beta
artist.add_arrow([2, 1], 0, [0, 1], "bra", "-2")
artist.add_arrow([2, 1], 1, [0, 1], "ket", "1")
artist.add_arrow([2, 1], 2, [1, 2], "ket", "2'")
artist.add_arrow([2, 1], 3, [2, 1], "out")

# pw3 gamma
artist.add_arrow([2, 2], 0, [0, 1], "bra", "-2")
artist.add_arrow([2, 2], 1, [1, 0], "bra", "1")
artist.add_arrow([2, 2], 2, [0, 1], "ket", "2'")
artist.add_arrow([2, 2], 3, [1, 0], "out")

# pw4 alpha
artist.add_arrow([3, 0], 0, [0, 1], "ket", "2'")
artist.add_arrow([3, 0], 1, [1, 2], "ket", "1")
artist.add_arrow([3, 0], 2, [2, 1], "ket", "-2")

(continues on next page)
artist.add_arrow([3, 0], 3, [1, 0], "out")

# pw4 beta
artist.add_arrow([3, 1], 0, [0, 1], "ket", "2'")
artist.add_arrow([3, 1], 1, [1, 2], "ket", "1")
artist.add_arrow([3, 1], 2, [0, 1], "bra", "-2")
artist.add_arrow([3, 1], 3, [2, 1], "out")

# pw5 alpha
artist.add_arrow([4, 0], 0, [0, 1], "bra", "-2")
artist.add_arrow([4, 0], 1, [1, 0], "bra", "2'")
artist.add_arrow([4, 0], 2, [0, 1], "ket", "1")
artist.add_arrow([4, 0], 3, [1, 0], "out")

# pw5 beta
artist.add_arrow([4, 1], 0, [0, 1], "bra", "2")
artist.add_arrow([4, 1], 1, [0, 1], "ket", "2'")
artist.add_arrow([4, 1], 2, [1, 2], "ket", "1")
artist.add_arrow([4, 1], 3, [2, 1], "out")

# pw5 gamma
artist.add_arrow([4, 2], 0, [0, 1], "bra", "-2")
artist.add_arrow([4, 2], 1, [0, 1], "ket", "2'")
artist.add_arrow([4, 2], 2, [1, 0], "bra", "1")
artist.add_arrow([4, 2], 3, [1, 0], "out")

# pw6 alpha
artist.add_arrow([5, 0], 0, [0, 1], "ket", "2'")
artist.add_arrow([5, 0], 1, [1, 0], "ket", "-2")
artist.add_arrow([5, 0], 2, [0, 1], "ket", "1")
artist.add_arrow([5, 0], 3, [1, 0], "out")

# pw6 beta
artist.add_arrow([5, 1], 0, [0, 1], "ket", "2'")
artist.add_arrow([5, 1], 1, [0, 1], "bra", "-2")
artist.add_arrow([5, 1], 2, [1, 2], "ket", "1")
artist.add_arrow([5, 1], 3, [2, 1], "out")

# pw6 beta
artist.add_arrow([5, 2], 0, [0, 1], "ket", "2'")
artist.add_arrow([5, 2], 1, [0, 1], "bra", "-2")
artist.add_arrow([5, 2], 2, [1, 0], "bra", "1")
artist.add_arrow([5, 2], 3, [1, 0], "out")

artist.plot()
plt.show()
1.12 Citing WrightTools

When publishing research which used WrightTools, please provide credit to WrightTools developers through citation or acknowledgement.

BibTex citation:

```bibtex
@article{Thompson2019,  
doi = {10.21105/joss.01141},  
url = {https://doi.org/10.21105/joss.01141},  
year = {2019},  
month = {jan},  
publisher = {The Open Journal},  
volume = {4},  
number = {33},  
pages = {1141},  
author = {Blaise Thompson and Kyle Sunden and Darien Morrow and Daniel Kohler and John Wright},  
title = {{WrightTools}: a Python package for multidimensional spectroscopy},  
journal = {Journal of Open Source Software}
}
```

Also see `WrightTools.__citation__`.

In addition to the main WrightTools citation, each released version of WrightTools has its own DOI through zenodo. Refer there to cite a particular version: https://zenodo.org/record/1198904.

1.12.1 Publications

The following publications were enabled, to some extent, by WrightTools. The authors of these publications have volunteered to appear on this page.

Would you like your publication to appear here? Email a developer—or better yet, make a pull request.

Ordered by date of publication, newest first.


3. Interference and phase mismatch effects in coherent triple sum frequency spectroscopy *Chemical Physics* 2018 512, 13–19 doi:10.1016/j.chemphys.2018.05.023


1.13 Alternatives

There are several packages with similar goals as WrightTools. None of them replace everything WrightTools does, but each of them overlaps with one of WrightTools’ main features:

- focus on spectroscopy
- multidimensional
- self-describing data formats
- openly licensed & freely available

Some of these packages are focused on adjacent analytical techniques that have different conventions than multidimensional spectroscopy. Others are focused on spectroscopy, but with a different approach than WrightTools. Others are more generic, and don’t have the conventions of any particular experimental strategy built in. All of them are really cool! Your project may be better served by one of them:

- glue
- gridded
- Gwyddion
- hyperspy
- nmrglue
- PyTrA
- scikit-spectra
- specutils
- xarray

Of course there are also the “default” python data-science structures:

- numpy ndarray
- pandas DataFrame

Those with general interest in array-oriented scientific data should be aware of hdf5 and netcdf.
• genindex
• modindex
• search
PYTHON MODULE INDEX

W
WrightTools.artists, 36
WrightTools.collection, 51
WrightTools.data, 57
WrightTools.diagrams, 101
WrightTools.diagrams.delay, 104
WrightTools.diagrams.WMEL, 101
WrightTools.exceptions, 104
WrightTools.kit, 106
WrightTools.units, 120
INDEX

Symbols

__call__() (WrightTools.kit.Spline method), 109
__enter__() (WrightTools.kit.Timer method), 111
__exit__() (WrightTools.kit.Timer method), 111
__init__() (WrightTools.collection.Collection method), 52
__init__() (WrightTools.data.Axis method), 77
__init__() (WrightTools.data.Channel method), 79
__init__() (WrightTools.data.Constant method), 86
__init__() (WrightTools.data.Data method), 58
__init__() (WrightTools.data.Variable method), 89
__init__() (WrightTools_diagrams.WMEL.Artist method), 101
__init__() (WrightTools_diagrams.WMEL.Subplot method), 103
__init__() (WrightTools.kit.INI method), 107
__init__() (WrightTools.kit.Spline method), 109
__init__() (WrightTools.kit.TimeStamp method), 109
__init__() (WrightTools.kit.Timer method), 111

add_arrow() (WrightTools_diagrams.WMEL.Artist method), 102
add_section() (WrightTools.kit.INI method), 107
add_sideplot() (in module WrightTools.artists), 41
add_sideplot() (WrightTools.artists.Axes method), 37
apply_rcparams() (in module WrightTools.artists), 41
argmax() (WrightTools.data.Channel method), 80
argmax() (WrightTools.data.Variable method), 90
argmin() (WrightTools.data.Channel method), 80
argmin() (WrightTools.data.Variable method), 90
attr() (WrightTools.collection.Collection method), 54
attr() (WrightTools.data.Channel property), 84
attr() (WrightTools.data.Data property), 73
attr() (WrightTools.data.Variable property), 93
Axes (class in WrightTools.artists), 37
axes() (WrightTools.data.Data property), 73
Axis (class in WrightTools.data), 77
axis_expressions() (WrightTools.data.Data property), 73
axis_names() (WrightTools.data.Data property), 73

B

bring_to_front() (WrightTools.data.Data method), 59

C

Channel (class in WrightTools.data), 79
channel_names() (WrightTools.data.Data property), 73
channels() (WrightTools.data.Data property), 73
clip() (WrightTools.data.Data method), 81
constant_names() (WrightTools.data.Data property), 86
constant_expressions() (WrightTools.data.Data property), 74

A

add_arrow() (WrightTools_diagrams.WMEL.Artist method), 102
add_section() (WrightTools.kit.INI method), 107
add_sideplot() (in module WrightTools.artists), 41
add_sideplot() (WrightTools.artists.Axes method), 37
argmax() (WrightTools.data.Channel method), 80
argmax() (WrightTools.data.Variable method), 90
argmin() (WrightTools.data.Channel method), 80
argmin() (WrightTools.data.Variable method), 90
artist() (WrightTools_diagrams.WMEL.Artist), 101
attr() (WrightTools.collection.Collection method), 54
attr() (WrightTools.data.Channel property), 84
attr() (WrightTools.data.Data property), 73
attr() (WrightTools.data.Variable property), 93
Axes (class in WrightTools.artists), 37
constant_units() (WrightTools.data.Data property), 74
constants() (WrightTools.data.Data property), 74
contour() (WrightTools.artists.Axes method), 37
contourf() (WrightTools.artists.Axes method), 38
convert() (WrightTools.data.Axis method), 77
convert() (WrightTools.data.Channel method), 81
convert() (WrightTools.data.Constant method), 87
convert() (WrightTools.data.Data method), 61
convert() (WrightTools.data.Variable method), 91
copier() (in module WrightTools.units), 120
copy() (WrightTools.data.Collection method), 52
copy() (WrightTools.data.Data method), 61
corner_text() (in module WrightTools.artists), 41
create_channel() (WrightTools.data.Data method), 62
create_collection() (WrightTools.data.Collection method), 53
create_constant() (WrightTools.data.Data method), 62
create_data() (WrightTools.data.Collection method), 53
create_dataset() (WrightTools.data.Data method), 63
create_figure() (in module WrightTools.artists), 42
create_group() (WrightTools.data.Data method), 63
create_variable() (WrightTools.data.Data method), 64
created() (WrightTools.data.Collection property), 55
created() (WrightTools.data.Data property), 74

data (class in WrightTools.data), 58
datasets() (WrightTools.data.Data property), 74
date (WrightTools.kit.TimeStamp attribute), 110
date() (WrightTools.kit.TimeStamp property), 110
diagonal_line() (in module WrightTools.artists), 43
dictionary() (WrightTools.kit.INI property), 107
diff() (in module WrightTools.kit), 111
dimensionality_error(), 105
discover_dimensions() (in module WrightTools.kit), 112
dtype() (WrightTools.data.Channel property), 84
dtype() (WrightTools.data.Variable property), 93
end (WrightTools.kit.Timer attribute), 110
enforce_mask_shape() (in module WrightTools.kit), 112
EntireDatasetInMemoryWarning, 105

F
fft() (in module WrightTools.kit), 112
Figure (class in WrightTools.artists), 40
data (WrightTools.collection.Collection method), 55
data() (WrightTools.data.Channel property), 84
data() (WrightTools.data.Data property), 74
data() (WrightTools.data.Variable property), 93
FileExistsError, 105
fillvalue() (WrightTools.data.Channel property), 84
fillvalue() (WrightTools.data.Variable property), 93
flatten_list() (in module WrightTools.kit), 113
fluence() (in module WrightTools.kit), 113
flush() (WrightTools.data.Channel attribute), 84
flush() (WrightTools.data.Variable attribute), 93
flush() (WrightTools.data.Collection method), 53
flush() (WrightTools.data.Data method), 64
from_Aramis() (in module WrightTools.data), 98
from_BrunoRaman() (in module WrightTools.data), 96
from_Cary() (in module WrightTools.collection), 55
from_COLORS() (in module WrightTools.data), 96
directory() (in module WrightTools.collection), 57
from_JASCO() (in module WrightTools.data), 97
from_KENT() (in module WrightTools.data), 97
from_ocean_optics() (in module WrightTools.data), 98
from_PyCMDS() (in module WrightTools.data), 98
from_shimadzu() (in module WrightTools.data), 99
from_Solis() (in module WrightTools.data), 99
dspcm() (in module WrightTools.data), 100
dspcm() (in module WrightTools.data), 100
full() (WrightTools.data.Axis property), 78
full() (WrightTools.data.Channel property), 84
full() (WrightTools.data.Constant property), 88
full() (WrightTools.data.Variable property), 93
fullpath() (WrightTools.data.Collection property), 55
fullpath() (WrightTools.data.Channel property), 84
fullpath() (WrightTools.data.Data property), 74
fullpath() (WrightTools.data.Variable property), 94

G
get() (WrightTools.data.Data method), 64
get_color_cycle() (in module WrightTools.artists), 44
get_nadir() (WrightTools.data.Data method), 65
get_options() (WrightTools.kit.INI method), 107
get_path_matching() (in module WrightTools.kit), 114
get_scaled_bounds() (in module WrightTools.artists), 44
global_handler() (in module WrightTools.kit), 114
gradient() (in module WrightTools.kit), 65
grayify_cmap() (in module WrightTools.artists), 44
GridSpec (class in WrightTools.artists), 40
has_option() (WrightTools.kit.INI method), 108
has_section() (WrightTools.kit.INI method), 108
heal() (WrightTools.data.Data method), 65
hms (WrightTools.kit.TimeStamp attribute), 110
hms() (WrightTools.kit.TimeStamp property), 110
human() (WrightTools.kit.TimeStamp attribute), 110
human() (WrightTools.kit.TimeStamp property), 110
id() (WrightTools.data.Data property), 75
identity() (WrightTools.data.Data property), 78
identity() (WrightTools.data.Constant property), 88
INI (class in WrightTools.kit), 107
interact2D() (in module WrightTools.artists), 44
intersperse() (in module WrightTools.artists), 44
interval (WrightTools.kit.Timer attribute), 111
is_valid_conversion() (in module WrightTools.units), 121
item_names() (WrightTools.collection.Collection property), 55
item_names() (WrightTools.data.Data property), 75
join() (in module WrightTools.data), 95
joint_shape() (in module WrightTools.kit), 115
kind() (in module WrightTools.units), 121
kind() (WrightTools.data.Data property), 75
label() (WrightTools.data.Axis property), 78
label() (WrightTools.data.Constant property), 88
label() (WrightTools.data.Variable property), 94
label_columns() (WrightTools.canvas.WMEL.Artist method), 102
label_rows() (WrightTools.canvas.WMEL.Artist method), 102
label_sectors() (in module WrightTools.canvas.WMEL.Artist method), 102
leastsqfitter() (in module WrightTools.kit), 115
legend() (WrightTools.canvas.Axes method), 38
level() (WrightTools.data.Data method), 66
log() (WrightTools.data.Channel method), 81
log() (WrightTools.data.Variable method), 91
log10() (WrightTools.data.Channel method), 81
log10() (WrightTools.data.Variable method), 91
log2() (WrightTools.data.Channel method), 81
log2() (WrightTools.data.Variable method), 91
M
mag() (WrightTools.data.Channel method), 82
major_extent() (WrightTools.data.Channel property), 85
map_variable() (WrightTools.data.Data method), 66
mask_reduce() (in module WrightTools.kit), 115
masked() (WrightTools.data.Axis property), 78
masked() (WrightTools.data.Constant property), 88
max() (WrightTools.data.Axis method), 77
max() (WrightTools.data.Channel method), 82
max() (WrightTools.data.Constant method), 87
min() (WrightTools.data.Axis method), 77
min() (WrightTools.data.Channel method), 82
min() (WrightTools.data.Constant method), 87
min() (WrightTools.data.Variable method), 92
minor_extent() (WrightTools.data.Channel property), 85
moment() (WrightTools.data.Data method), 67
mono_resolution() (in module WrightTools.kit), 116
MultidimensionalAxisError, 105
N
name() (WrightTools.collection.Collection property), 55
name() (WrightTools.data.Channel property), 85
name() (WrightTools.data.Data property), 75
name() (WrightTools.data.Variable property), 94
NameNotUniqueError, 105
natural_name() (WrightTools.collection.Collection property), 55
natural_name() (WrightTools.data.Axis property), 78
natural_name() (WrightTools.data.Channel property), 85
natural_name() (WrightTools.data.Constant property), 88
natural_name() (WrightTools.data.Data property), 75
natural_name() (WrightTools.data.Variable property), 94

Index

173
plot_colormap_components() (WrightTools.data.Axis property), 79
plot_colorbar() (WrightTools.data.Axis property), 85
plot() (WrightTools.data.Channel property), 75
plot() (WrightTools.data.Channel property), 88
plot() (WrightTools.data.Variable property), 75
plot() (WrightTools.data.Variable property), 94
plot_gridlines() (WrightTools.data.Channel property), 79
plot_gridlines() (WrightTools.data.Channel property), 85
normalize() (WrightTools.data.Channel method), 82
null() (WrightTools.data.Channel property), 85
nde() (WrightTools.data.Channel property), 45
nde() (WrightTools.data.Channel property), 85
nde() (WrightTools.data.Variable property), 75
nde() (WrightTools.data.Variable property), 88
nde() (WrightTools.data.Variable property), 88
nde() (WrightTools.data.Variable property), 94
nde() (WrightTools.data.Variable property), 94
nde() (WrightTools.data.Variable property), 116
normalize() (WrightTools.data_CHANNEL method), 82
null() (WrightTools.data_Channel property), 85
O
ObjectExistsWarning, 105
offset() (WrightTools.data.Data method), 67
open() (WrightTools.data.Data), 120
orthogonal() (in module WrightTools.kit), 116
overline_colors (in module WrightTools.artists), 45
pairwise() (in module WrightTools.kit), 116
parent() (WrightTools.collection.Collection property), 55
parent() (WrightTools.data.Channel property), 85
parent() (WrightTools.data.Channel property), 75
parent() (WrightTools.data.Variable property), 94
path() (WrightTools.data.Data property), 94
path() (WrightTools.data.TimeStamp property), 110
path() (WrightTools.data.TimeStamp property), 110
pcolor() (WrightTools.artists.Axes method), 39
pcolor_helper() (in module WrightTools.artists), 45
pcolormesh() (WrightTools.artists.Axes method), 39
plot() (WrightTools.artists.Axes method), 39
plot() (WrightTools.artists.Axes method), 40
plot() (WrightTools.diagrams.WMEL.Artist method), 102
plot_colorbar() (in module WrightTools.artists), 45
plot_colormap_components() (in module WrightTools.artists), 46
plot_gridlines() (in module WrightTools.artists), 46
plot_gridlines() (in module WrightTools.artists), 47
points() (WrightTools.data.Axes property), 79
points() (in module WrightTools.data.Channel property), 79
points() (in module WrightTools.data.Channel property), 85
points() (in module WrightTools.data.Channel property), 88
points() (WrightTools.data.Channel property), 88
points() (WrightTools.data.Channel property), 94
points() (WrightTools.data.Variable property), 94
points() (WrightTools.data.Variable property), 75
points() (WrightTools.data.Variable property), 85
points() (WrightTools.data.Variable property), 88
points() (WrightTools.data.Variable property), 94
print_tree() (WrightTools.collection.Collection method), 53
print_tree() (WrightTools.data.Data method), 68
prune() (WrightTools.data.Data method), 68
plot() (WrightTools.data.Data method), 68
Q
quick1D() (in module WrightTools.artists), 47
quick2D() (in module WrightTools.artists), 48
read() (WrightTools.data.Data property), 75
region() (WrightTools.data.Data property), 76
regionref() (WrightTools.data.Data property), 76
remove_channel() (WrightTools.data.Data method), 68
remove_constant() (WrightTools.data.Data method), 69
remove_nans_1D() (in module WrightTools.kit), 117
remove_variable() (WrightTools.data.Data method), 69
rename_channels() (WrightTools.data.Data method), 69
rename_variables() (WrightTools.data.Data method), 69
RFC3339 (WrightTools.data.Data property), 75
RFC3339() (WrightTools.data.Data property), 109
RFC5322 (WrightTools.data.TimeStamp attribute), 110
RFC5322() (WrightTools.data.TimeStamp property), 109
S
save() (WrightTools.collection.Collection method), 54
save() (WrightTools.data.Data method), 70
savefig() (in module WrightTools.artists), 49
sections() (WrightTools.kit.INI property), 108
set_ax_labels() (in module WrightTools.artists), 49
set_ax_spines() (in module WrightTools.artists), 50
set_constants() (WrightTools.data.Data method), 70
set_fig_labels() (in module WrightTools.artists), 50
shape() (WrightTools.data.Axes property), 79
shape() (WrightTools.data.Channel property), 86
shape() (WrightTools.data.Channel property), 88
shape() (WrightTools.data.Channel property), 76
shape() (WrightTools.data.Variable property), 94
shape() (WrightTools.data.Channel property), 117
shape() (WrightTools.data.Channel property), 117
shape() (WrightTools.data.Channel property), 70
shape() (WrightTools.data.Channel property), 86
shape() (WrightTools.data.Channel property), 86
shape() (WrightTools.data.Channel property), 89
shape() (WrightTools.data.Channel property), 89
shape() (WrightTools.data.Channel property), 86
shape() (WrightTools.data.Channel property), 76
shape() (WrightTools.data.Channel property), 95
slices() (WrightTools.data.Channel method), 82
slices() (WrightTools.data.Channel method), 92
smooth() (WrightTools.data.Data method), 70
smooth_1D() (in module WrightTools.kit), 117
source() (WrightTools.data.Data property), 76
Spline (class in WrightTools.kit), 109
split() (WrightTools.data.Data method), 71
start (WrightTools.kit.Timer attribute), 110
std() (WrightTools.data.Constant property), 89
stitch_to_animation() (in module WrightTools.artists), 51
string2identifier() (in module WrightTools.kit), 117
Subplot (class in WrightTools.diagrams.WMEL), 103
subplots_adjust() (in module WrightTools.artists), 51
svd() (in module WrightTools.kit), 118
symmetric_root() (WrightTools.data.Channel method), 82
symmetric_root() (WrightTools.data.Variable method), 92
symmetric_sqrt() (in module WrightTools.kit), 118
T
Timer (class in WrightTools.kit), 110
TimeStamp (class in WrightTools.kit), 109
timestamp_from_RFC3339() (in module WrightTools.kit), 118
transform() (WrightTools.data.Data method), 71
trim() (WrightTools.data.Channel method), 82
TypeError, 105
U
unique() (in module WrightTools.kit), 119
units() (WrightTools.data.Channel property), 86
units() (WrightTools.data.Data property), 76
units() (WrightTools.data.Variable property), 95
units_kind() (WrightTools.data.Axis property), 79
units_kind() (WrightTools.data.Constant property), 89
UnitsError, 106
unix (WrightTools.kit.TimeStamp attribute), 109
update() (WrightTools.data.Data method), 72
V
valid_index() (in module WrightTools.kit), 119
value() (WrightTools.data.Channel property), 86
value() (WrightTools.data.Constant property), 89
ValueError, 106
Variable (class in WrightTools.data), 89
variable_names() (WrightTools.data.Data property), 76
variables() (WrightTools.data.Axis property), 79
variables() (WrightTools.data.Constant property), 89
variables() (WrightTools.data.Data property), 76
VisibleDeprecationWarning, 106
W
WrightTools.artists (module), 36
WrightTools.collection (module), 51
WrightTools.data (module), 57