## CONTENTS

1 Contents

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 ...................................................... 99
1.12 Citing WrightTools ....................................... 146
1.13 Alternatives .............................................. 147

2 Index ......................................................... 149

Python Module Index ........................................ 151

Index ......................................................... 153
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).

<table>
<thead>
<tr>
<th>PyPI</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td></td>
</tr>
</tbody>
</table>

| conda-forge |     |
| version     |     |
| status      |     |

| Read the Docs |      |
| stable        |      |
| latest        |      |

<table>
<thead>
<tr>
<th>Published</th>
<th>JOSS</th>
</tr>
</thead>
</table>
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/811qwfvb.wt5)
| axes
|  0: w2 (nm) (41, 1, 1)
|  1: w1=wm (nm) (1, 41, 1)
|  2: d2 (fs) (1, 1, 23)
| constants
|  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)
| 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 $w_1=\lambda_m$. 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.

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

_001_dat

$\lambda_2 = 621\text{nm}, \tau_2 = -120\text{fs}$
**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—$w = w_m$ and $d_2$, 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']})
```

---

![Graph showing the plot of two-dimensional data with labels $\lambda_2 = 621$ nm and $T_2$ (fs).]
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
>>> data.units
('nm', 'nm', 'fs')
>>> data.convert('eV')
axis w2 converted from nm to eV
axis wl=wm converted from nm to eV
>>> 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
>>> 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.

```python
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...

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

and then open...

```python
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
\[ \lambda_2 = 621 \text{ nm} \]
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:

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

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

```python
>>> from WrightTools import datasets
cols recognized as v0 (19)
data created at /tmp/wlizsmt.wt5::/d1_d2_diagonal_dat
axes: ('d1', 'd2')
shape: (21, 21)
```

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:

```python
>>> collection.print_tree()
test (/tmp/wlizsmt.wt5)
0: d1_d2_diagonal_dat (21, 21)
   axes: d1 (fs), d2 (fs)
   constants:
   channels: ai0, ai1, ai2, ai3
1: tsunami (2048,)
   axes: energy (nm)
   constants:
   channels: signal
2: 3d1580hi (35, 11, 11)
   axes: wm (wn), w2 (wn), w1 (wn)
   constants:
   channels: signal_diff, signal_mean, pyro1, pyro2, pyro3, PMT voltage
```

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/xxxxxx/download?fname=file.csv.gz. Google Drive direct download
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):
    def single(arr, intensity=intensity, FWHM=FWHM, x0=x0):
        return intensity*(0.5*FWHM)**2/((xi-x0)**2+(0.5*FWHM)**2)
    return single(xi) * single(yi)
x0 = np.linspace(6000, 8000, 75)
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>max()</td>
<td>variable maximum</td>
</tr>
<tr>
<td>min()</td>
<td>variable minimum</td>
</tr>
<tr>
<td>natural_name</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>label()</td>
<td>LaTeX-formatted label, appropriate for plotting</td>
</tr>
<tr>
<td>min()</td>
<td>coordinates minimum, in current units</td>
</tr>
<tr>
<td>max()</td>
<td>coordinates maximum, in current units</td>
</tr>
<tr>
<td>natural_name</td>
<td>axis name</td>
</tr>
<tr>
<td>units</td>
<td>current axis units (change with <code>convert()</code>)</td>
</tr>
<tr>
<td>variables</td>
<td>component variables</td>
</tr>
<tr>
<td>expression</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`. The label uses the python `format` function, 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 `round()` function. 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 `$\text{\LaTeX} \lambda_1 = 123.460 \, \text{nm}$`, 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>
<tbody>
<tr>
<td>format_spec</td>
<td>Format specification for how to represent the value, as in format().</td>
</tr>
<tr>
<td>round_spec</td>
<td>Specify which digit to round to, as in round()</td>
</tr>
<tr>
<td>label</td>
<td>LaTeX formatted label which includes a symbol and the constant value.</td>
</tr>
<tr>
<td>value</td>
<td>The mean (ignoring NaNs) of the evaluated expression.</td>
</tr>
<tr>
<td>std</td>
<td>The standard deviation of the points used to compute the value.</td>
</tr>
</tbody>
</table>

**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>heal()</td>
<td>use interpolation to guess the value of NaNs within a channel</td>
<td>Heal</td>
</tr>
<tr>
<td>join()</td>
<td>join together multiple data objects, accounting for dimensionality and over-</td>
<td>Join</td>
</tr>
<tr>
<td></td>
<td>lap</td>
<td></td>
</tr>
<tr>
<td>map_variable()</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.

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.

calibration.parent
<WrightTools.Collection 'results'>

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

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 particularly 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.vl0p0_perovskite_TA  # axes w1=w2, 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

\[ \hbar \omega_1 = 1.65 \text{ eV}, \, \hbar \omega_2 = 1.7 \text{ eV} \]

```
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 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
perovskite_TA

$\hbar \omega_2 = 1.7 \text{ eV}$
perovskite_TA
\[ \tau_{21} = 0.821 \text{fs} \]
1.5.2 Interactive artists

\texttt{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)

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 x and y projections of the slice (shaded gray). For \textit{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. \texttt{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 choice colormaps built-in.

All of these are held in the \texttt{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.\textsuperscript{1} The cubehelix parameters have been fine-tuned to roughly mimic the colors of the historically popular “jet” colormap.

\textsuperscript{1} A colour scheme for the display of astronomical intensity images, Dave Green, Bulletin of the Astronomical Society of India 2011, arXiv:1108.5083
WrightTools Documentation, Release 3.3.1

perovskite_TA

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

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

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
    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

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
idxs = [(row, col) for row in range(1, 3) for col in range(2)]
for indx, wigner, color in zip(idxs, wigners, wigner_colors):
    ax = plt.subplot(gs[indx])
    ...
idxs = [(0, col) for col in range(2)]
for indx, color, traces in zip(idxs, 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(idxs, 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(idxs, 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

1.5. Artists
• `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[0].min(), traces[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>mm</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.CuPCt5_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.wl_000</td>
<td>('wl',)</td>
<td>(51,)</td>
<td></td>
</tr>
<tr>
<td>PyCMDS.wl_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 Fields</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.CuPCt5_powder_ATR</td>
<td>('energy',)</td>
<td>(7259,)</td>
<td></td>
</tr>
<tr>
<td>wt5.v1p0p0_perovskite_TA</td>
<td>('wi=wm', 'w2', 'd2')</td>
<td>(52, 52, 13)</td>
<td>Quick 2D Signed</td>
</tr>
<tr>
<td>wt5.v1p0p0_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 \texttt{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

---

Block, Andrei V. Pakoulev, Rachel S. Selinsky, Song Jin, and John Wright The Journal of Physical Chemistry C 2011 115 (46), 22833-22844 doi:10.1021/jp207273x


---

1.8. Contributing
$ 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.

```bash
$ 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:

```bash
$ 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).
<table>
<thead>
<tr>
<th>name</th>
<th>Collection</th>
<th>Data</th>
<th>Variable</th>
<th>Channel</th>
<th>description/notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Usually matches the last component of the path, except for root, /, which does not have a path with it’s name</td>
</tr>
<tr>
<td>class</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>Identifies which kind of WrightTools object it is.</td>
</tr>
<tr>
<td>created</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Timestamp of when the object was made, can be overwritten with source file creation time by from_&lt;x&gt; functions.</td>
</tr>
<tr>
<td><strong>version</strong></td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>wt5 version identifier</td>
</tr>
<tr>
<td>item_names</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Ordered list of the children</td>
</tr>
<tr>
<td>variable_names</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Ordered list of all Variables</td>
</tr>
<tr>
<td>channel_names</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Ordered list of all Channels</td>
</tr>
<tr>
<td>axes</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>Ordered list of axes expressions which define how a Data object is represented</td>
</tr>
<tr>
<td>constants</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>Ordered list of expressions for values which are constant</td>
</tr>
<tr>
<td>kind</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>Short description of what type of file it originated from, usually the instrument</td>
</tr>
<tr>
<td>source</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>File path/url to the original file as read in</td>
</tr>
<tr>
<td>label</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Identifier used to create more complex labels in Axes or Constants, which are used to plot</td>
</tr>
<tr>
<td>units</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Units assigned to the dataset</td>
</tr>
<tr>
<td>min</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Cached minimum value</td>
</tr>
<tr>
<td>max</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Cached maximum value</td>
</tr>
<tr>
<td>argmin</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Cached index of minimum value</td>
</tr>
<tr>
<td>argmax</td>
<td>yes</td>
<td>yes</td>
<td></td>
<td></td>
<td>Cached index of maximum value</td>
</tr>
<tr>
<td>signed</td>
<td>yes</td>
<td></td>
<td></td>
<td></td>
<td>Boolean for treating channel as signed/unsigned</td>
</tr>
</tbody>
</table>

### 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 computers 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 reccomend 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.

```python
Axes(fig, rect[, facecolor, frameon, ...])  # Axes.
Figure([figsize, dpi, facecolor, edgecolor, ...])  # Figure.
GridSpec(nrows, ncols[, figure, left, ...])  # GridSpec.
add_sideplot(ax, along[, pad, grid, ...])  # Add a sideplot to an axis.
apply_rcparams(kind)  # Quickly apply rcparams for given purposes.
```

```python
colormaps
corner_text(text[, distance, ax, corner, ...])  # Place some text in the corner of the figure.
create_figure(*[, width, nrows, cols, ...])  # Re-parameterization of matplotlib figure creation tools, exposing convenient variables.
diagonal_line([xi, yi, ax, c, ls, lw, zorder])  # Plot a diagonal line.
get_color_cycle(n[, cmap, rotations])  # Get a list of RGBA colors following a colormap.
get_scaled_bounds(ax, position, *[...])  # Get scaled bounds.
grayscale_cmap(cmap)  # Return a grayscale version of the colormap.
interact2D(data[, xaxis, yaxis, channel, ...])  # Interactive 2D plot of the dataset.
```

```python
overline_colors  # list() -> new empty list list(iterable) -> new list initialized from iterable’s items
pcolor_helper(xi, yi[, zi])  # Prepare a set of arrays for plotting using `pcolor`.
plot_colorbar([clax, cmap, ticks, clim, ...])  # Easily add a colormap to an axis.
plot_colormap_components(cmap)  # Plot the components of a given colormap.
plot_gridlines([ax, c, lw, diagonal, ...])  # Plot dotted gridlines onto an axis.
plot_margins(*[, fig, inches, centers, edges])  # Add lines onto a figure indicating the margins, centers, and edges.
quick1D(data[, axis, at, channel, local, ...])  # Quickly plot 1D slice(s) of data.
quick2D(data[, xaxis, yaxis, at, channel, ...])  # Quickly plot 2D slice(s) of data.
savefig(path[, fig, close])  # Save a figure.
set_ax_labels([ax, xlabel, ylabel, xticks, ...])  # Set all axis labels properties easily.
set_ax_spines([ax, c, lw, zorder])  # Easily set the properties of all four axis spines.
set_fig_labels([fig, xlabel, ylabel, ...])  # Set all axis labels of a figure simultaneously.
```

continues 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, box_aspect=None, **kwargs)
```

Build an axes in a figure.

**Parameters**

- `fig` (~matplotlib.figure.Figure) – The axes is build in the .Figure fig.
- `rect` ([left, bottom, width, height]) – The axes is build in the rectangle rect. rect is in .Figure coordinates.
- `sharex` (~axes.Axes, optional) – The x or y .matplotlib.axis is shared with the x or y axis in the input .axes.Axes.
- `sharey` (~axes.Axes, optional) – The x or y .matplotlib.axis is shared with the x or y axis in the input .axes.Axes.
- `frameon` (bool, default: True) – Whether the axes frame is visible.
- `box_aspect` (None, or a number, optional) – Sets the aspect of the axes box. See .axes.Axes.set_box_aspect for details.
- `**kwargs` – Other optional keyword arguments:

**Returns**

The new .axes.Axes object.

**Return type**

- .axes.Axes
Methods

```python
__init__(fig, rect[, facecolor, frameon, ...])  # Build an axes in a figure.
acorr(x, *[, data])  # Plot the autocorrelation of x.
add_artist(a)  # Add an ~Artist to the axes, and return the artist.
add_callback(func)  # Add a callback function that will be called whenever one of the .Artist's properties changes.
add_child_axes(ax)  # Add an ~AxesBase to the axes' children; return the child axes.
add_collection(collection[, autolim])  # Add a ~Collection to the axes' collections; return the collection.
add_container(container)  # Add a ~Container to the axes' containers; return the container.
add_image(image)  # Add an ~AxesImage to the axes' images; return the image.
add_line(line)  # Add a ~Line2D to the axes' lines; return the line.
add_patch(p)  # Add a ~Patch to the axes' patches; return the patch.
add_sideplot(along[, pad, height, ymin, ymax])  # Add a side axis.
add_table(tab)  # Add a ~Table to the axes' tables; return the table.
angle_spectrum(x[, Fs, Fc, window, pad_to, ...])  # Plot the angle spectrum.
annotate(text, xy, *args, **kwargs)  # Annotate the point xy with text text.
apply_aspect([position])  # Adjust the Axes for a specified data aspect ratio.
arrow(x, y, dx, dy, **kwargs)  # Add an arrow to the axes.
autoscale([enable, axis, tight])  # Autoscale the axis view to the data (toggle).
autoscale_view([tight, scalex, scaley])  # Autoscale the view limits using the data limits.
axhline([y, xmin, xmax])  # Add a horizontal line across the axis.
axhspan(ymin, ymax[, xmin, xmax])  # Add a horizontal span (rectangle) across the axis.
axis(*args[, emit])  # Convenience method to get or set some axis properties.
axline(xy1[, xy2, slope])  # Add an infinitely long straight line.
axvline(x, ymin, ymax)  # Add a vertical line across the axes.
axvspan(xmin, xmax[, ymin, ymax])  # Add a vertical span (rectangle) across the axes.
bar(x, height[, width, bottom, align, data])  # Make a bar plot.
barbs(*args[, data])  # Plot a 2D field of barbs.
barh(y, width[, height, left, align])  # Make a horizontal bar plot.
boxplot(x[, notch, sym, vert, whis, ...])  # Make a box and whisker plot.
broken_barh(xranges, yrange[, *[, data]])  # Plot a horizontal sequence of rectangles.
bxp(bxpstats[, positions, widths, vert, ...])  # Drawing function for box and whisker plots.
can_pan()  # Return True if this axes supports any pan/zoom button functionality.
can_zoom()  # Return True if this axes supports the zoom box button functionality.
cla()  # Clear the current axes.
clabel(CS[, levels])  # Label a contour plot.
clear()  # Clear the axes.
cohere(x, y[, NFFT, Fs, detrend, ...])  # Plot the coherence between x and y.
contains(mouseevent)  # Test whether the artist contains the mouse event.
contains_point(point)  # Return whether point (pair of pixel coordinates) is inside the axes patch.
contour(*args, **kwargs)  # Plot contours.
```

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>contourf(*args, **kwargs)</code></td>
<td>Plot contours.</td>
</tr>
<tr>
<td><code>convert_xunits(x)</code></td>
<td>Convert x using the unit type of the xaxis.</td>
</tr>
<tr>
<td><code>convert_yunits(y)</code></td>
<td>Convert y using the unit type of the yaxis.</td>
</tr>
<tr>
<td><code>csd(x, y[, NFFT, Fs, Fc, detrend, window, ...])</code></td>
<td>Plot the cross-spectral density.</td>
</tr>
<tr>
<td><code>drag_pan(button, key, x, y)</code></td>
<td>Called when the mouse moves during a pan operation.</td>
</tr>
<tr>
<td><code>draw([renderer, inframe])</code></td>
<td>Draw the Artist (and its children) using the given renderer.</td>
</tr>
<tr>
<td><code>draw_artist(a)</code></td>
<td>Efficiently redraw a single artist.</td>
</tr>
<tr>
<td><code>end_pan()</code></td>
<td>Called when a pan operation completes (when the mouse button is up.)</td>
</tr>
<tr>
<td><code>errorbar(x, y[, yerr, xerr, fmt, ecolor, ...])</code></td>
<td>Plot y versus x as lines and/or markers with attached errorbars.</td>
</tr>
<tr>
<td><code>eventplot(positions[, orientation, ...])</code></td>
<td>Plot identical parallel lines at the given positions.</td>
</tr>
<tr>
<td><code>fill(*args[, data])</code></td>
<td>Plot filled polygons.</td>
</tr>
<tr>
<td><code>fill_between(x, y1[, y2, where, ...])</code></td>
<td>Fill the area between two horizontal curves.</td>
</tr>
<tr>
<td><code>fill_betweenx(y, x1[, x2, where, step, ...])</code></td>
<td>Fill the area between two vertical curves.</td>
</tr>
<tr>
<td><code>findobj([match, include_self])</code></td>
<td>Find artist objects.</td>
</tr>
<tr>
<td><code>format_coord(x, y)</code></td>
<td>Return a format string formatting the x, y coordinates.</td>
</tr>
<tr>
<td><code>format_cursor_data(data)</code></td>
<td>Return a string representation of data.</td>
</tr>
<tr>
<td><code>format_xdata(x)</code></td>
<td>Return x formatted as an x-value.</td>
</tr>
<tr>
<td><code>format_ydata(y)</code></td>
<td>Return y formatted as an y-value.</td>
</tr>
<tr>
<td><code>get_adjustable()</code></td>
<td>Return whether the Axes will adjust its physical dimension ('box') or its data limits ('datalim') to achieve the desired aspect ratio.</td>
</tr>
<tr>
<td><code>get_agg_filter()</code></td>
<td>Return filter function to be used for agg filter.</td>
</tr>
<tr>
<td><code>get_alpha()</code></td>
<td>Return the alpha value used for blending - not supported on all backends.</td>
</tr>
<tr>
<td><code>get_anchor()</code></td>
<td>Get the anchor location.</td>
</tr>
<tr>
<td><code>get_animated()</code></td>
<td>Return whether the artist is animated.</td>
</tr>
<tr>
<td><code>get_aspect()</code></td>
<td>Get the axes box aspect.</td>
</tr>
<tr>
<td><code>get_autoscale_on()</code></td>
<td>Get whether autoscaling is applied for both axes on plot commands.</td>
</tr>
<tr>
<td><code>get_autoscalex_on()</code></td>
<td>Get whether autoscaling for the x-axis is applied on plot commands.</td>
</tr>
<tr>
<td><code>get_autoscaley_on()</code></td>
<td>Get whether autoscaling for the y-axis is applied on plot commands.</td>
</tr>
<tr>
<td><code>get_axes_locator()</code></td>
<td>Return the axes_locator.</td>
</tr>
<tr>
<td><code>get_axisbelow()</code></td>
<td>Get whether axis ticks and gridlines are above or below most artists.</td>
</tr>
<tr>
<td><code>get_box_aspect()</code></td>
<td>Get the axes box aspect.</td>
</tr>
<tr>
<td><code>get_clip_box()</code></td>
<td>Return the clipbox.</td>
</tr>
<tr>
<td><code>get_clip_on()</code></td>
<td>Return whether the artist uses clipping.</td>
</tr>
<tr>
<td><code>get_clip_path()</code></td>
<td>Return the clip path.</td>
</tr>
<tr>
<td><code>get_contain()</code></td>
<td><code>[Deprecated] Return the custom contains function of the artist if set, or None.</code></td>
</tr>
<tr>
<td><code>get_cursor_data(event)</code></td>
<td>Return the cursor data for a given event.</td>
</tr>
<tr>
<td><code>get_data_ratio()</code></td>
<td>Return the aspect ratio of the scaled data.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_data_ratio_log()</td>
<td>[Deprecated] Return the aspect ratio of the raw data in log scale.</td>
</tr>
<tr>
<td>get_default_bbox_extra_artists()</td>
<td>Return a default list of artists that are used for the bounding box calculation.</td>
</tr>
<tr>
<td>get_facecolor()</td>
<td>Get the facecolor of the Axes.</td>
</tr>
<tr>
<td>get_fc()</td>
<td>Alias for get_facecolor.</td>
</tr>
<tr>
<td>get_figure()</td>
<td>Return the .Figure instance the artist belongs to.</td>
</tr>
<tr>
<td>get_frame_on()</td>
<td>Get whether the axes rectangle patch is drawn.</td>
</tr>
<tr>
<td>get_gid()</td>
<td>Return the group id.</td>
</tr>
<tr>
<td>get_images()</td>
<td>Return a list of .AxesImages contained by the Axes.</td>
</tr>
<tr>
<td>get_in_layout()</td>
<td>Return boolean flag, True if artist is included in layout calculations.</td>
</tr>
<tr>
<td>get_label()</td>
<td>Return the label used for this artist in the legend.</td>
</tr>
<tr>
<td>get_legend()</td>
<td>Return the .Legend instance, or None if no legend is defined.</td>
</tr>
<tr>
<td>get_legend_handles_labels([legend_handler_map])</td>
<td>Return handles and labels for legend</td>
</tr>
<tr>
<td>get_lines()</td>
<td>Return a list of lines contained by the Axes.</td>
</tr>
<tr>
<td>get_navigate()</td>
<td>Get whether the axes responds to navigation commands</td>
</tr>
<tr>
<td>get_navigate_mode()</td>
<td>Get the navigation toolbar button status: 'PAN', 'ZOOM', or None</td>
</tr>
<tr>
<td>get_path_effects()</td>
<td></td>
</tr>
<tr>
<td>get_picker()</td>
<td>Return the picking behavior of the artist.</td>
</tr>
<tr>
<td>get_position([original])</td>
<td>Get a copy of the axes rectangle as a .Bbox.</td>
</tr>
<tr>
<td>get_rasterization_zorder()</td>
<td>Return the zorder value below which artists will be rasterized.</td>
</tr>
<tr>
<td>get_rasterized()</td>
<td>Return whether the artist is to be rasterized.</td>
</tr>
<tr>
<td>get_shared_x_axes()</td>
<td>Return a reference to the shared axes Grouper object for x axes.</td>
</tr>
<tr>
<td>get_shared_y_axes()</td>
<td>Return a reference to the shared axes Grouper object for y axes.</td>
</tr>
<tr>
<td>get_skech_params()</td>
<td>Return the sketch parameters for the artist.</td>
</tr>
<tr>
<td>get_snap()</td>
<td>Return the snap setting.</td>
</tr>
<tr>
<td>get_tightbbox(renderer[, call_axes_locator, ...])</td>
<td>Return the tight bounding box of the axes, including axis and their decorators (xlabel, title, etc).</td>
</tr>
<tr>
<td>get_title([loc])</td>
<td>Get an axes title.</td>
</tr>
<tr>
<td>get_transform()</td>
<td>Return the .Transform instance used by this artist.</td>
</tr>
<tr>
<td>get_transformed_clip_path_and_affine()</td>
<td>Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.</td>
</tr>
<tr>
<td>get_url()</td>
<td>Return the url.</td>
</tr>
<tr>
<td>get_visible()</td>
<td>Return the visibility.</td>
</tr>
<tr>
<td>get_window_extent(*args, **kwargs)</td>
<td>Return the axes bounding box in display space; args and kwargs are empty.</td>
</tr>
<tr>
<td>get_xaxis()</td>
<td>Return the XAxis instance.</td>
</tr>
</tbody>
</table>
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_xaxis_text1_transform(pad_points)</code></td>
<td>returns</td>
</tr>
<tr>
<td><code>get_xaxis_text2_transform(pad_points)</code></td>
<td>returns</td>
</tr>
<tr>
<td><code>get_xaxis_text1_transform(pad_points)</code></td>
<td>returns</td>
</tr>
<tr>
<td><code>get_xaxis_text2_transform(pad_points)</code></td>
<td>returns</td>
</tr>
</tbody>
</table>

1.10. WrightTools API
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>get_yscale()</code></td>
<td>Return the yaxis’ scale (as a str).</td>
</tr>
<tr>
<td><code>get_yticklabels([minor, which])</code></td>
<td>Get the yaxis’ tick labels.</td>
</tr>
<tr>
<td><code>get_yticklines([minor])</code></td>
<td>Return the yaxis’ tick lines as a list of <code>.Line2D</code>s.</td>
</tr>
<tr>
<td><code>get_yticks(*[, minor])</code></td>
<td>Return the yaxis’ tick locations in data coordinates.</td>
</tr>
<tr>
<td><code>get_zorder()</code></td>
<td>Return the artist’s zorder.</td>
</tr>
<tr>
<td><code>grid([b, which, axis])</code></td>
<td>Configure the grid lines.</td>
</tr>
<tr>
<td><code>has_data()</code></td>
<td>Return <code>True</code> if any artists have been added to axes.</td>
</tr>
<tr>
<td><code>have_units()</code></td>
<td>Return <code>True</code> if units are set on any axis.</td>
</tr>
<tr>
<td><code>hexbin(x, y[, C, gridsize, bins, xscale, ...])</code></td>
<td>Make a 2D hexagonal binning plot of points x, y.</td>
</tr>
<tr>
<td><code>hist(x[, bins, range, density, weights, ...])</code></td>
<td>Plot a histogram.</td>
</tr>
<tr>
<td><code>hist2d(x, y[, bins, range, density, ...])</code></td>
<td>Make a 2D histogram plot.</td>
</tr>
<tr>
<td><code>hlines(y, xmin, xmax[, colors, linestyles, ...])</code></td>
<td>Plot horizontal lines at each y from xmin to xmax.</td>
</tr>
<tr>
<td><code>imshow(X[, cmap, norm, aspect, ...])</code></td>
<td>Display data as an image, i.e., on a 2D regular raster.</td>
</tr>
<tr>
<td><code>in_axes(mouseevent)</code></td>
<td>Return <code>True</code> if the given mouseevent (in display coords) is in the Axes</td>
</tr>
<tr>
<td><code>indicate_inset(bounds[, inset_ax, ...])</code></td>
<td>Add an inset indicator to the axes.</td>
</tr>
<tr>
<td><code>indicate_inset_zoom(inset_ax, **kwargs)</code></td>
<td>Add an inset indicator rectangle to the axes based on the axis limits for an inset_ax and draw connectors between inset_ax and the rectangle.</td>
</tr>
<tr>
<td><code>inset_axes(bounds, *[transform, zorder])</code></td>
<td>Add a child inset axes to this existing axes.</td>
</tr>
<tr>
<td><code>invert_xaxis()</code></td>
<td>Invert the x-axis.</td>
</tr>
<tr>
<td><code>invert_yaxis()</code></td>
<td>Invert the y-axis.</td>
</tr>
<tr>
<td><code>is_transform_set()</code></td>
<td>Return whether the Artist has an explicitly set transform.</td>
</tr>
<tr>
<td><code>legend(*args, **kwargs)</code></td>
<td>Add a legend.</td>
</tr>
<tr>
<td><code>locator_params([axis, tight])</code></td>
<td>Control behavior of major tick locators.</td>
</tr>
<tr>
<td><code>loglog(*args, **kwargs)</code></td>
<td>Make a plot with log scaling on both the x and y axis.</td>
</tr>
<tr>
<td><code>magnitude_spectrum(x[, Fs, Fc, window, ...])</code></td>
<td>Plot the magnitude spectrum.</td>
</tr>
<tr>
<td><code>margins(*margins, x, y, tight)</code></td>
<td>Set or retrieve autoscaling margins.</td>
</tr>
<tr>
<td><code>matshow(Z, **kwargs)</code></td>
<td>Plot the values of a 2D matrix or array as color-coded image.</td>
</tr>
<tr>
<td><code>minorticks_off()</code></td>
<td>Remove minor ticks from the axes.</td>
</tr>
<tr>
<td><code>minorticks_on()</code></td>
<td>Display minor ticks on the axes.</td>
</tr>
<tr>
<td><code>pchanged()</code></td>
<td>Call all of the registered callbacks.</td>
</tr>
<tr>
<td><code>pcolor(*args, **kwargs)</code></td>
<td>Create a pseudocolor plot of a 2-D array.</td>
</tr>
<tr>
<td><code>pcolorfast(*args, alpha, norm, cmap, vmin, ...)</code></td>
<td>Create a pseudocolor plot with a non-regular rectangular grid.</td>
</tr>
<tr>
<td><code>pcolormesh(*args, **kwargs)</code></td>
<td>Create a pseudocolor plot of a 2-D array.</td>
</tr>
<tr>
<td><code>phase_spectrum(x[, Fs, Fc, window, pad_to, ...])</code></td>
<td>Plot the phase spectrum.</td>
</tr>
<tr>
<td><code>pick(mouseevent)</code></td>
<td>Process a pick event.</td>
</tr>
<tr>
<td><code>pickable()</code></td>
<td>Return whether the artist is pickable.</td>
</tr>
<tr>
<td><code>piet(x[, explode, labels, colors, autopct, ...])</code></td>
<td>Plot a pie chart.</td>
</tr>
<tr>
<td><code>plot(*args, **kwargs)</code></td>
<td>Plot lines and/or markers.</td>
</tr>
<tr>
<td><code>plot_date(x, y[, fmt, tz, xdate, ydate, data])</code></td>
<td>Plot data that contains dates.</td>
</tr>
<tr>
<td><code>properties()</code></td>
<td>Return a dictionary of all the properties of the artist.</td>
</tr>
<tr>
<td><code>psd(x[, NFFT, Fs, Fc, detrend, window, ...])</code></td>
<td>Plot the power spectral density.</td>
</tr>
<tr>
<td><code>quiver(*args, data)</code></td>
<td>Plot a 2D field of arrows.</td>
</tr>
<tr>
<td><code>quiverkey(Q, X, Y, U, label, **kw)</code></td>
<td>Add a key to a quiver plot.</td>
</tr>
<tr>
<td><code>redraw_in_frame()</code></td>
<td>Efficiently redraw Axes data, but not axis ticks, labels, etc.</td>
</tr>
</tbody>
</table>

continues on next page
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>relim()</code></td>
<td>Recompute the data limits based on current artists.</td>
</tr>
<tr>
<td><code>remove()</code></td>
<td>Remove the artist from the figure if possible.</td>
</tr>
<tr>
<td><code>remove_callback(oid)</code></td>
<td>Remove a callback based on its observer id.</td>
</tr>
<tr>
<td><code>reset_position()</code></td>
<td>Reset the active position to the original position.</td>
</tr>
<tr>
<td><code>scatter(x, y[, s, c, marker, cmap, norm, ...])</code></td>
<td>A scatter plot of y vs.</td>
</tr>
<tr>
<td><code>secondary_xaxis(location, *[functions])</code></td>
<td>Add a second x-axis to this axes.</td>
</tr>
<tr>
<td><code>secondary_yaxis(location, *[functions])</code></td>
<td>Add a second y-axis to this axes.</td>
</tr>
<tr>
<td><code>semilogx(*args, **kwargs)</code></td>
<td>Make a plot with log scaling on the x axis.</td>
</tr>
<tr>
<td><code>semilogy(*args, **kwargs)</code></td>
<td>Make a plot with log scaling on the y axis.</td>
</tr>
<tr>
<td><code>set(**kwargs)</code></td>
<td>A property batch setter.</td>
</tr>
<tr>
<td><code>set_adjustable(adjustable[, share])</code></td>
<td>Set how the Axes adjusts to achieve the required aspect ratio.</td>
</tr>
<tr>
<td><code>set_agg_filter(filter_func)</code></td>
<td>Set the agg filter.</td>
</tr>
<tr>
<td><code>set_alpha(alpha)</code></td>
<td>Set the alpha value used for blending - not supported on all backends.</td>
</tr>
<tr>
<td><code>set_anchor(anchor[, share])</code></td>
<td>Define the anchor location.</td>
</tr>
<tr>
<td><code>set_animated(b)</code></td>
<td>Set the artist's animation state.</td>
</tr>
<tr>
<td><code>set_aspect(aspect[, adjustable, anchor, share])</code></td>
<td>Set the aspect of the axis scaling, i.e. the ratio of y-unit to x-unit.</td>
</tr>
<tr>
<td><code>set_autoscale_on(b)</code></td>
<td>Set whether autoscaling is applied on plot commands.</td>
</tr>
<tr>
<td><code>set_autoscalex_on(b)</code></td>
<td>Set whether autoscaling for the x-axis is applied on plot commands.</td>
</tr>
<tr>
<td><code>set_autoscaley_on(b)</code></td>
<td>Set whether autoscaling for the y-axis is applied on plot commands.</td>
</tr>
<tr>
<td><code>set_axes_locator(locator)</code></td>
<td>Set the axes locator.</td>
</tr>
<tr>
<td><code>set_axes_off()</code></td>
<td>Turn the x- and y-axis off.</td>
</tr>
<tr>
<td><code>set_axes_on()</code></td>
<td>Turn the x- and y-axis on.</td>
</tr>
<tr>
<td><code>set_axisbelow(b)</code></td>
<td>Set whether axis ticks and gridlines are above or below most artists.</td>
</tr>
<tr>
<td><code>set_box_aspect([aspect])</code></td>
<td>Set the axes box aspect.</td>
</tr>
<tr>
<td><code>set_clip_box(clipbox)</code></td>
<td>Set the artist's clip Bbox.</td>
</tr>
<tr>
<td><code>set_clip_on(b)</code></td>
<td>Set whether the artist uses clipping.</td>
</tr>
<tr>
<td><code>set_clip_path(path[, transform])</code></td>
<td>Set the artist's clip path.</td>
</tr>
<tr>
<td><code>set_contains(picker)</code></td>
<td>[Deprecated] Define a custom contains test for the artist.</td>
</tr>
<tr>
<td><code>set_facecolor(color)</code></td>
<td>Set the facecolor of the Axes.</td>
</tr>
<tr>
<td><code>set_fc(color)</code></td>
<td>Alias for <code>set_facecolor</code>.</td>
</tr>
<tr>
<td><code>set_figure(fig)</code></td>
<td>Set the Figure instance the artist belongs to.</td>
</tr>
<tr>
<td><code>set_frame_on(b)</code></td>
<td>Set whether the axes rectangle patch is drawn.</td>
</tr>
<tr>
<td><code>set_gid(gid)</code></td>
<td>Set the (group) id for the artist.</td>
</tr>
<tr>
<td><code>set_in_layout(in_layout)</code></td>
<td>Set if artist is to be included in layout calculations, E.g.</td>
</tr>
<tr>
<td><code>set_label(s)</code></td>
<td>Set a label that will be displayed in the legend.</td>
</tr>
<tr>
<td><code>set_navigate(b)</code></td>
<td>Set whether the axes responds to navigation toolbar commands.</td>
</tr>
<tr>
<td><code>set_navigate_mode(b)</code></td>
<td>Set the navigation toolbar button status.</td>
</tr>
<tr>
<td><code>set_path_effects(path_effects)</code></td>
<td>Set the path effects.</td>
</tr>
<tr>
<td><code>set_picker(picker)</code></td>
<td>Define the picking behavior of the artist.</td>
</tr>
<tr>
<td><code>set_position(pos[, which])</code></td>
<td>Set the axes position.</td>
</tr>
<tr>
<td><code>set_prop_cycle(*args, **kwargs)</code></td>
<td>Set the property cycle of the Axes.</td>
</tr>
</tbody>
</table>

continues on next page
Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_rasterization_zorder(z)</td>
<td><strong>param z</strong> zorder below which artists are rasterized. None means that artists are not rasterized.</td>
</tr>
<tr>
<td>set_rasterized(rasterized)</td>
<td>Force rasterized (bitmap) drawing in vector backend output.</td>
</tr>
<tr>
<td>set_sketch_params([scale, length, randomness])</td>
<td>Sets the sketch parameters.</td>
</tr>
<tr>
<td>set_snap(snap)</td>
<td>Set the snapping behavior.</td>
</tr>
<tr>
<td>set_title(label[, fontdict, loc, pad, y])</td>
<td>Set a title for the axes.</td>
</tr>
<tr>
<td>set_transform(t)</td>
<td>Set the transform.</td>
</tr>
<tr>
<td>set_url(url)</td>
<td>Set the url for the artist.</td>
</tr>
<tr>
<td>set_visible(b)</td>
<td>Set the artist’s visibility.</td>
</tr>
<tr>
<td>set_xbound([lower, upper])</td>
<td>Set the lower and upper numerical bounds of the x-axis.</td>
</tr>
<tr>
<td>set_xlabel(xlabel[, fontdict, labelpad, loc])</td>
<td>Set the label for the x-axis.</td>
</tr>
<tr>
<td>set_xlim([left, right, emit, auto, xmin, xmax])</td>
<td>Set the x-axis view limits.</td>
</tr>
<tr>
<td>set_xmargin(m)</td>
<td>Set padding of X data limits prior to autoscaling.</td>
</tr>
<tr>
<td>set_xscale(value, **kwargs)</td>
<td>Set the x-axis scale.</td>
</tr>
<tr>
<td>set_xticklabels(labels, *[, fontdict, minor])</td>
<td>Set the xaxis’ labels with list of string labels.</td>
</tr>
<tr>
<td>set_xticks(ticks, *[, minor])</td>
<td>Set the xaxis’ tick locations.</td>
</tr>
<tr>
<td>set_ybound([lower, upper])</td>
<td>Set the lower and upper numerical bounds of the y-axis.</td>
</tr>
<tr>
<td>set_ylabel(ylabel[, fontdict, labelpad, loc])</td>
<td>Set the label for the y-axis.</td>
</tr>
<tr>
<td>set_ylim([bottom, top, emit, auto, ymin, ymax])</td>
<td>Set the y-axis view limits.</td>
</tr>
<tr>
<td>set_ymargin(m)</td>
<td>Set padding of Y data limits prior to autoscaling.</td>
</tr>
<tr>
<td>set_yscale(value, **kwargs)</td>
<td>Set the y-axis scale.</td>
</tr>
<tr>
<td>set_yticklabels(labels, *[, fontdict, minor])</td>
<td>Set the yaxis’ labels with list of string labels.</td>
</tr>
<tr>
<td>set_yticks(ticks, *[, minor])</td>
<td>Set the yaxis’ tick locations.</td>
</tr>
<tr>
<td>set_zorder(level)</td>
<td>Set the zorder for the artist.</td>
</tr>
<tr>
<td>sharex(other)</td>
<td>Share the x-axis with other.</td>
</tr>
<tr>
<td>sharey(other)</td>
<td>Share the y-axis with other.</td>
</tr>
<tr>
<td>specgram(x[, NFFT, Fs, Fc, detrend, window, ...])</td>
<td>Plot a spectrogram.</td>
</tr>
<tr>
<td>spy(Z[, precision, marker, markersize, ...])</td>
<td>Plot the sparsity pattern of a 2D array.</td>
</tr>
<tr>
<td>stackplot(x, *args[, labels, colors, ...])</td>
<td>Draw a stacked area plot.</td>
</tr>
<tr>
<td>start_pan(x, y, button)</td>
<td>Called when a pan operation has started.</td>
</tr>
<tr>
<td>stem(*args[, linefmt, markerfmt, basefmt, ...])</td>
<td>Create a stem plot.</td>
</tr>
<tr>
<td>step(x, y, *args[, where, data])</td>
<td>Make a step plot.</td>
</tr>
<tr>
<td>streamplot(x, y, u, v[, density, linewidth, ...])</td>
<td>Draw streamlines of a vector flow.</td>
</tr>
<tr>
<td>table(cellText, cellColours, cellLoc, ...)</td>
<td>Add a table to an ~.axes.Axes.</td>
</tr>
<tr>
<td>text(x, y, s[, fontdict])</td>
<td>Add text to the axes.</td>
</tr>
<tr>
<td>tick_params([axis])</td>
<td>Change the appearance of ticks, tick labels, and grid-lines.</td>
</tr>
<tr>
<td>ticklabel_format(*[, axis, style, ...])</td>
<td>Configure the .ScalarFormatter used by default for linear axes.</td>
</tr>
<tr>
<td>tricontour(*args, **kwargs)</td>
<td>Draw contour lines on an unstructured triangular grid.</td>
</tr>
<tr>
<td>tricontourf(*args, **kwargs)</td>
<td>Draw contour regions on an unstructured triangular grid.</td>
</tr>
</tbody>
</table>

continues on next page
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>tripcolor(*args[, alpha, norm, cmap, vmin, ...])</code></td>
<td>Create a pseudocolor plot of an unstructured triangular grid.</td>
</tr>
<tr>
<td><code>triplot(*args, **kwargs)</code></td>
<td>Draw a unstructured triangular grid as lines and/or markers.</td>
</tr>
<tr>
<td><code>twinx()</code></td>
<td>Create a twin Axes sharing the xaxis.</td>
</tr>
<tr>
<td><code>twiny()</code></td>
<td>Create a twin Axes sharing the yaxis.</td>
</tr>
<tr>
<td><code>update(props)</code></td>
<td>Update this artist’s properties from the dict <code>props</code>.</td>
</tr>
<tr>
<td><code>update_datalim(xys[, updatex, updatey])</code></td>
<td>Extend the <code>~Axes.dataLim</code> Bbox to include the given points.</td>
</tr>
<tr>
<td><code>update_datalim_bounds(bounds)</code></td>
<td>[Deprecated] Extend the <code>~Axes.dataLim</code> Bbox to include the given <code>~matplotlib.transforms.Bbox</code>.</td>
</tr>
<tr>
<td><code>update_from(other)</code></td>
<td>Copy properties from <code>other</code> to <code>self</code>.</td>
</tr>
<tr>
<td><code>violin(vpstats[, positions, vert, widths, ...])</code></td>
<td>Drawing function for violin plots.</td>
</tr>
<tr>
<td><code>violinplot(dataset[, positions, vert, ...])</code></td>
<td>Make a violin plot.</td>
</tr>
<tr>
<td><code>vlines(x, ymin, ymax[, colors, linestyles, ...])</code></td>
<td>Plot vertical lines.</td>
</tr>
<tr>
<td><code>xaxis_date([tz])</code></td>
<td>Sets up axis ticks and labels to treat data along the xaxis as dates.</td>
</tr>
<tr>
<td><code>xaxis_inverted()</code></td>
<td>Return whether the xaxis is oriented in the “inverse” direction.</td>
</tr>
<tr>
<td><code>xcorr(x, y[, normed, detrend, usevlines, ...])</code></td>
<td>Plot the cross correlation between <code>x</code> and <code>y</code>.</td>
</tr>
<tr>
<td><code>yaxis_date([tz])</code></td>
<td>Sets up axis ticks and labels to treat data along the yaxis as dates.</td>
</tr>
<tr>
<td><code>yaxis_inverted()</code></td>
<td>Return whether the yaxis is oriented in the “inverse” direction.</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>axes</code></td>
<td>The <code>~axes.Axes</code> instance the artist resides in, or <code>None</code>.</td>
</tr>
<tr>
<td><code>is_sideplot</code></td>
<td>If this property is set to <code>True</code>, the artist will be queried for custom context information when the mouse cursor moves over it.</td>
</tr>
<tr>
<td><code>mouseover</code></td>
<td>Whether the artist is ‘stale’ and needs to be re-drawn for the output to match the internal state of the artist.</td>
</tr>
<tr>
<td><code>sticky_edges</code></td>
<td><code>x</code> and <code>y</code> sticky edge lists for autoscaling.</td>
</tr>
<tr>
<td><code>transposed</code></td>
<td>When autoscaling, whether to obey all <code>Artist.sticky_edges</code>.</td>
</tr>
<tr>
<td><code>viewLim</code></td>
<td><code>zorder</code></td>
</tr>
</tbody>
</table>
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):

    __init__(figsize=None, dpi=None, facecolor=None, edgecolor=None, linewidth=0.0, frameon=None, subplotpars=None, tight_layout=None, constrained_layout=None)

Parameters

- figsize (2-tuple of floats, default: :rc:`figure.figsize`) – Figure dimension (width, height) in inches.
- dpi (float, default: :rc:`figure.dpi`) – Dots per inch.
- facecolor (default: :rc:`figure.facecolor`) – The figure patch facecolor.
- edgecolor (default: :rc:`figure.edgecolor`) – The figure patch edge color.
- linewidth (float) – The linewidth of the frame (i.e. the edge linewidth of the figure patch).
- frameon (bool, default: :rc:`figure.frameon`) – If False, suppress drawing the figure background patch.
- subplotpars (SubplotParams) – Subplot parameters. If not given, the default subplot parameters :rc:`figure.subplot.*` are used.
- tight_layout (bool or dict, default: :rc:`figure.autolayout`) – If False use subplotpars. If True adjust subplot parameters using .tight_layout with default padding. When providing a dict containing the keys pad, w_pad, h_pad, and rect, the default .tight_layout paddings will be overridden.
- constrained_layout (bool, default: :rc:`figure.constrained_layout.use`) – If True use constrained layout to adjust positioning of plot elements. Like tight_layout, but designed to be more flexible. See /tutorials/intermediate/constrainedlayout_guide for examples. (Note: does not work with add_subplot or ~/.pyplot.subplot2grid.)

Methods

    __init__([[figsize, dpi, facecolor, ...]])

    param figsize Figure dimension (width, height) in inches.

    add_artist(artist[, clip]) Add an Artist to the figure.
    add_axes(*args, **kwargs) Add an axes to the figure.
    add_axobserver(func) Whenever the axes state change, func(self) will be called.
    add_callback(func) Add a callback function that will be called whenever one of the Artist’s properties changes.
    add_gridspec([nrows, ncols]) Return a GridSpec that has this figure as a parent.
    add_subplot(*args, **kwargs) Add a subplot to the figure.

continues on next page
align_labels([axs])
Align the xlabels and ylabels of subplots with the same subplots row or column (respectively) if label alignment is being done automatically (i.e.

align_xlabels([axs])
Align the xlabels of subplots in the same subplot column if label alignment is being done automatically (i.e.

align_ylabels([axs])
Align the ylabels of subplots in the same subplot column if label alignment is being done automatically (i.e.

autofmt_xdate([bottom, rotation, ha, which])
Date ticklabels often overlap, so it is useful to rotate them and right align them.

clear([keepObservers])
Clear the figure – synonym for clf.

clf([keepObservers])
Clear the figure.

colorbar(mappable[, cax, ax, use_gridspec])
Create a colorbar for a ScalarMappable instance, mappable.

contains(mouseevent)
Test whether the mouse event occurred on the figure.

convert_xunits(x)
Convert x using the unit type of the xaxis.

convert_yunits(y)
Convert y using the unit type of the yaxis.

delaxes(ax)
Remove the ~.axes.Axes ax from the figure; update the current axes.

draw(renderer)
Draw the Artist (and its children) using the given renderer.

draw_artist(a)
Draw Artist instance a only.

execute_constrained_layout([renderer])
Use layoutbox to determine pos positions within axes.

figimage(X[, xo, yo, alpha, norm, cmap, . . .])
Add a non-resampled image to the figure.

findobj([match, include_self])
Find artist objects.

format_cursor_data(data)
Return a string representation of data.

gca(**kwargs)
Get the current axes, creating one if necessary.

get_agg_filter()
Return filter function to be used for agg filter.

get_alpha()
Return the alpha value used for blending - not supported on all backends.

get_animated()
Return whether the artist is animated.

get_axes()
Return a list of axes in the Figure.

get_children()
Get a list of artists contained in the figure.

get_clip_box()
Return the clipbox.

get_clip_on()
Return whether the artist uses clipping.

get_clip_path()
Return the clip path.

get_constrained_layout()
Return whether constrained layout is being used.

get_constrained_layout_pads([relative])
Get padding for constrained_layout.

get_contains()
[Deprecated] Return the custom contains function of the artist if set, or None.

get_cursor_data(event)
Return the cursor data for a given event.

get_default_bbox_extra_artists()

get_dpi()
Return the resolution in dots per inch as a float.

get_edgecolor()
Get the edge color of the Figure rectangle.

get_facecolor()
Get the face color of the Figure rectangle.

get_fignheight()
Return the figure height in inches.

get_figure()
Return the .Figure instance the artist belongs to.

get_figwidth()
Return the figure width in inches.

continues on next page
Table 5 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_frameon()</td>
<td>Return the figure’s background patch visibility, i.e. whether the figure background will be drawn.</td>
</tr>
<tr>
<td>get_gid()</td>
<td>Return the group id.</td>
</tr>
<tr>
<td>get_in_layout()</td>
<td>Return boolean flag, True if artist is included in layout calculations.</td>
</tr>
<tr>
<td>get_label()</td>
<td>Return the label used for this artist in the legend.</td>
</tr>
<tr>
<td>get_path_effects()</td>
<td></td>
</tr>
<tr>
<td>get_picker()</td>
<td>Return the picking behavior of the artist.</td>
</tr>
<tr>
<td>get_rasterized()</td>
<td>Return whether the artist is to be rasterized.</td>
</tr>
<tr>
<td>get_size_inches()</td>
<td>Return the current size of the figure in inches.</td>
</tr>
<tr>
<td>get_sketch_params()</td>
<td>Return the sketch parameters for the artist.</td>
</tr>
<tr>
<td>get_snap()</td>
<td>Return the snap setting.</td>
</tr>
<tr>
<td>get_tight_layout()</td>
<td>Return whether .tight_layout is called when drawing.</td>
</tr>
<tr>
<td>get_tightbbox(renderer[, bbox_extra_artists])</td>
<td>Return a (tight) bounding box of the figure in inches.</td>
</tr>
<tr>
<td>get_transform()</td>
<td>Return the .Transform instance used by this artist.</td>
</tr>
<tr>
<td>get_transformed_clip_path_and_affine()</td>
<td>Return the clip path with the non-affine part of its transformation applied, and the remaining affine part of its transformation.</td>
</tr>
<tr>
<td>get_url()</td>
<td>Return the url.</td>
</tr>
<tr>
<td>get_visible()</td>
<td>Return the visibility.</td>
</tr>
<tr>
<td>get_window_extent(*args, **kwargs)</td>
<td>Return the figure bounding box in display space.</td>
</tr>
<tr>
<td>get_zorder()</td>
<td>Return the artist’s zorder.</td>
</tr>
<tr>
<td>ginput([n, timeout, show_clicks, mouse_add,...])</td>
<td>Blocking call to interact with a figure.</td>
</tr>
<tr>
<td>have_units()</td>
<td>Return True if units are set on any axis.</td>
</tr>
<tr>
<td>init_layoutbox()</td>
<td>Initialize the layoutbox for use in constrained_layout.</td>
</tr>
<tr>
<td>is_transform_set()</td>
<td>Return whether the Artist has an explicitly set transform.</td>
</tr>
<tr>
<td>legend(*args, **kwargs)</td>
<td>Place a legend on the figure.</td>
</tr>
<tr>
<td>pchanged()</td>
<td>Call all of the registered callbacks.</td>
</tr>
<tr>
<td>pick(mouseevent)</td>
<td>Process a pick event.</td>
</tr>
<tr>
<td>pickable()</td>
<td>Return whether the artist is pickable.</td>
</tr>
<tr>
<td>properties()</td>
<td>Return a dictionary of all the properties of the artist.</td>
</tr>
<tr>
<td>remove()</td>
<td>Remove the artist from the figure if possible.</td>
</tr>
<tr>
<td>remove_callback(oid)</td>
<td>Remove a callback based on its observer id.</td>
</tr>
<tr>
<td>savefig(fname, *[transparent])</td>
<td>Save the current figure.</td>
</tr>
<tr>
<td>sca(a)</td>
<td>Set the current axes to be a and return a.</td>
</tr>
<tr>
<td>set(**kwargs)</td>
<td>A property batch setter.</td>
</tr>
<tr>
<td>set_agg_filter(filter_func)</td>
<td>Set the agg filter.</td>
</tr>
<tr>
<td>set_alpha(alpha)</td>
<td>Set the alpha value used for blending - not supported on all backends.</td>
</tr>
<tr>
<td>set_animated(b)</td>
<td>Set the artist’s animation state.</td>
</tr>
<tr>
<td>set_canvas(canvas)</td>
<td>Set the canvas that contains the figure</td>
</tr>
<tr>
<td>set_clip_box(clipbox)</td>
<td>Set the artist’s clip .Bbox.</td>
</tr>
<tr>
<td>set_clip_on(b)</td>
<td>Set whether the artist uses clipping.</td>
</tr>
<tr>
<td>set_clip_path(path[, transform])</td>
<td>Set the artist’s clip path.</td>
</tr>
<tr>
<td>set_constrained_layout(constrained)</td>
<td>Set whether constrained_layout is used upon drawing. If None, :rc:<code>figure.constrained_layout.use</code> value will be used.</td>
</tr>
<tr>
<td>set_constrained_layout_pads(**kwargs)</td>
<td>Set padding for constrained_layout.</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>set_contains(picker)</code></td>
<td>Set a custom contains test for the artist. <code>Deprecated</code></td>
</tr>
<tr>
<td><code>set_dpi(val)</code></td>
<td>Set the resolution of the figure in dots-per-inch.</td>
</tr>
<tr>
<td><code>set_edgecolor(color)</code></td>
<td>Set the edge color of the Figure rectangle.</td>
</tr>
<tr>
<td><code>set_facecolor(color)</code></td>
<td>Set the face color of the Figure rectangle.</td>
</tr>
<tr>
<td><code>set_figheight(val[, forward])</code></td>
<td>Set the height of the figure in inches.</td>
</tr>
<tr>
<td><code>set_figwidth(val[, forward])</code></td>
<td>Set the width of the figure in inches.</td>
</tr>
<tr>
<td><code>set_frameon(b)</code></td>
<td>Set the figure’s background patch visibility, i.e. whether the figure background will be drawn.</td>
</tr>
<tr>
<td><code>set_gid(gid)</code></td>
<td>Set the (group) id for the artist.</td>
</tr>
<tr>
<td><code>set_in_layout(in_layout)</code></td>
<td>Set if artist is to be included in layout calculations, e.g.</td>
</tr>
<tr>
<td><code>set_label(s)</code></td>
<td>Set a label that will be displayed in the legend.</td>
</tr>
<tr>
<td><code>set_path_effects(path_effects)</code></td>
<td>Set the path effects.</td>
</tr>
<tr>
<td><code>set_picker(picker)</code></td>
<td>Define the picking behavior of the artist.</td>
</tr>
<tr>
<td><code>set_rasterized(rasterized)</code></td>
<td>Force rasterized (bitmap) drawing in vector backend output.</td>
</tr>
<tr>
<td><code>set_size_inches(w[, h, forward])</code></td>
<td>Set the figure size in inches.</td>
</tr>
<tr>
<td><code>set_sketch_params([scale, length, randomness])</code></td>
<td>Sets the sketch parameters.</td>
</tr>
<tr>
<td><code>set_snap(snap)</code></td>
<td>Set the snapping behavior.</td>
</tr>
<tr>
<td><code>set_tight_layout(tight)</code></td>
<td>Set whether and how <code>.tight_layout</code> is called when drawing.</td>
</tr>
<tr>
<td><code>set_transform(t)</code></td>
<td>Set the artist transform.</td>
</tr>
<tr>
<td><code>set_url(url)</code></td>
<td>Set the url for the artist.</td>
</tr>
<tr>
<td><code>set_visible(b)</code></td>
<td>Set the artist’s visibility.</td>
</tr>
<tr>
<td><code>set_zorder(level)</code></td>
<td>Set the zorder for the artist.</td>
</tr>
<tr>
<td><code>show([warn])</code></td>
<td>If using a GUI backend with pyplot, display the figure window.</td>
</tr>
<tr>
<td><code>subplot_mosaic(layout, *[subplot_kw, ...])</code></td>
<td>Build a layout of Axes based on ASCII art or nested lists.</td>
</tr>
<tr>
<td><code>subplots([nrows, ncols, sharex, sharey, ...])</code></td>
<td>Add a set of subplots to this figure.</td>
</tr>
<tr>
<td><code>subplots_adjust([left, bottom, right, top, ...])</code></td>
<td>Adjust the subplot layout parameters.</td>
</tr>
<tr>
<td><code>suptitle(t, **kwargs)</code></td>
<td>Add a centered title to the figure.</td>
</tr>
<tr>
<td><code>text(x, y, s[, fontdict])</code></td>
<td>Add text to figure.</td>
</tr>
<tr>
<td><code>tight_layout([renderer, pad, h_pad, w_pad, rect])</code></td>
<td>Adjust the padding between and around subplots.</td>
</tr>
<tr>
<td><code>update(props)</code></td>
<td>Update this artist’s properties from the dict <code>props</code>.</td>
</tr>
<tr>
<td><code>update_from(other)</code></td>
<td>Copy properties from <code>other</code> to <code>self</code>.</td>
</tr>
<tr>
<td><code>waitforbuttonpress([timeout])</code></td>
<td>Blocking call to interact with the figure.</td>
</tr>
</tbody>
</table>
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>axes</td>
<td>List of axes in the Figure.</td>
</tr>
<tr>
<td>dpi</td>
<td>The resolution in dots per inch.</td>
</tr>
<tr>
<td>frameon</td>
<td>Return the figure’s background patch visibility, i.e. whether the figure background will be drawn.</td>
</tr>
<tr>
<td>mouseover</td>
<td>If this property is set to True, the artist will be queried for custom context information when the mouse cursor moves over it.</td>
</tr>
<tr>
<td>stale</td>
<td>Whether the artist is ‘stale’ and needs to be re-drawn for the output to match the internal state of the artist.</td>
</tr>
<tr>
<td>sticky_edges x</td>
<td>x and y sticky edge lists for autoscaling.</td>
</tr>
<tr>
<td>zorder</td>
<td></td>
</tr>
</tbody>
</table>

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)

GridSpec.

__init__ (nrows, ncols, figure=None, left=None, bottom=None, right=None, top=None, wspace=None, hspace=None, width_ratios=None, height_ratios=None)

Parameters

- **nrows (int)** – The number of rows and columns of the grid.
- **ncols (int)** – The number of rows and columns of the grid.
- **figure (~figure.Figure, optional)** – Only used for constrained layout to create a proper layoutbox.
- **left (float, optional)** – Extent of the subplots as a fraction of figure width or height. Left cannot be larger than right, and bottom cannot be larger than top. If not given, the values will be inferred from a figure or rcParams at draw time. See also GridSpec.get_subplot_params.
- **right (float, optional)** – Extent of the subplots as a fraction of figure width or height. Left cannot be larger than right, and bottom cannot be larger than top. If not given, the values will be inferred from a figure or rcParams at draw time. See also GridSpec.get_subplot_params.
- **top (float, optional)** – Extent of the subplots as a fraction of figure width or height. Left cannot be larger than right, and bottom cannot be larger than top. If not given, the values will be inferred from a figure or rcParams at draw time. See also GridSpec.get_subplot_params.
- **bottom (float, optional)** – Extent of the subplots as a fraction of figure width or height. Left cannot be larger than right, and bottom cannot be larger than top. If not given, the values will be inferred from a figure or rcParams at draw time. See also GridSpec.get_subplot_params.
- **wspace (float, optional)** – The amount of width reserved for space between subplots, expressed as a fraction of the average axis width. If not given, the values will be inferred from a figure or rcParams when necessary. See also GridSpec.get_subplot_params.
• **hspace** (*float, optional*) – The amount of height reserved for space between subplots, expressed as a fraction of the average axis height. If not given, the values will be inferred from a figure or rcParams when necessary. See also `GridSpec.get_subplot_params`.

• **width_ratios** (*array-like of length ncols*, optional) – Defines the relative widths of the columns. Each column gets a relative width of `width_ratios[i] / sum(width_ratios)`. If not given, all columns will have the same width.

• **height_ratios** (*array-like of length nrows*, optional) – Defines the relative heights of the rows. Each column gets a relative height of `height_ratios[i] / sum(height_ratios)`. If not given, all rows will have the same height.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong><strong>init</strong></strong>(nrows, ncols[, figure, left, ...])</td>
<td>param nrows The number of rows and columns of the grid.</td>
</tr>
<tr>
<td>get_geometry()</td>
<td>Return a tuple containing the number of rows and columns in the grid.</td>
</tr>
<tr>
<td>get_grid_positions(fig[, raw])</td>
<td>Return the positions of the grid cells in figure coordinates.</td>
</tr>
<tr>
<td>get_height_ratios()</td>
<td>Return the height ratios.</td>
</tr>
<tr>
<td>get_subplot_params([figure])</td>
<td>Return the <code>~SubplotParams</code> for the GridSpec.</td>
</tr>
<tr>
<td>get_width_ratios()</td>
<td>Return the width ratios.</td>
</tr>
<tr>
<td>locally_modified_subplot_params()</td>
<td>Return a list of the names of the subplot parameters explicitly set in the GridSpec.</td>
</tr>
<tr>
<td>new_subplotspec(loc[, rowspan, colspan])</td>
<td>Create and return a <code>SubplotSpec</code> instance.</td>
</tr>
<tr>
<td>set_height_ratios(height_ratios)</td>
<td>Set the relative heights of the rows.</td>
</tr>
<tr>
<td>set_width_ratios(width_ratios)</td>
<td>Set the relative widths of the columns.</td>
</tr>
<tr>
<td>subplots(*[, sharex, sharey, squeeze, ...])</td>
<td>Add all subplots specified by this <code>GridSpec</code> to its parent figure.</td>
</tr>
<tr>
<td>tight_layout(figure[, renderer, pad, h_pad, ...])</td>
<td>Adjust subplot parameters to give specified padding.</td>
</tr>
<tr>
<td>update(**kwargs)</td>
<td>Update the subplot parameters of the grid.</td>
</tr>
</tbody>
</table>

### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ncols</td>
<td>The number of columns in the grid.</td>
</tr>
<tr>
<td>nrows</td>
<td>The number of rows in the grid.</td>
</tr>
</tbody>
</table>
WrightTools.artists.add_sideplot

WrightTools.artists.add_sideplot(ax, along=0, 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** (dict {'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** (dict {'default', 'fast', 'publication'} (optional)) – Settings to use. Default is ‘fast’.

WrightTools.artists.colormaps

WrightTools.artists.colormaps = {'cubehelix': <matplotlib.colors.LinearSegmentedColormap object>, 'default': ..., 'turbo': <matplotlib.colors.ListedColormap object>, 'wright': <matplotlib.colors.LinearSegmentedColormap object>}

WrightTools.artists.corner_text

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

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.

• `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

Other Parameters: **kwargs (matplotlib.text.Text properties.) – Other miscellaneous text parameters passed to ax.text. Default font size is 18.

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 seperating two subplots vertically), in inches. Default is 0.25.

• `wspace` (float (optional)) – The ‘width space’ (space seperating 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[:, -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.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

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**

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**

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

Prepare a set of arrays for plotting using pcolor.

This function is Deprecated as of WrightTools 3.3.0. Matplotlib introduced the shading="nearest" in version 3.3.0 on pcolor and associated methods, which accomplishes the same goal, in a much cleaner way.

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**

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 Documentation, Release 3.3.1

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 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.
- **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 (deulany) 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)** –

- ------------------
- **kwargs** (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, palette_size=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 apperence.
- **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.
- **palette_size** *(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.

| **Collection** (*args, **kwargs) | Nestable Collection of Data objects. |
| **from_Cary**(*filepath[, name, parent, verbose]) | Create a collection object from a Cary UV VIS absorbance file. |
| **from_directory**(*filepath, from_methods, *[...]) | Create a WrightTools Collection from a directory of source files. |
WrightTools.collection.Collection

class WrightTools.collection.Collection(*args, **kwargs)
    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

__init__(file, parent, name)] Create a new Group object by binding to a low-level
GroupID.
clear()
    Close the file that contains the Group.
copy(parent, name, verbose)] Create a copy under parent.
create_collection(name, position)] Create a new child collection.
create_data(name, position)] Create a new child data.
create_dataset(name, shape, dtype, data)] Create a new HDF5 dataset
create_dataset_like(name, other, **kwargs)] Create a dataset similar to other.
create_group(name, track_order)] Create and return a new subgroup.
create_virtual_dataset(name, layout, fillvalue)] Create a new virtual dataset in this group.
flush)] Ensure contents are written to file.
get(name, default, getclass, getlink)] Retrieve an item or other information.
index)] Index.
items)] Get a view object on member items
keys)] Get a view object on member names
move(source, dest)] Move a link to a new location in the file.
pop(k, d)] If key is not found, d is returned if given, otherwise
        KeyError is raised.
popitem() as a 2-tuple; but raise KeyError if D is empty.
print_tree(depth, verbose)] Print a ascii-formatted tree representation of the collection contents.
require_dataset(name, shape, dtype, exact)] Open a dataset, creating it if it doesn’t exist.
require_group(name)] Return a group, creating it if it doesn’t exist.
save(filepath, overwrite, verbose)] Save as root of a new file.
setdefault(k, d)]
update(E, **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
values)] Get a view object on member objects
visit(func)] Recursively visit all names in this group and sub-
groups (HDF5 1.8).
visititems(func)] Recursively visit names and objects in this group
(HDF5 1.8).
Attributes

- **attrs** Attributes attached to this object
- **class_name**
- **created**
- **file** Return a File instance associated with this object
- **fullpath** file and internal structure.
- **id** Low-level identifier appropriate for this object
- **item_names** Item names.
- **name** Return the full name of this object.
- **natural_name** Natural name.
- **parent** Parent.
- **ref** An (opaque) HDF5 reference to this object
- **regionref** Create a region reference (Datasets only).

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

Multidimensional dataset.

#### __init__(*args, **kwargs)

Create a new Group object by binding to a low-level GroupID.

### Methods

- **__init__(*args, **kwargs)**: Create a new Group object by binding to a low-level GroupID.
- **bring_to_front(channel)**: Bring a specific channel to the zero-indexed position in channels.
- **chop(*args[, at, parent, verbose])**: Divide the dataset into its lower-dimensionality components.
- **clear()**: Close the file that contains the Group.
- **close()**: Close the file that contains the Group.
- **collapse(axis[, method])**: Collapse the dataset along one axis, adding lower rank channels.
- **convert(destination_units, *[...])**: Convert all compatible axes and constants to given units.
- **copy([parent, name, verbose])**: Create a copy under parent.
Table 13 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>create_channel(name[, values, shape, units, ...])</code></td>
<td>Append a new channel.</td>
</tr>
<tr>
<td><code>create_constant(expression, *[, verbose])</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_dataset_like(name, other, **kwargs)</code></td>
<td>Create a dataset similar to <code>other</code>.</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>create_virtual_dataset(name, layout[, fill_value])</code></td>
<td>Create a new virtual dataset in this group.</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>Retrieve an item or other information.</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>items()</code></td>
<td>Get a view object on member items.</td>
</tr>
<tr>
<td><code>keys()</code></td>
<td>Get a view object on member names.</td>
</tr>
<tr>
<td><code>level(channel, axis, npts[, method, verbose])</code></td>
<td>Subtract the average value of <code>npts</code> 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>
<tr>
<td><code>move(source, dest)</code></td>
<td>Move a link to a new location in the file.</td>
</tr>
<tr>
<td><code>offset(points, offsets, along, offset_axis)</code></td>
<td>Offset one axis based on another axis' values.</td>
</tr>
<tr>
<td><code>pop(k[, d])</code></td>
<td>If key is not found, <code>d</code> is returned if given, otherwise <code>KeyError</code> is raised.</td>
</tr>
<tr>
<td><code>popitem()</code></td>
<td>As a 2-tuple; but raise <code>KeyError</code> if D is empty.</td>
</tr>
<tr>
<td><code>print_tree(*[, verbose])</code></td>
<td>Print an ascii-formatted tree representation of the data contents.</td>
</tr>
<tr>
<td><code>prune([keep_channels, verbose])</code></td>
<td>Remove unused variables and (optionally) channels from the Data object.</td>
</tr>
<tr>
<td><code>remove_channel(channel[, verbose])</code></td>
<td>Remove channel from data.</td>
</tr>
<tr>
<td><code>remove_constant(constant[, verbose])</code></td>
<td>Remove a constant from the stored list.</td>
</tr>
<tr>
<td><code>remove_variable(variable, *[, implied, verbose])</code></td>
<td>Remove variable from data.</td>
</tr>
<tr>
<td><code>rename_channels(*[, verbose])</code></td>
<td>Rename a set of channels.</td>
</tr>
<tr>
<td><code>rename_variables(*[, implied, verbose])</code></td>
<td>Rename a set of variables.</td>
</tr>
<tr>
<td><code>require_dataset(name, shape, dtype[, exact])</code></td>
<td>Open a dataset, creating it if it doesn’t exist.</td>
</tr>
<tr>
<td><code>require_group(name)</code></td>
<td>Return a group, creating it if it doesn’t exist.</td>
</tr>
<tr>
<td><code>save([filepath, overwrite, verbose])</code></td>
<td>Save as root of a new file.</td>
</tr>
<tr>
<td><code>set_constants(*constants[, verbose])</code></td>
<td>Set the constants associated with the data.</td>
</tr>
<tr>
<td><code>setdefault(k[, d])</code></td>
<td></td>
</tr>
<tr>
<td><code>share_nans()</code></td>
<td>Share not-a-numbers between all channels.</td>
</tr>
<tr>
<td><code>smooth(factors[, channel, verbose])</code></td>
<td>Smooth a channel using an n-dimensional kaiser window.</td>
</tr>
<tr>
<td><code>split(expression, positions[, ...])</code></td>
<td>Split the data object along a given expression, in units.</td>
</tr>
</tbody>
</table>

continues on next page
### Table 13 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>transform(*axes[, verbose])</code></td>
<td>Transform the data.</td>
</tr>
<tr>
<td><code>update([E, ]**F)</code></td>
<td>If E present and has a <code>.keys()</code> method, does: for k in E: D[k] = E[k] If E present and lacks <code>.keys()</code> 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>values()</code></td>
<td>Get a view object on member objects</td>
</tr>
<tr>
<td><code>visit(func)</code></td>
<td>Recursively visit all names in this group and subgroups (HDF5 1.8).</td>
</tr>
<tr>
<td><code>visititems(func)</code></td>
<td>Recursively visit names and objects in this group (HDF5 1.8).</td>
</tr>
<tr>
<td><code>zoom(factor[, order, verbose])</code></td>
<td>Zoom the data array using spline interpolation of the requested order.</td>
</tr>
</tbody>
</table>

### Attributes

- `attrs` Attributes attached to this object
- `axes` Axis expressions.
- `axis_expressions` Axis expressions.
- `axis_names` Axis names.
- `channel_names` Channel names.
- `channels` Channels.
- `class_name` Class name.
- `constant_expressions` Axis expressions.
- `constant_names` Axis names.
- `constant_units` All constant units.
- `constants` Constants.
- `created` Created.
- `created` Created.
- `created` Created.
- `created` Created.
- `datasets` Datasets.
- `file` Return a File instance associated with this object.
- `fullpath` file and internal structure.
- `id` Low-level identifier appropriate for this object.
- `item_names` Item names.
- `kind` Kind.
- `name` Return the full name of this object.
- `natural_name` Natural name.
- `ndim` Get number of dimensions.
- `parent` Parent.
- `ref` An (opaque) HDF5 reference to this object.
- `regionref` Create a region reference (Datasets only).
- `shape` Shape.
- `size` Size.
- `source` Source.
- `units` All axis units.
- `variable_names` Variable names.
- `variables` Variables.
WrightTools.data.Axis

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

 initiate (parent, expression[, units]) Data axis.
 convert(destination_units, *[,...]) Convert axis to destination_units.
 max() Axis max.
 min() Axis min.

Attributes

 full Axis expression evaluated and repeated to match the shape of the parent data object.
 identity Complete identifier written to disk in data.attrs[‘axes’].
 label A latex formatted label representing axis expression.
 masked Axis expression evaluated, and masked with NaN shared from data channels.
 natural_name Valid python identifier representation of the expression.
 ndim Get number of dimensions.
 points Squeezed array.
 shape Shape.
 size Size.
 units_kind Units kind.
 variables Variables.

WrightTools.data.Channel

class WrightTools.data.Channel(parent, id, **kwargs)
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><strong><strong>init</strong></strong></td>
<td>Construct a channel object.</td>
</tr>
<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>astype(dtype)</td>
<td>Get a context manager allowing you to perform reads to a different destination type, e.g.:</td>
</tr>
<tr>
<td>chunkwise(func, **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>flush()</td>
<td>Flush the dataset data and metadata to the file.</td>
</tr>
<tr>
<td>len()</td>
<td>The size of the first axis.</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>make_scale([name])</td>
<td>Make this dataset an HDF5 dimension scale.</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>read_direct(dest[, source_sel, dest_sel])</td>
<td>Read data directly from HDF5 into an existing NumPy array.</td>
</tr>
<tr>
<td>refresh()</td>
<td>Refresh the dataset metadata by reloading from the file.</td>
</tr>
<tr>
<td>resize(size[, axis])</td>
<td>Resize the dataset, or the specified axis.</td>
</tr>
<tr>
<td>slices()</td>
<td>Returns a generator yielding tuple of slice objects.</td>
</tr>
<tr>
<td>symmetric_root([root])</td>
<td>Remove outliers from the dataset.</td>
</tr>
<tr>
<td>virtual_sources()</td>
<td>Write data directly to HDF5 from a NumPy array.</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>
## 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>chunks</td>
<td>Dataset chunks (or None)</td>
</tr>
<tr>
<td>class_name</td>
<td>Compression strategy (or None)</td>
</tr>
<tr>
<td>compression</td>
<td>Compression setting</td>
</tr>
<tr>
<td>compression_opts</td>
<td>Access dimension scales attached to this dataset.</td>
</tr>
<tr>
<td>dims</td>
<td>Numpy dtype representing the datatype</td>
</tr>
<tr>
<td>external</td>
<td>External file settings</td>
</tr>
<tr>
<td>file</td>
<td>Return a File instance associated with this object</td>
</tr>
<tr>
<td>fillvalue</td>
<td>Fill value for this dataset (0 by default)</td>
</tr>
<tr>
<td>fletcher32</td>
<td>Fletcher32 filter is present (T/F)</td>
</tr>
<tr>
<td>full</td>
<td>File and internal structure.</td>
</tr>
<tr>
<td>id</td>
<td>Low-level identifier appropriate for this object</td>
</tr>
<tr>
<td>is_virtual</td>
<td>Maximum deviation from null.</td>
</tr>
<tr>
<td>major_extent</td>
<td>Shape up to which this dataset can be resized.</td>
</tr>
<tr>
<td>min_extent</td>
<td>Minimum deviation from null.</td>
</tr>
<tr>
<td>name</td>
<td>Return the full name of this object</td>
</tr>
<tr>
<td>natural_name</td>
<td>Natural name of the dataset.</td>
</tr>
<tr>
<td>ndim</td>
<td>Numpy-style attribute giving the number of dimensions</td>
</tr>
<tr>
<td>null</td>
<td>Parent.</td>
</tr>
<tr>
<td>parent</td>
<td>Squeezed array.</td>
</tr>
<tr>
<td>points</td>
<td>An (opaque) HDF5 reference to this object</td>
</tr>
<tr>
<td>ref</td>
<td>Create a region reference (Datasets only).</td>
</tr>
<tr>
<td>regionref</td>
<td>Scale/offset filter settings.</td>
</tr>
<tr>
<td>shape</td>
<td>Numpy-style shape tuple giving dataset dimensions</td>
</tr>
<tr>
<td>shuffle</td>
<td>Shuffle filter present (T/F)</td>
</tr>
<tr>
<td>signed</td>
<td>Numpy-style attribute giving the total dataset size</td>
</tr>
<tr>
<td>size</td>
<td>Units.</td>
</tr>
<tr>
<td>value</td>
<td>Alias for dataset[()]</td>
</tr>
</tbody>
</table>

### WrightTools.data.Constant

```python
class WrightTools.data.Constant (parent, expression, units=None, format_spec='0.3g', round_spec=None)
```

Constant class.

```python
__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><strong>init</strong></td>
<td>Data constant.</td>
</tr>
<tr>
<td>convert</td>
<td>Convert axis to destination_units.</td>
</tr>
<tr>
<td>max()</td>
<td>Axis max.</td>
</tr>
<tr>
<td>min()</td>
<td>Axis min.</td>
</tr>
</tbody>
</table>

### Attributes

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

WrightTools.data.Variable

```python
class WrightTools.data.Variable(parent, id, **kwargs)
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><code>__init__</code></td>
<td>Variable.</td>
</tr>
<tr>
<td><code>argmax()</code></td>
<td>Index of the maximum, ignoring nans.</td>
</tr>
<tr>
<td><code>argmin()</code></td>
<td>Index of the minimum, ignoring nans.</td>
</tr>
<tr>
<td><code>astype(dtype)</code></td>
<td>Get a context manager allowing you to perform reads to a different destination type, e.g.:</td>
</tr>
<tr>
<td><code>chunkwise(func, *args, **kwargs)</code></td>
<td>Execute a function for each chunk in the dataset.</td>
</tr>
<tr>
<td><code>clip([min, max, replace])</code></td>
<td>Clip values outside of a defined range.</td>
</tr>
<tr>
<td><code>convert(destination_units)</code></td>
<td>Convert units.</td>
</tr>
<tr>
<td><code>flush()</code></td>
<td>Flush the dataset data and metadata to the file.</td>
</tr>
<tr>
<td><code>len()</code></td>
<td>The size of the first axis.</td>
</tr>
<tr>
<td><code>log([base, floor])</code></td>
<td>Take the log of the entire dataset.</td>
</tr>
<tr>
<td><code>log10([floor])</code></td>
<td>Take the log base 10 of the entire dataset.</td>
</tr>
<tr>
<td><code>log2([floor])</code></td>
<td>Take the log base 2 of the entire dataset.</td>
</tr>
<tr>
<td><code>make_scale(name)</code></td>
<td>Make this dataset an HDF5 dimension scale.</td>
</tr>
<tr>
<td><code>max()</code></td>
<td>Maximum, ignoring nans.</td>
</tr>
<tr>
<td><code>min()</code></td>
<td>Minimum, ignoring nans.</td>
</tr>
<tr>
<td><code>read_direct(dest[, source_sel, dest_sel])</code></td>
<td>Read data directly from HDF5 into an existing NumPy array.</td>
</tr>
<tr>
<td><code>refresh()</code></td>
<td>Refresh the dataset metadata by reloading from the file.</td>
</tr>
<tr>
<td><code>resize(size[, axis])</code></td>
<td>Resize the dataset, or the specified axis.</td>
</tr>
<tr>
<td><code>slices()</code></td>
<td>Returns a generator yielding tuple of slice objects.</td>
</tr>
<tr>
<td><code>symmetric_root([root])</code></td>
<td></td>
</tr>
<tr>
<td><code>virtual_sources()</code></td>
<td></td>
</tr>
<tr>
<td><code>write_direct(source[, source_sel, dest_sel])</code></td>
<td>Write data directly to HDF5 from a NumPy array.</td>
</tr>
</tbody>
</table>

## Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>attrs</code></td>
<td>Attributes attached to this object.</td>
</tr>
<tr>
<td><code>chunks</code></td>
<td>Dataset chunks (or None)</td>
</tr>
<tr>
<td><code>class_name</code></td>
<td>Compression strategy (or None)</td>
</tr>
<tr>
<td><code>compression</code></td>
<td>Compression setting.</td>
</tr>
<tr>
<td><code>compression_opts</code></td>
<td>Access dimension scales attached to this dataset.</td>
</tr>
<tr>
<td><code>dtype</code></td>
<td>Numpy dtype representing the datatype</td>
</tr>
<tr>
<td><code>external</code></td>
<td>External file settings.</td>
</tr>
<tr>
<td><code>file</code></td>
<td>Return a File instance associated with this object.</td>
</tr>
<tr>
<td><code>fillvalue</code></td>
<td>Fill value for this dataset (0 by default)</td>
</tr>
<tr>
<td><code>fletcher32</code></td>
<td>Fletcher32 filter is present (T/F)</td>
</tr>
<tr>
<td><code>full</code></td>
<td>file and internal structure.</td>
</tr>
<tr>
<td><code>id</code></td>
<td>Low-level identifier appropriate for this object.</td>
</tr>
<tr>
<td><code>is_virtual</code></td>
<td>Shape up to which this dataset can be resized.</td>
</tr>
<tr>
<td><code>label</code></td>
<td>Return the full name of this object.</td>
</tr>
<tr>
<td><code>natural_name</code></td>
<td>Natural name of the dataset.</td>
</tr>
</tbody>
</table>
Table 22 – continued from previous page

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndim</td>
<td>Numpy-style attribute giving the number of dimensions</td>
</tr>
<tr>
<td>parent</td>
<td>Parent.</td>
</tr>
<tr>
<td>points</td>
<td>Squeezed array.</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>scaleoffset</td>
<td>Scale/offset filter settings.</td>
</tr>
<tr>
<td>shape</td>
<td>Numpy-style shape tuple giving dataset dimensions</td>
</tr>
<tr>
<td>shuffle</td>
<td>Shuffle filter present (T/F)</td>
</tr>
<tr>
<td>size</td>
<td>Numpy-style attribute giving the total dataset size</td>
</tr>
<tr>
<td>units</td>
<td>Units.</td>
</tr>
<tr>
<td>value</td>
<td>Alias for dataset[()]</td>
</tr>
</tbody>
</table>

WrightTools.data.join

WrightTools.data.join\(\text{(datas, *, atol=None, rtol=None, name='join', parent=None, method='first', verbose=True)}\) → 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_BrunoldrRaman

WrightTools.data.from_BrunoldrRaman(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, 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_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 (boolean (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, 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_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** *(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_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.
WrightTools Documentation, Release 3.3.1

- **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 homepage 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, file-name 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.CuFCTs_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, 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.

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.*

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

Add arrow.

**Parameters**

- **diagram** *(list of integers (optional)) – Indices of states which are virtual. Default is [None].*
- **number** *(number (optional)) – State font size. Default is 8.*
- **between** *(number (optional)) – Size of buffer around state text. Default is 0.5.*

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

```python
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)) – Arrow head length. Default 0.075.*
- **color** *(matplotlib color) – Arrow color. Default is ‘k’.*

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

**Return type** list

```python
clear_diagram(diagram)
```

Clear diagram.

**Parameters**

- **diagram** ([column, row]) – Diagram to clear.

```python
clear_diagram(diagram)
```

```python
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.*

```python
label_columns(labels, font_size=15, text_buffer=1.15)
```

```python
label_rows(labels, font_size=15, text_buffer=1.15)
```

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.*

```python
label_rows(labels, font_size=15, text_buffer=1.15)
```

```python
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.*

```python
plot(save_path=None, close=False, bbox_inches='tight', pad_inches=1)
```
• `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')

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 vitual energy states

• `state_font_size(numtype (optional))` – font size for the state lables

• `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.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>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DimensionalityError(expected, recieved)</td>
<td>DimensionalityError.</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(axis, operation)</td>
<td>Error for when operation does not support Multidimensional Axes.</td>
</tr>
<tr>
<td>NameNotUniqueError([name])</td>
<td>NameNotUniqueError.</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(expected, recieved)</td>
<td>Units Error.</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>VisibleDeprecationWarning.</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>WrongFileTypeWarning.</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)
     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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INI(filepath)</strong></td>
<td>Handle communication with an INI file.</td>
</tr>
<tr>
<td><strong>Spline(xi, yi[, k, s, ignore_nans])</strong></td>
<td>Spline.</td>
</tr>
<tr>
<td><strong>TimeStamp([at, timezone])</strong></td>
<td>Class for representing a moment in time.</td>
</tr>
<tr>
<td><strong>Timer([verbose])</strong></td>
<td>Context manager for timing code.</td>
</tr>
<tr>
<td><strong>closest_pair(arr[, give])</strong></td>
<td>Find the pair of indices corresponding to the closest elements in an array.</td>
</tr>
<tr>
<td><strong>diff(xi, yi[, order])</strong></td>
<td>Take the numerical derivative of a 1D array.</td>
</tr>
<tr>
<td><strong>discover_dimensions(arr, cols)</strong></td>
<td>Discover the dimensions of a flattened multidimensional array.</td>
</tr>
<tr>
<td><strong>enforce_mask_shape(mask, shape)</strong></td>
<td>Reduce a boolean mask to fit a given shape.</td>
</tr>
<tr>
<td><strong>fft(xi, yi[, axis])</strong></td>
<td>Take the 1D FFT of an N-dimensional array and return “sensible” properly shifted arrays.</td>
</tr>
<tr>
<td><strong>flatten_list(items[, seqtypes, in_place])</strong></td>
<td>Flatten an irregular sequence.</td>
</tr>
<tr>
<td><strong>fluence(power_mW, color, beam_radius, ...[, ...])</strong></td>
<td>Calculate the fluence of a beam.</td>
</tr>
</tbody>
</table>
Table 24 – 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>Return a list of all files matching specified inputs.</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>svd(a[, i])</code></td>
<td>Singular Value Decomposition.</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, yi_zoom, ...])</code></td>
<td>Zoom a 2D array, with axes.</td>
</tr>
</tbody>
</table>

WrightTools.kit.INI

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

Handle communication with an INI file.

**__init__**(filepath)

Create an INI handler object.

**Parameters**

filepath (path-like) – Filepath.

**Methods**

**__init__**(filepath)

Create an INI handler object.

add_section(section)

Add section.

clear()

Remove all contents from file.

get_options(section)

List the options in a section.

has_option(section, option)

Test if file has option.

has_section(section)

Test if file has section.

read(section, option)

Read from file.

write(section, option, value)

Write to file.
Attributes

<table>
<thead>
<tr>
<th>dictionary</th>
<th>Get a python dictionary of contents.</th>
</tr>
</thead>
<tbody>
<tr>
<td>sections</td>
<td>List of sections.</td>
</tr>
</tbody>
</table>

WrightTools.kit.Spline

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

Spline.

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

Initialize.

Parameters

- **xi** *(1D array)* – x points.
- **yi** *(1D array)* – y points.
- **k** *(integer (optional))* – Degree of smoothing. Must be between 1 and 5 (inclusive). Default is 3.
- **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:

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

If 0, spline will interpolate through all data points. Default is 1000.
- **ignore_nans** *(boolean (optional))* – Toggle removal of nans. Default is True.

Note: Use k=1 and s=0 for a linear interpolation.

Methods

__init__(xi, yi[, k, s, ignore_nans])

Initialize.

WrightTools.kit.TimeStamp

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

Class for representing a moment in time.

__init__(at=None, timezone='local')

Create a TimeStamp object.

Parameters

- **at** *(float (optional))* – Seconds since epoch (unix time). If None, current time will be used. Default is None.
- **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

def date
  Date.
  Type  string

def hms
  Hours, minutes, seconds.
  Type  string

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

def legacy
  Legacy WrightTools timestamp representation.
  Type  string

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

def RFC5322
  RFC5322 representation.
  Type  string

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

Methods

__init__(at, timezone)  Create a TimeStamp object.

Attributes

<table>
<thead>
<tr>
<th>RFC3339</th>
<th>RFC3339.</th>
</tr>
</thead>
<tbody>
<tr>
<td>RFC5322</td>
<td>RFC5322.</td>
</tr>
<tr>
<td>date</td>
<td>year-month-day.</td>
</tr>
<tr>
<td>hms</td>
<td>Get time formatted.</td>
</tr>
<tr>
<td>human</td>
<td>Human-readable timestamp.</td>
</tr>
<tr>
<td>path</td>
<td>Timestamp for placing into filepaths.</td>
</tr>
</tbody>
</table>
**WrightTools.kit.Timer**

**class WrightTools.kit.Timer(verbosetrue)**

Context manager for timing code.

```python
>>> with Timer():
...     your_code()
```

__init__(verbosetrue)

Initialize self. See help(type(self)) for accurate signature.

**Methods**

__init__(verbosetrue) Initialize self.

**WrightTools.kit.closest_pair**

WrightTools.kit.closest_pair(arr, give='indicies')

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.

```python
>>> arr = np.array([0, 1, 2, 3, 3, 4, 5, 6, 1])
>>> closest_pair(arr)
[[[1,], [8,]], [[3,], [4,]]]
```

**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) \rightarrow 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) \rightarrow 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.

**Example**

```python
def flatten_list_example():
    l = [[[1, 2, 3], [4, 5]], 6]
    return wt.kit.flatten_list(l)
```

```python
>>> flatten_list_example()
[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.

**Example**

```python
def fluence_example():
    power_mW = 1000
    color = 5
    beam_radius = 0.5
    reprate_Hz = 100
    pulse_width = 10
    return wt.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')
```

```python
>>> fluence_example()
(1.234, 5.678, 9.012)
```
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 dttype) – The mask which is to be reduced

Returns

Return type A boolean mask with no all False slices.
WrightTools Documentation, Release 3.3.1

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 -> (s0,s1), (s1,s2), (s2, s3), ...

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 indicies 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 indicies 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 indicies 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 \(u \times \text{np.diag}(s) \times v\), 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: sign(x) * sqrt(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) \rightarrow 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

1.11.1 Plot colormap components

Quickly plot the RGB components of a colormap.

import WrightTools as wt

cmap = wt.artists.colormaps['default']
wt.artists.plot_colormap_components(cmap)

Total running time of the script: 0.224 seconds
1.11.2 Quick 2D Signed

A quick 2D plot of a signed channel.

\[
\text{perovskite}_{\text{TA}}
\]
\[
\tau_{21} = 0.821 \text{fs}
\]
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 1.825 seconds)

1.11.3 Quick 2D

A quick 2D plot.

---

Out:
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: 0.0 minutes 2.081 seconds

1.11.4 Resonance Raman

A Resonance Raman plot.

LDS821_514nm_80mW

Out:
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)
wt.artists.quick1D(data)

Total running time of the script: ( 0 minutes 0.184 seconds)

1.11.5 Quick 1D

A quick 1D plot.

\[ \tilde{\nu}_2 = 1.52e + 03 \text{ cm}^{-1} \]
```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.quick1D(data, 'w1', at={'w2': [1520, 'wn']}, verbose=False)
```

Total running time of the script: ( 0 minutes 1.964 seconds)

## 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("wl", np.linspace(-10, 10, 100))
data.create_channel("sig", 1 / (np.pi * (1 + (data.wl[:]:) - 1) ** 2))
data.transform("wl")
data.gradient("wl")

wt.artists.quick1D(data)
wt.artists.quick1D(data, channel="sig_wl_gradient")
```

**Total running time of the script:** (0 minutes 0.330 seconds)
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.

```python
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 0x7f938122a320>
```
```python
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 0.388 seconds)

### 1.11.8 Tune test

An example of transform on a tune test.

![Graph](image)

Out:

Correction factor applied to d1
Correction factor applied to d2
Data created at /tmp/m3o_04y_.wt5://
axes: ('w1__e__wm', 'wa')
shape: (25, 256)

˓→6/site-packages/WrightTools/artists/_base.py:374: UserWarning: The input
˓→coordinates to pcolor are interpreted as cell centers, but are not monotonically
˓→increasing or decreasing. This may lead to incorrectly calculated cell edges, in
˓→which case, please supply explicit cell edges to pcolor.

(continues on next page)
return super().pcolor(*args, **kwargs)
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Please pass vmin/vmax directly to the norm when creating it.
return super().pcolormesh(*args, **kwargs)

<matplotlib.colorbar.ColorbarBase object at 0x7f9381300668>

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])
ax.pcolor(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")
ax.pcolor(data)
wt.artists.set_ax_labels(xlabel=data.wl.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 0.860 seconds)
1.11.9 DOVE transform

An example of transform on a dataset from a DOVE experiment.

```
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)
```

Out:

```
    Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Please pass vmin/vmax directly to the norm when creating it.
    return super().pcolormesh(*args, **kwargs)
<matplotlib.colorbar.ColorbarBase object at 0x7f937e240780>
```

(continues on next page)
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/6qp2dap7.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
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)
data.signal_mean.symmetric_root(2)  # to amplitude level
data.convert("wn")

fig, gs = wt.artists.create_figure(width="double", cols=[1, 1, "cbar"])

# as taken
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")
1.11.11 Level

Leveling a dataset.

Out:

```
axis w2 converted from nm to eV
axis w1=wm converted from nm to eV
chopped data into 1 piece(s) in ['w1=wm', 'd2']
˓
˓→6/site-packages/WrightTools/data/_data.py:1060: EntireDatasetInMemoryWarning: level
˓warnings.warn("level", category=wt_exceptions.EntireDatasetInMemoryWarning)
channel a10 leveled along axis 2
chopped data into 1 piece(s) in ['w1=wm', 'd2']
˓→6/site-packages/WrightTools/artists/_base.py:414: MatplotlibDeprecationWarning:
˓Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
˓will become an error two minor releases later. Please pass vmin/vmax directly to
˓the norm when creating it.
   return super().pcolormesh(*args, **kwargs)
```

```
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.vlp0pl_MoS2_TrEE_movie
```

(continues on next page)
```python
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)

# leveled
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
wt.artists.set_fig_labels(xlabel=data.w1__e__wm.label, ylabel=data.d2.label)

# colorbar
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 0.716 seconds)

1.11.12 Heal

An example of how heal works.
```

Out:
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 0.319 seconds)
1.11.13 Split

Some examples of how splitting works.

Out:

```
Correction factor applied to d1
Correction factor applied to d2
data created at /tmp/q00bq79d.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
```

(continues on next page)
variable \( w_m \) converted from \( \text{nm} \) to \( \text{wn} \)
split data into 3 pieces along \( <w_2> \):
0 : \(-\infty \) to 7000.00 wn (26, 1)
1 : 7000.00 to 8000.00 wn (32, 1)
2 : 8000.00 to \( \infty \) wn (23, 1)

axis \( w_1+w_2+7000 \) converted from \( \text{wn} \) to \( \text{wn} \)
split data into 3 pieces along \( <w_1+w_2+7000> \):
0 : \(-\infty \) to 20400.00 wn (33, 33)
1 : 20400.00 to 23000.00 wn (81, 81)
2 : 23000.00 to \( \infty \) wn (45, 45)

arr, keepdims=True, axis=tuple(i for i in range(self.ndim) if self.shape[i] == 1)
split data into 3 pieces along \( \text{<strange>} \):
0 : \(-\infty \) to 0.20 None (81, 81)
1 : 0.20 to 0.40 None (71, 76)
2 : 0.40 to \( \infty \) None (54, 46)

```python
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 2.908 seconds)
1.11.14 Colormaps

Different colormaps.
Out:

warnings.warn("level", category=wt_exceptions.EntireDatasetInMemoryWarning)
channel ai0 leveled along axis 2
axis w2 converted from nm to eV
axis w1=wm converted from nm to eV
chopped data into 1 piece(s) in ['w1=wm', 'w2']
Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
will become an error two minor releases later. Please pass vmin/vmax directly to
the norm when creating it.
return super().pcolormesh(*args, **kwargs)

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)
)

Total running time of the script: 0 minutes 1.220 seconds

## 1.11.15 Map-Variable

An example of how map-variable works.

```
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
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 0.399 seconds

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/_7e_ko0u.wt5::/
    axes: ('d1', 'd2')
    shape: (101, 101)
axis d1 converted from ps to fs
axis d2 converted from ps to fs
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/ho8b8ivv.wt5: /
   axes: ('d1', 'd2')
   shape: (101, 101)
axis d1 converted from ps to fs
axis d2 converted from ps to fs
   Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and
   will become an error two minor releases later. Please pass vmin/vmax directly to
   the norm when creating it.
   return super().pcolormesh(*args, **kwargs)
<matplotlib.colorbar.ColorbarBase object at 0x7f937d70c5f8>

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)
wt.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)
labels = ['II', 'I', 'III', 'V', 'VI', '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 1.799 seconds)

### 1.11.17 Join

Some examples of how joining works.
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, :])
```python
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 1.560 seconds

### 1.11.18 Lineshapes

Common lineshapes included in kit
import matplotlib.pyplot as plt
import WrightTools as wt
import numpy as np

1.11. Gallery
# initiate figure
fig, gs = wt.artists.create_figure(nrows=2, default_aspect=0.6)
axs = [plt.subplot(gs[i]) for i in range(2)]

# initial parameters
x = np.linspace(-2, 2, 1001)
x0 = 0
FWHM = 1
G = 0.5

# plot all of the real versions
ax = axs[0]
y = wt.kit.gaussian(x, x0, FWHM, norm="height")
ax.plot(x, y, label="Gaussian, height")
y = wt.kit.gaussian(x, x0, FWHM, norm="area")
ax.plot(x, y, label="Gaussian, area")
y = wt.kit.lorentzian_real(x, x0, G, norm="height")
ax.plot(x, y, label="Lorentzian, height")
y = wt.kit.lorentzian_real(x, x0, G, norm="area")
ax.plot(x, y, label="Lorentzian, area")
y = wt.kit.voigt(x, x0, FWHM, G)
ax.plot(x, y / y.max(), label="Voigt")

# plot the complex variants
ax = axs[1]
y = wt.kit.lorentzian_complex(x, x0, G, norm="height_imag")
ax.plot(x, y.real, label="Re[L], height_imag")
ax.plot(x, y.imag, label="Im[L], height_imag")
y = wt.kit.lorentzian_complex(x, x0, G, norm="area_int")
ax.plot(x, y.real, label="Re[L], area_int")
ax.plot(x, y.imag, label="Im[L], area_int")

# finish
for ax in axs:
    ax.legend(
        bbox_to_anchor=(1.04, 0.5),
        loc="center left",
        borderaxespad=0,
        title="Lineshape, normalization",
    )
    ax.grid()
    ax.set_xlim(-2, 2)
wt.artists.set_fig_labels(fig, "X", "Y")

Total running time of the script: (0 minutes 0.270 seconds)
1.11.19 Sideplots and Moments

An example showing how to use moments and sideplots.

Out:

```
warnings.warn("moment", category=wt_exceptions.EntireDatasetInMemoryWarning)
/home/docs/checkouts/readthedocs.org/user_builds/wrighttools/envs/stable/lib/python3.6/site-packages/WrightTools/artists/_base.py:414: MatplotlibDeprecationWarning: Passing parameters norm and vmin/vmax simultaneously is deprecated since 3.3 and will become an error two minor releases later. Please pass vmin/vmax directly to the norm when creating it.
    return super().pcolormesh(*args, **kwargs)
<matplotlib.colorbar.ColorbarBase object at 0x7f937e23d128>
```
import WrightTools as wt
import numpy as np
import matplotlib.pyplot as plt

def S(x):
    return (1 + np.exp(-1 * x)) ** -1

# instantiate data object
d1 = np.linspace(-1, 3, 101)[:, None]
d2 = np.linspace(-1, 3, 102)[None, :]
arr = S(10 * (d1 + 0.5)) * S(10 * d2) * S(-2 * d1) + 0.1 * S(5 * d1) * S(5 * (d2 - 1))
d = wt.Data(name="test")
d.create_variable("d1", values=d1, units="ps", label="1")
d.create_variable("d2", values=d2, units="ps", label="2")
d.create_channel("z", values=arr)
d.transform("d1", "d2")

# calculate moments
for d_moment in [d.moment(axis="d1", channel=0, moment=0), d.moment(axis="d2", channel=0, moment=0)]:
    d_moment.normalize()

# create figure
fig, gs = wt.artists.create_figure(cols=[1, "cbar"], pad=0.1, ymin=-0.1, ymax=1.1)
ax = plt.subplot(gs[0])
axcorrx = ax.add_sideplot(along="x", pad=0.1, ymin=-0.1, ymax=1.1)
axcorry = ax.add_sideplot(along="y", pad=0.1, ymin=-0.1, ymax=1.1)

# plot data
ax.pcolor(d, autolabel="both")

# plot integral moments in sideplot
axcorrx.plot(d, channel="z_d2_moment_0", color="k", linewidth=3)
axcorry.plot(d.d2.points, d.d1.points, color="k", linewidth=3)

# center of mass
ax.plot(d.d1.points, d.d2.points, label="COM along x")
ax.plot(d.d2.points, d.d1.points, label="COM along y")

# grids
for ax in [ax, axcorrx, axcorry]:
    ax.grid()

# plot colorbar
plt.colorbar()
wt.artists.plot_colorbar(cax=cax, label="amplitude")
1.11.20 Fill types

Different ways to plot 2D data.

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import WrightTools as wt
```

(continues on next page)
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"])

# 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])
avx.pcolor(data, cmap=cmap)
avx.set_title("pcolor", fontsize=20)
decorate(ax)

# tripcolor
xi = data.d1.points
yi = data.d2.points
zi = data.channels[0][:].T
ax = plt.subplot(gs[1, 0])
points = [xi, yi]
x, y = tuple(np.meshgrid(*points, indexing="ij"))
avx.tripcolor(x.flatten(), y.flatten(), zi.T.flatten(), cmap=cmap, vmin=0, vmax=1)
decorate(ax)

# plot_delaunay_edges
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(132 Chapter 1. Contents
[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)

# 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]
w.t.artists.set_fig_labels(xlabel=data.d1.label, ylabel=data.d2.label, xticks=ticks,
→ yticks=ticks)

# colorbar
cax = plt.subplot(gs[:, -1])
w.t.artists.plot_colorbar(cax=cax, label="amplitude")

Total running time of the script: ( 0 minutes 4.939 seconds)

1.11.21 Custom Figure

Example of custom figure layout, beautification, and saving.
channel ai0 leveled along axis 2

smoothed data
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 = [...]

(continues on next page)
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)]
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)
1.11.22 WMELs: TRIVE off diagonal

Draw WMELs for TRIVE off diagonal.

```python
import matplotlib.pyplot as plt
import WrightTools as W

artist = W.Artist(
    size=[6, 2], energies=[0., 0.43, 0.57, 1.], state_names=["g", "a", "b", "a+b"]
)

artist.label_rows([r"\text{oil}", r"\text{beta}", r"\text{gamma}"])
artist.label_columns(["I", "II", "III", "IV", "V", "VI"])

# pwl 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")

# pwl 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, [2, 0], "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)
# pw6 beta
artist.add_arrow([5, 1], 0, [0, 2], "ket", "2")
artist.add_arrow([5, 1], 1, [0, 2], "bra", "-2")
artist.add_arrow([5, 1], 2, [2, 3], "ket", "1")
artist.add_arrow([5, 1], 3, [3, 2], "out")

artist.plot()
plt.show()

Total running time of the script: ( 0 minutes 0.476 seconds)

1.11.23 WMELs: TRIVE population transfer

Draw WMELs for TRIVE population transfer.
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'$\text{\alpha}$', r'$\text{\beta}$', r'$\text{\gamma}$'})
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''
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''
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''
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''
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''
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, 1], 'bra')
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''
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''
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''

(continued on next page)
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()

**Total running time of the script:** (0 minutes 0.525 seconds)

### 1.11.24 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"\text{\alpha}", r"\text{\beta}", r"\text{\gamma}"])

# Diagrams

# (continued 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")
artist.add_arrow([3, 0], 3, [1, 0], "out")

# pw4 beta
artist.add_arrow([3, 1], 0, [0, 1], "ket", "2'")
artist.addarrow([3, 1], 1, [1, 2], "ket", "1")
artist.addarrow([3, 1], 2, [0, 1], "bra", "-2")
artist.addarrow([3, 1], 3, [2, 1], "out")

# pw5 alpha
artist.addarrow([4, 0], 0, [0, 1], "bra", "-2")
artist.addarrow([4, 0], 1, [1, 0], "bra", "2'")
artist.addarrow([4, 0], 2, [0, 1], "ket", "1")
artist.addarrow([4, 0], 3, [1, 0], "out")

# pw5 beta
artist.addarrow([4, 1], 0, [0, 1], "bra", "-2")
artist.addarrow([4, 1], 1, [0, 1], "ket", "2'")
artist.addarrow([4, 1], 2, [1, 2], "ket", "1")
artist.addarrow([4, 1], 3, [2, 1], "out")

# pw5 gamma
artist.addarrow([4, 2], 0, [0, 1], "bra", "-2")
artist.addarrow([4, 2], 1, [0, 1], "ket", "2'")
artist.addarrow([4, 2], 2, [1, 0], "bra", "1")
artist.addarrow([4, 2], 3, [1, 0], "out")

# pw6 alpha
artist.addarrow([5, 0], 0, [0, 1], "ket", "2'")
artist.addarrow([5, 0], 1, [1, 0], "ket", "-2")
artist.addarrow([5, 0], 2, [0, 1], "ket", "1")
artist.addarrow([5, 0], 3, [1, 0], "out")

# pw6 beta
artist.addarrow([5, 1], 0, [0, 1], "ket", "2'")
artist.addarrow([5, 1], 1, [0, 1], "bra", "-2")
artist.addarrow([5, 1], 2, [1, 2], "ket", "1")
artist.addarrow([5, 1], 3, [2, 1], "out")

# pw6 beta
artist.addarrow([5, 2], 0, [0, 1], "ket", "2'")
artist.addarrow([5, 2], 1, [0, 1], "bra", "-2")
artist.addarrow([5, 2], 2, [1, 0], "bra", "1")
artist.addarrow([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:

```latex
@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.
INDEX

- genindex
- modindex
- search
W
WrightTools.artists, 36
WrightTools.collection, 62
WrightTools.data, 66
WrightTools.diagrams, 80
WrightTools.diagrams.delay, 83
WrightTools.diagrams.WMEL, 80
WrightTools.exceptions, 83
WrightTools.kit, 85
WrightTools.units, 98
Symbols

__init__() (WrightTools.artists.Axes method), 37
__init__() (WrightTools.artists.Figure method), 46
__init__() (WrightTools.artists.GridSpec method), 50
__init__() (WrightTools.collection.Collection method), 63
__init__() (WrightTools.data.Axis method), 69
__init__() (WrightTools.data.Channel method), 69
__init__() (WrightTools.data.Constant method), 71
__init__() (WrightTools.data.Data method), 66
__init__() (WrightTools.data.Variable method), 72
__init__() (WrightTools.diagrams.WMEL.Artist method), 80
__init__() (WrightTools.diagrams.WMEL.Subplot method), 82
__init__() (WrightTools.kit.INI method), 86
__init__() (WrightTools.kit.Spline method), 87
__init__() (WrightTools.kit.TimeStamp method), 87
__init__() (WrightTools.kit.Timer method), 89
corner_text() (in module WrightTools.artists), 52
create_figure() (in module WrightTools.artists), 53

D
Data (class in WrightTools.data), 66
date (WrightTools.kit.TimeStamp attribute), 88
diagonal_line() (in module WrightTools.artists), 54
diff() (in module WrightTools.kit), 89
DimensionalityError, 84
discover_dimensions() (in module WrightTools.kit), 90

E
enforce_mask_shape() (in module WrightTools.kit), 90
EntireDatasetInMemoryWarning, 84

F
fft() (in module WrightTools.kit), 90
Figure (class in WrightTools.artists), 46
FileExistsError, 84
flatten_list() (in module WrightTools.artists), 52
fluence() (in module WrightTools.kit), 91
from_Aramis() (in module WrightTools.data), 77
from_BrunoldrRaman() (in module WrightTools.data), 75
from_Cary() (in module WrightTools.collection), 64
from_COLORS() (in module WrightTools.data), 75
from_directory() (in module WrightTools.collection), 64
from_JASCO() (in module WrightTools.data), 76
from_KENT() (in module WrightTools.data), 76
from_ocean_optics() (in module WrightTools.data), 77
from_PyCMDS() (in module WrightTools.data), 77
from_shimadzu() (in module WrightTools.data), 78
from_Solis() (in module WrightTools.data), 78
from_spcm() (in module WrightTools.data), 79
from_Tensor27() (in module WrightTools.data), 79
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_color_cycle</td>
<td>WrightTools.artists</td>
<td>55</td>
</tr>
<tr>
<td>get_index</td>
<td>WrightTools.kit</td>
<td>92</td>
</tr>
<tr>
<td>get_path_matching</td>
<td>WrightTools.kit</td>
<td>92</td>
</tr>
<tr>
<td>get_scaled_bounds</td>
<td>WrightTools.artists</td>
<td>55</td>
</tr>
<tr>
<td>get_symbol</td>
<td>WrightTools.units</td>
<td>98</td>
</tr>
<tr>
<td>get_valid_conversions</td>
<td>WrightTools.units</td>
<td>99</td>
</tr>
<tr>
<td>glob_handler</td>
<td>WrightTools.kit</td>
<td>92</td>
</tr>
<tr>
<td>grayify_cmap</td>
<td>WrightTools.artists</td>
<td>55</td>
</tr>
<tr>
<td>GridSpec</td>
<td>class in WrightTools.artists</td>
<td>50</td>
</tr>
<tr>
<td>hms</td>
<td>WrightTools.kit.TimeStamp attribute</td>
<td>88</td>
</tr>
<tr>
<td>human</td>
<td>WrightTools.kit.TimeStamp attribute</td>
<td>88</td>
</tr>
<tr>
<td>INI</td>
<td>class in WrightTools.kit</td>
<td>86</td>
</tr>
<tr>
<td>interact2D</td>
<td>WrightTools.artists</td>
<td>55</td>
</tr>
<tr>
<td>intersperse</td>
<td>WrightTools.kit</td>
<td>94</td>
</tr>
<tr>
<td>is_valid_conversion</td>
<td>WrightTools.units</td>
<td>99</td>
</tr>
<tr>
<td>join</td>
<td>WrightTools.data</td>
<td>74</td>
</tr>
<tr>
<td>joint_shape</td>
<td>WrightTools.kit</td>
<td>93</td>
</tr>
<tr>
<td>kind</td>
<td>WrightTools.units</td>
<td>99</td>
</tr>
<tr>
<td>label_columns</td>
<td>WrightTools.diagrams.WMEL.Artist method</td>
<td>81</td>
</tr>
<tr>
<td>label_rows</td>
<td>WrightTools.diagrams.WMEL.Artist method</td>
<td>81</td>
</tr>
<tr>
<td>label_sectors</td>
<td>WrightTools.diagrams.delay</td>
<td>83</td>
</tr>
<tr>
<td>leastsqfitter</td>
<td>WrightTools.kit</td>
<td>95</td>
</tr>
<tr>
<td>legacy</td>
<td>WrightTools.kit.TimeStamp attribute</td>
<td>88</td>
</tr>
<tr>
<td>mask_reduce</td>
<td>WrightTools.kit</td>
<td>93</td>
</tr>
<tr>
<td>module</td>
<td>WrightTools.collection, data, diagrams, delay</td>
<td>36, 62, 66, 80, 83</td>
</tr>
<tr>
<td>savefig</td>
<td>WrightTools.artists</td>
<td>60</td>
</tr>
<tr>
<td>set_ax_labels</td>
<td>WrightTools.artists</td>
<td>60</td>
</tr>
<tr>
<td>set_ax_spines</td>
<td>WrightTools.artists</td>
<td>61</td>
</tr>
<tr>
<td>set_fig_labels</td>
<td>WrightTools.artists</td>
<td>61</td>
</tr>
<tr>
<td>share_nans</td>
<td>WrightTools.kit</td>
<td>95</td>
</tr>
<tr>
<td>smooth_1D</td>
<td>WrightTools.kit</td>
<td>95</td>
</tr>
<tr>
<td>Spline</td>
<td>class in WrightTools.kit</td>
<td>87</td>
</tr>
<tr>
<td>stitch_to_animation</td>
<td>WrightTools.artists</td>
<td>62</td>
</tr>
</tbody>
</table>

**Index**
string2identifier() (in module WrightTools.kit), 95
Subplot (class in WrightTools.diagrams.WMEL), 82
subplots_adjust() (in module WrightTools.artists), 62
svd() (in module WrightTools.kit), 96
symmetric_sqrt() (in module WrightTools.kit), 96

T
Timer (class in WrightTools.kit), 89
TimeStamp (class in WrightTools.kit), 87
timestamp_from_RFC3339() (in module WrightTools.kit), 96
TypeError, 84

U
unique() (in module WrightTools.kit), 97
UnitsError, 85
unix (WrightTools.kit.TimeStamp attribute), 87

V
valid_index() (in module WrightTools.kit), 97
ValueError, 85
Variable (class in WrightTools.data), 72
VisibleDeprecationWarning, 85

W
WrightTools.artists module, 36
WrightTools.collection module, 62
WrightTools.data module, 66
WrightTools.diagrams module, 80
WrightTools.diagrams.delay module, 83
WrightTools.diagrams.WMEL module, 80
WrightTools.exceptions module, 83
WrightTools.kit module, 85
WrightTools.units module, 98
WrightToolsException, 85
WrightToolsWarning, 85
WrongFileTypeWarning, 85

Z
zoom2D() (in module WrightTools.kit), 97