iris

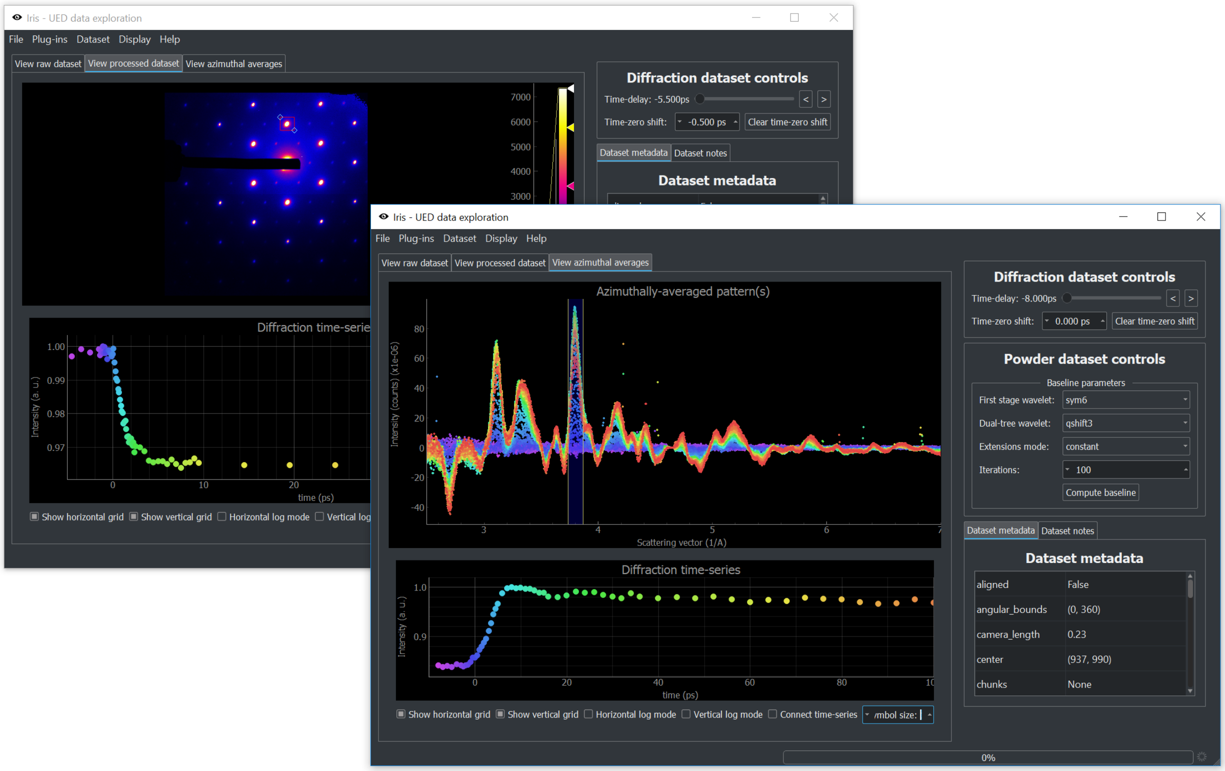
Release 5.2.0

Nov 22, 2019

Contents

iris is both a library for interacting with ultrafast electron diffraction data, as well as a GUI frontend to interactively explore this data.

The code presented herein has been in use at some point by the Siwick research group.



Time-delay: -5.500ps

Time-zero shift: -0.500 ps

Time-delay: -8.000ps

Time-zero shift: 0.000 ps

First stage wavelet: sym6

Dual-tree wavelet: qshift3

Extensions mode: constant

Iterations: 100

aligned	False
angular_bounds	(0, 360)
camera_length	0.23
center	(937, 990)
chunks	None

CHAPTER 1

Links

- Source code
- Issues

CHAPTER 2

General Documentation

2.1 Installation

2.1.1 Standalone Installation

Starting with *iris* 5.1.0, **standalone Windows installers and executables are available**. You can find them on the *GitHub release page <https://github.com/LaurentRDC/iris-ued/releases/latest/>*.

The standalone installers and executables make the installation of *iris* completely separate from any other Python installation. This method should be preferred, unless Python scripting using the *iris* library is required.

2.1.2 Installing the Python Package

If you want to script using *iris* data structures and algorithms, you need to install the *iris-ued* package.

Note: Users are strongly recommended to manage these dependencies with the excellent Intel Distribution for Python which provides easy access to all of the above dependencies and more.

`iris` is available on PyPI as **iris-ued**:

```
python -m pip install iris-ued
```

`iris` is also available on the conda-forge channel:

```
conda config --add channels conda-forge
conda install iris-ued
```

You can install the latest developer version of `iris` by cloning the git repository:

```
git clone https://github.com/LaurentRDC/iris-ued.git
```

. . . then installing the package with:

```
cd iris-ued
python setup.py install
```

In Python code, `iris` can be imported as follows

```
import iris
```

2.1.3 Test data

Test reduced datasets are made available by the Siwick research group. The data can be accessed on the public data repository

2.1.4 Testing

If you want to check that all the tests are running correctly with your Python configuration, type:

```
python setup.py test
```

2.2 Using iris: typical workflow

2.2.1 Before you start

You might want to download test datasets before you start to play around. Test reduced datasets are made available by the Siwick research group. The data can be accessed on the public data repository

2.2.2 Startup

To start the GUI from the command line:

```
> python -m iris
```

Note that the command-line interface has some useful options:

```
> python -m iris --help
usage: iris [-h] [-v] {open,docs} ...

Iris is both a library for interacting with ultrafast electron diffraction
data, as well as a GUI frontend for interactively exploring this data. Below
are some helpful commands.

optional arguments:
-h, --help         show this help message and exit
-v, --version      show program's version number and exit

Subcommands:
{open,docs}  Available sub-commands
    open            Dataset to open with iris start-up.
    docs            Open online documentation in your default web browser.
```

(continues on next page)

(continued from previous page)

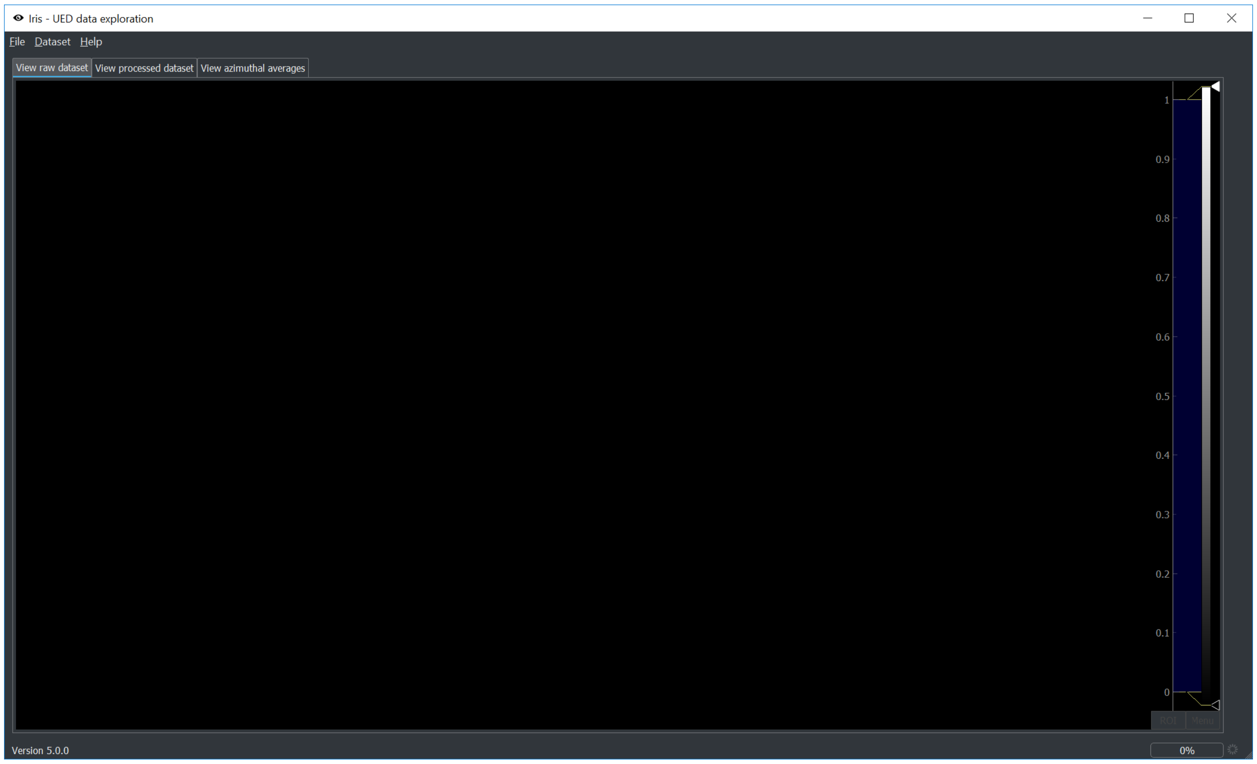
```
Running this command without any parameters will launch the graphical user
interface. Documentation is available here: https://iris-ued.readthedocs.io/
```

Most importantly, you can programatically start the GUI with opening a dataset:

```
> python -m iris open --reduced ~/dataset.hdf5
```

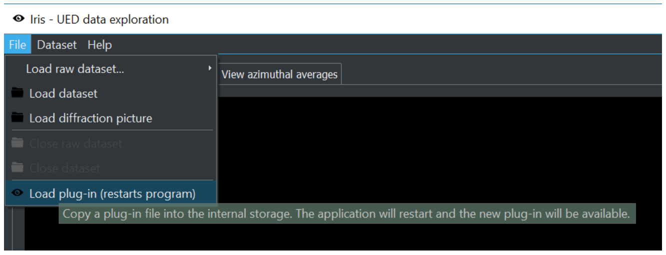
The path can lead to a reduced HDF5 file (flag *–reduced*) or a raw dataset (flag *–raw*). In case of a raw dataset, the dataset format will be guessed with the same rules as `iris.open_raw()`.

The first blank screen is shown below.



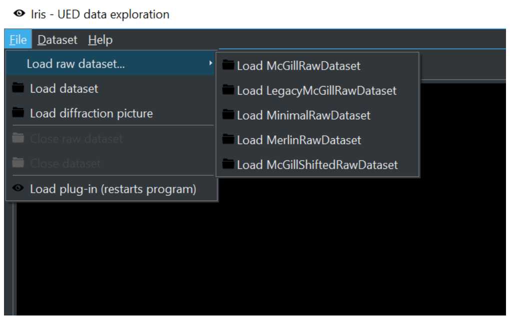
2.2.3 Loading raw data

The file menu can be used to load raw data. Depending on the installed plugins, options will be available. To install a new plug-in, use the following option:

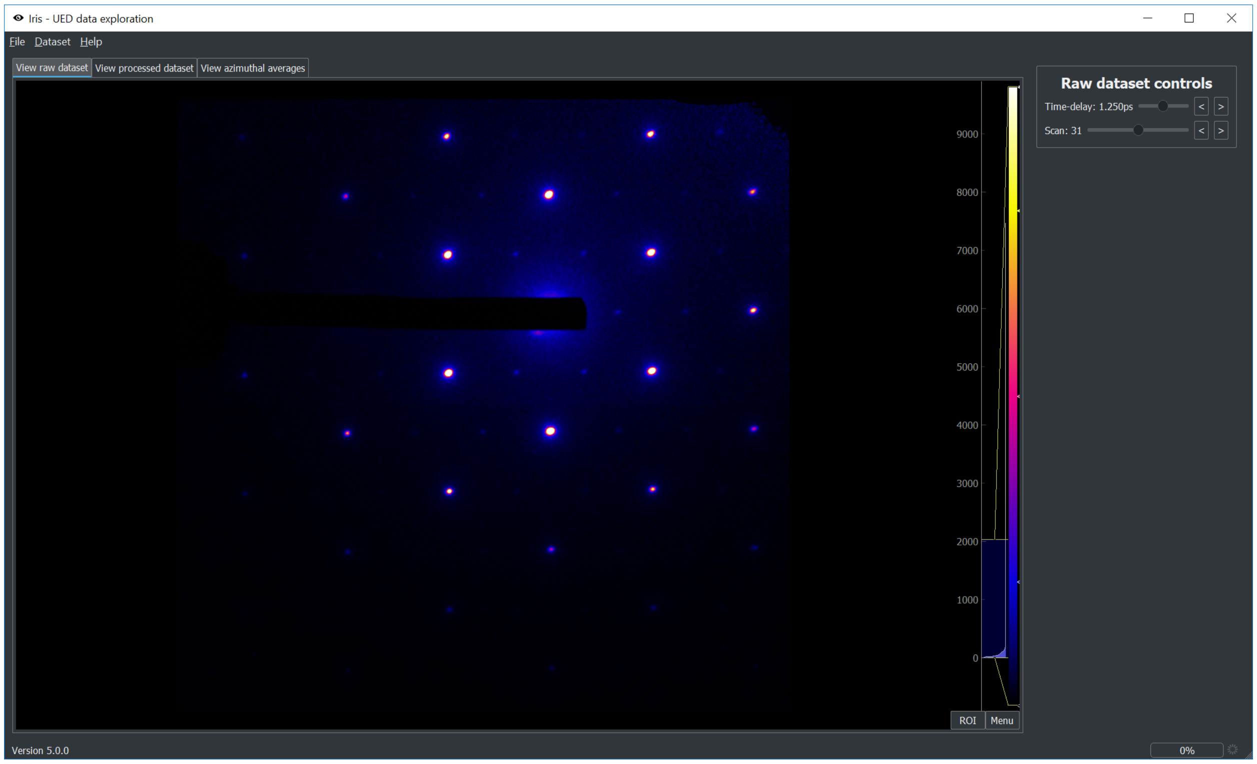


Copy a plug-in file into the internal storage. The application will restart and the new plug-in will be available.

You'll be able to select a plug-in file which will be copied to the plug-in directory. The plugin can be used immediately. Once a plug-in is installed, a new raw data format will appear.

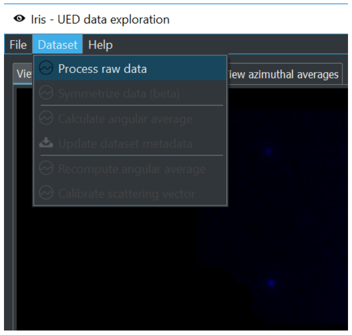


Here is an example of loaded raw data: Raw data controls are available to the right.

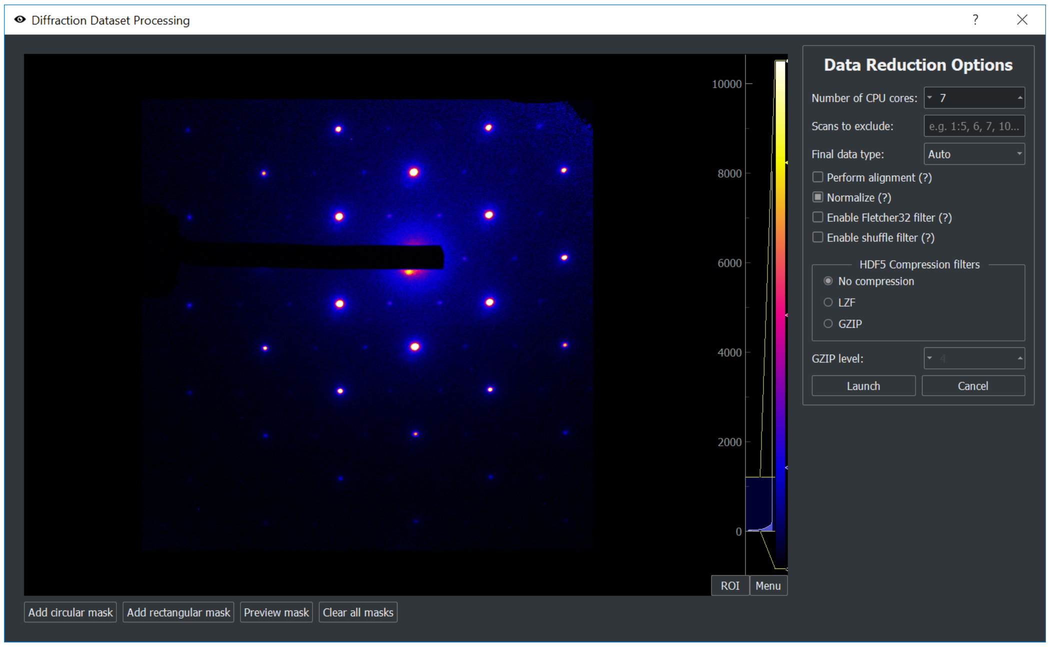


2.2.4 Data reduction

Once raw data is loaded, the following option becomes available:


Iris - UED data exploration
File Dataset Help
Process raw data
Symmetrize data (beta)
Calculate angular average
Update dataset metadata
Recompute angular average
Calibrate scattering vector

This opens the data reduction dialog.



Data Reduction Options

Number of CPU cores: 7

Scans to exclude: e.g. 1:5, 6, 7, 10...

Final data type: Auto

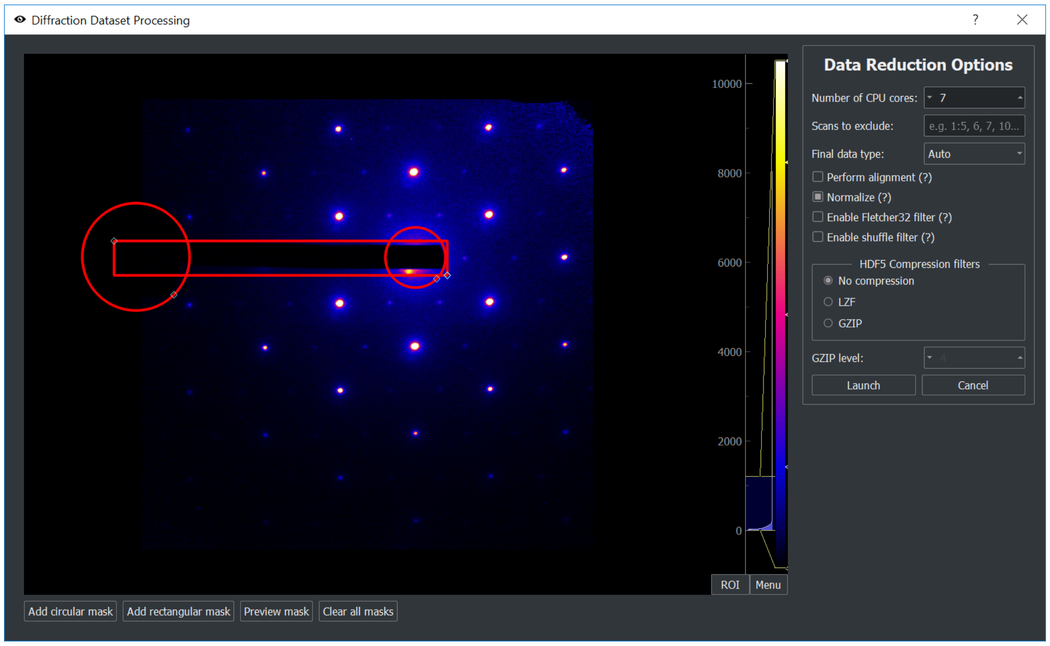
- [ ] Perform alignment (?)
- [x] Normalize (?)
- [ ] Enable Fletcher32 filter (?)
- [ ] Enable shuffle filter (?)

HDF5 Compression filters

[✓ No compression] LZF GZIP

GZIP level: 4

Parts of the data can be masked. To add a mask, use the controls on the top of the dialog. Masks can be moved and resized. Note that all images will be masked, so this is best for beam blocks, known hot pixels, etc.



Data Reduction Options

Number of CPU cores: 7

Scans to exclude: e.g. 1:5, 6, 7, 10...

Final data type: Auto

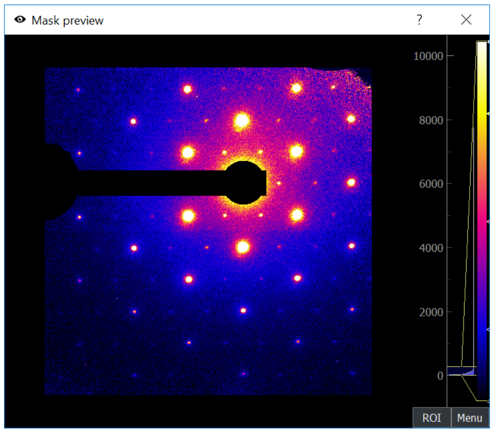
- [ ] Perform alignment (?)
- [x] Normalize (?)
- [ ] Enable Fletcher32 filter (?)
- [ ] Enable shuffle filter (?)

HDF5 Compression filters

[✓ No compression] LZF GZIP

GZIP level: 4

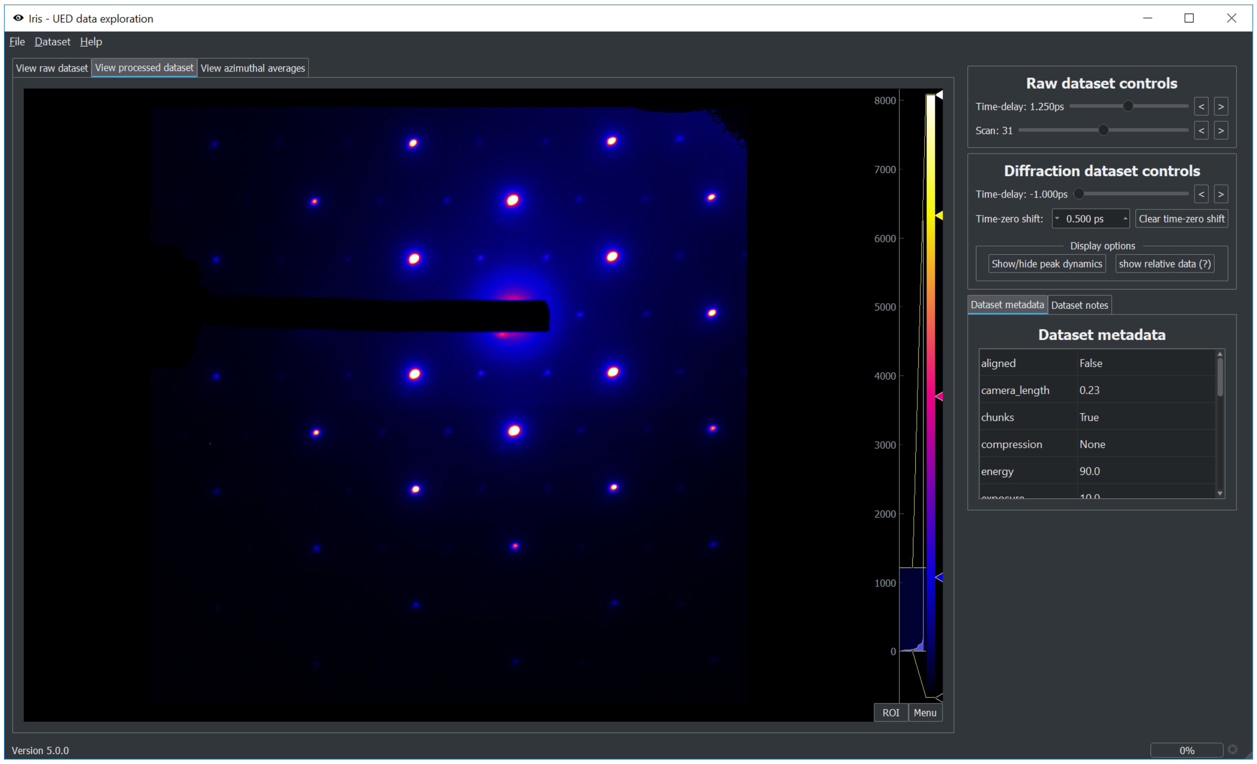
A preview of the mask can be generated:



Once you are satisfied with the processing parameters, the ‘Launch processing’ button will open a file dialog so that you can choose where to save the reduced HDF5 file. Processing might take a few minutes.

2.2.5 Data exploration

Once processing is complete, the resulting diffraction dataset will be loaded. New controls will be available.



Raw dataset controls

Time-delay: 1.250ps

Scan: 31

Diffraction dataset controls

Time-delay: -1.000ps

Time-zero shift: 0.500 ps

Clear time-zero shift

Display options

Show/hide peak dynamics

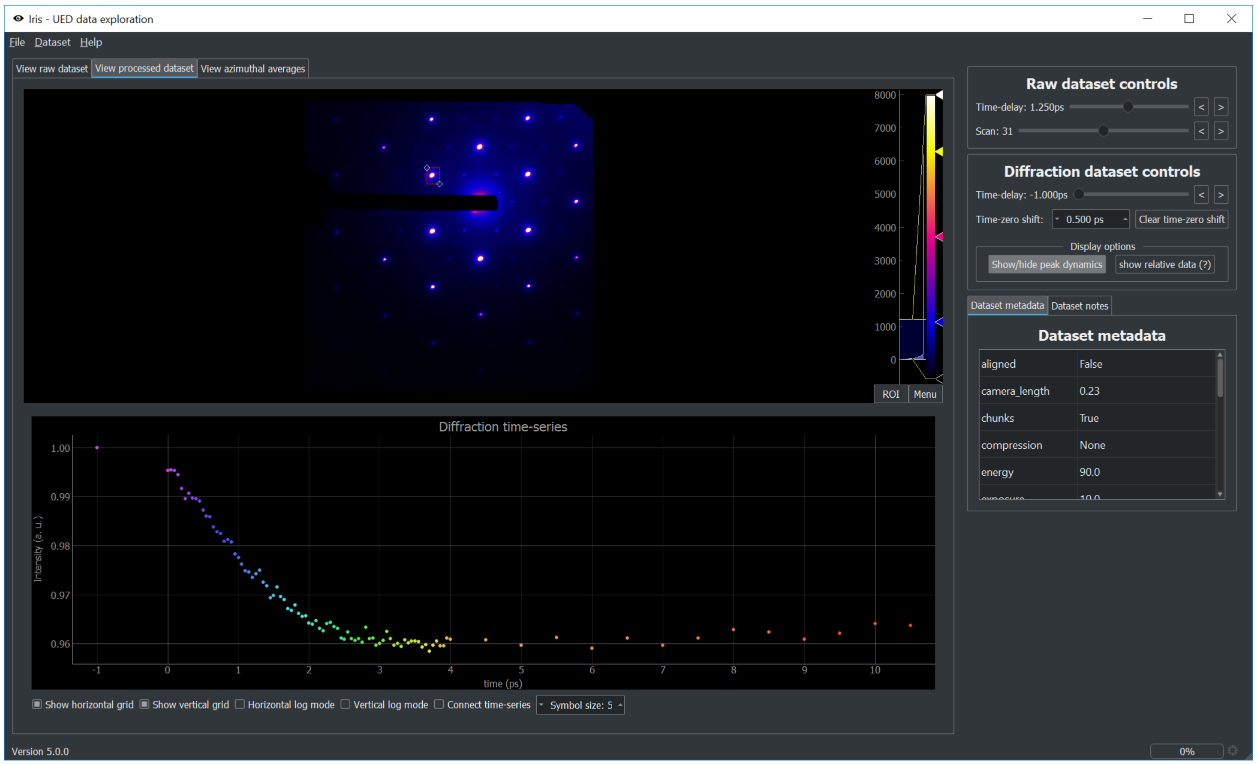
show relative data (?)

Dataset metadata | Dataset notes

Dataset metadata

aligned	False
camera_length	0.23
chunks	True
compression	None
energy	90.0

The ‘Show/hide peak dynamics’ button can be toggled. Doing so allows for the exploration of the time-evolution of the data.



Time-delay: 1.250ps

Scan: 31

Time-delay: -1.000ps

Time-zero shift: 0.500 ps

Symbol size: 5

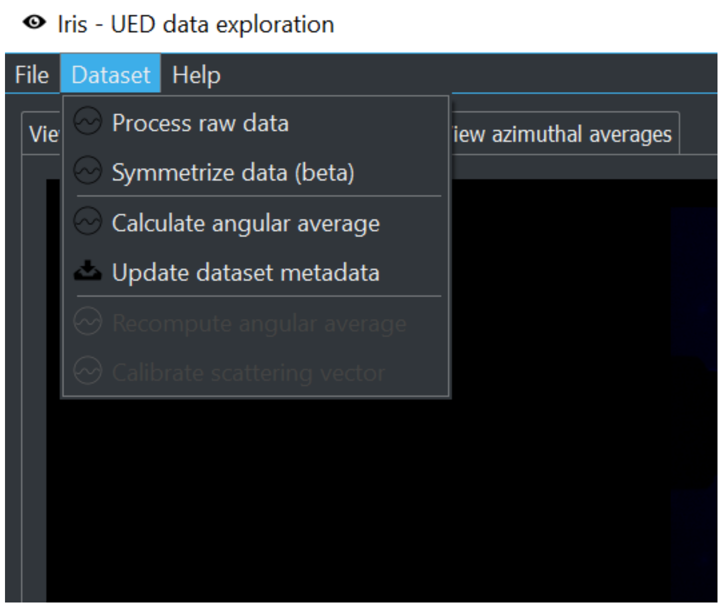
Dataset metadata

aligned	False
camera_length	0.23
chunks	True
compression	None
energy	90.0

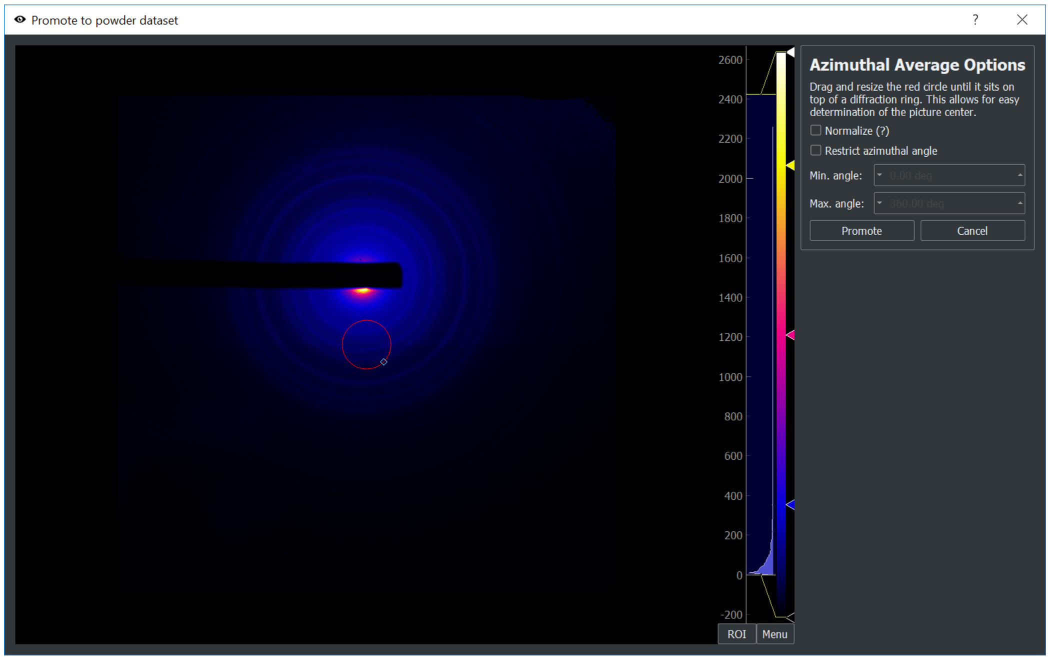
Version 5.0.0

0%

When a diffraction dataset is loaded, new options become available.


Iris - UED data exploration
File
Dataset
Help
Vie
iew azimuthal averages
Process raw data
Symmetrize data (beta)
Calculate angular average
Update dataset metadata
Recompute angular average
Calibrate scattering vector

One of these options, ‘Compute angular averages’, is best suited for polycrystalline diffraction. It opens the following dialog:



Azimuthal Average Options

Drag and resize the red circle until it sits on top of a diffraction ring. This allows for easy determination of the picture center.

☐ Normalize (?)

☐ Restrict azimuthal angle

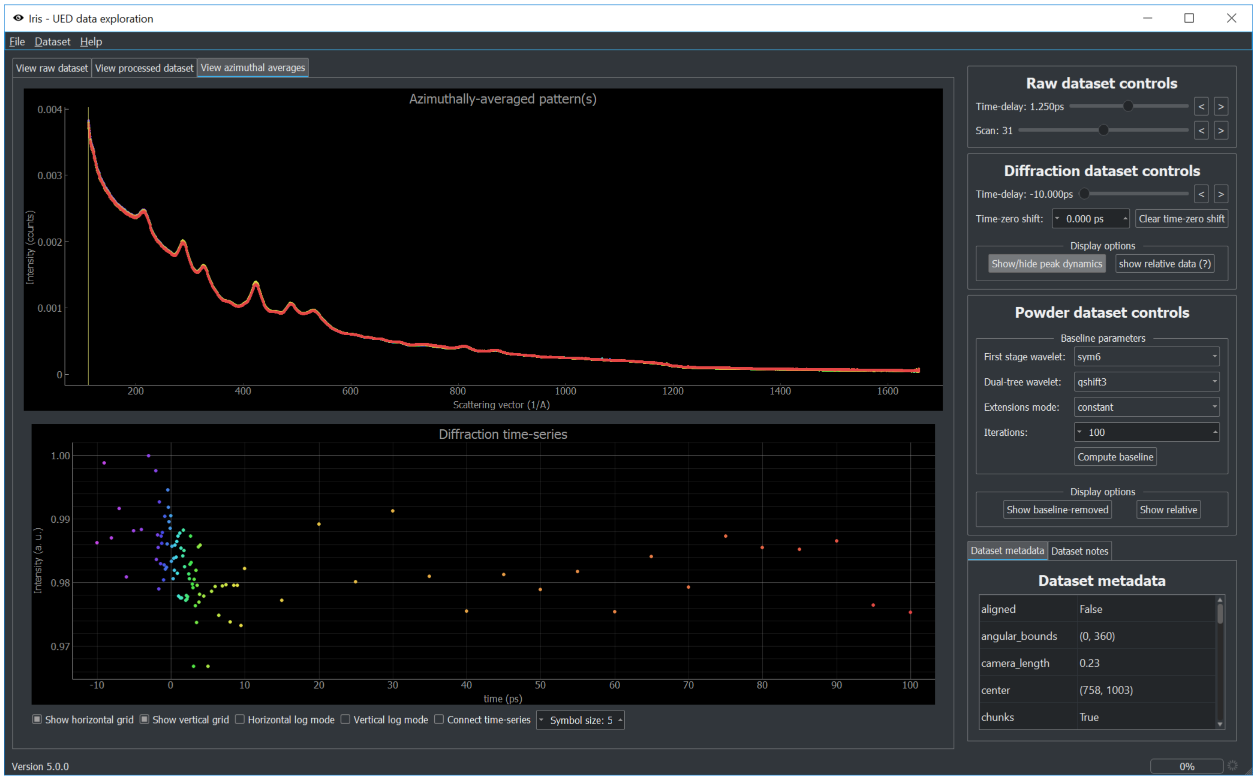
Min. angle: 0.00 deg

Max. angle: 360.00 deg

Drag and resize the red circle so it coincides with a diffraction ring. This will allow for the determination of the diffraction center. The averaging will happen after clicking ‘Promote’. This might take a few minutes.

2.2.6 Polycrystalline data exploration

After the azimuthal averages have been computed, a new section of the GUI will be made available, with additional controls.



Raw dataset controls

Time-delay: 1.250ps

Scan: 31

Diffraction dataset controls

Time-delay: -10.000ps

Time-zero shift: 0.000 ps

Display options

Powder dataset controls

Baseline parameters

First stage wavelet: sym6

Dual-tree wavelet: qshift3

Extensions mode: constant

Iterations: 100

Display options

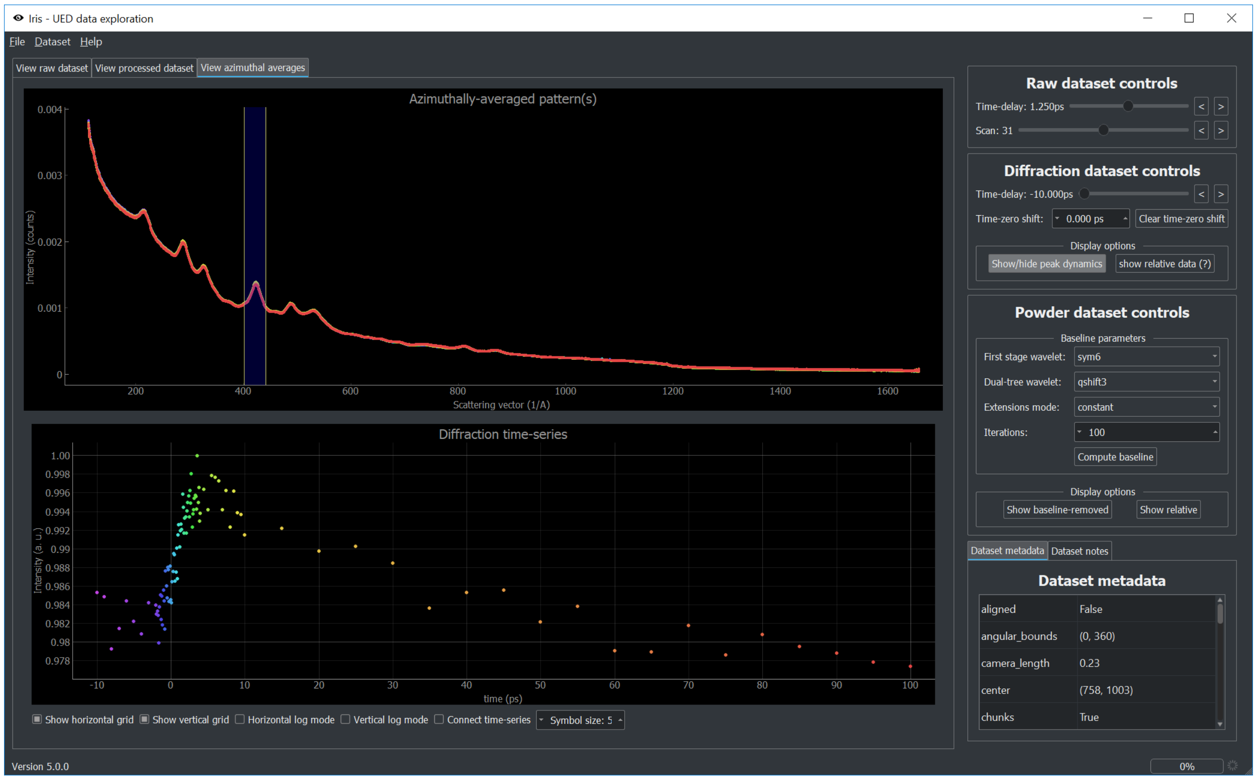
☒ Show horizontal grid ☒ Show vertical grid ☐ Horizontal log mode ☐ Vertical log mode ☐ Connect time-series Symbol size: 5

Dataset metadata

aligned	False
angular_bounds	(0, 360)
camera_length	0.23
center	(758, 1003)
chunks	True

Version 5.0.0

The top screen shows the superposition of all radial profiles. Dragging the yellow lines allows for exploration of time-evolution on the bottom screen. Note that the trace colors on the top are associated with the time-points and colors of the bottom image.



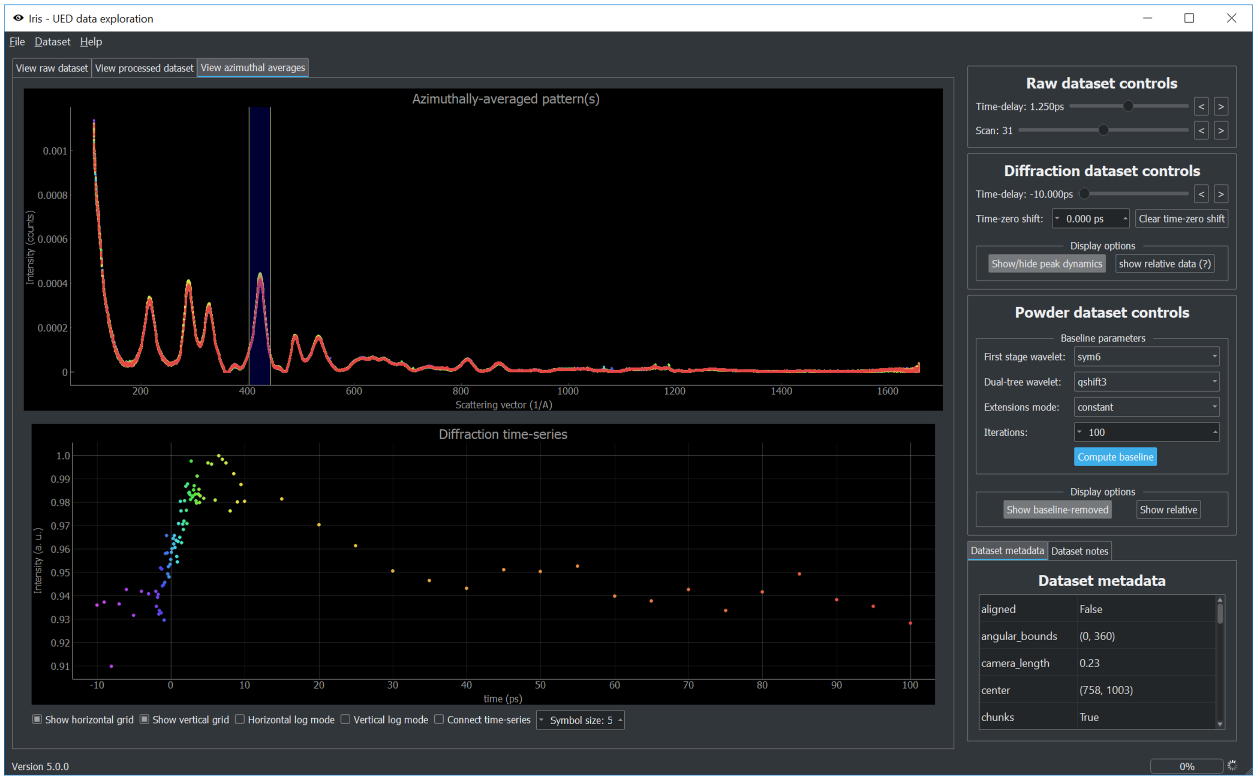
Time-delay: 1.250ps
Scan: 31
Time-delay: -10.000ps
Time-zero shift: 0.000 ps
First stage wavelet: sym6
Dual-tree wavelet: qshift3
Extensions mode: constant
Iterations: 100
Symbol size: 5

aligned	False
angular_bounds	(0, 360)
camera_length	0.23
center	(758, 1003)
chunks	True

Version 5.0.0

0%

The baseline can be removed using the controls on the right. You can play with the baseline parameters and compute a baseline many times without any problems.



Raw dataset controls

Time-delay: 1.250ps

Scan: 31

Diffraction dataset controls

Time-delay: -10.000ps

Time-zero shift: 0.000 ps

Powder dataset controls

Baseline parameters

First stage wavelet:	sym6
Dual-tree wavelet:	qshift3
Extensions mode:	constant
Iterations:	100

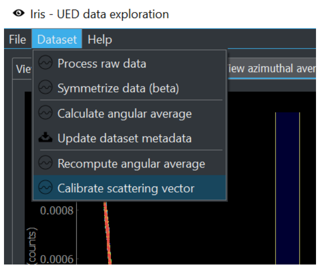
☐ Show horizontal grid ☐ Show vertical grid ☐ Horizontal log mode ☐ Vertical log mode ☐ Connect time-series Symbol size: 5

Dataset metadata

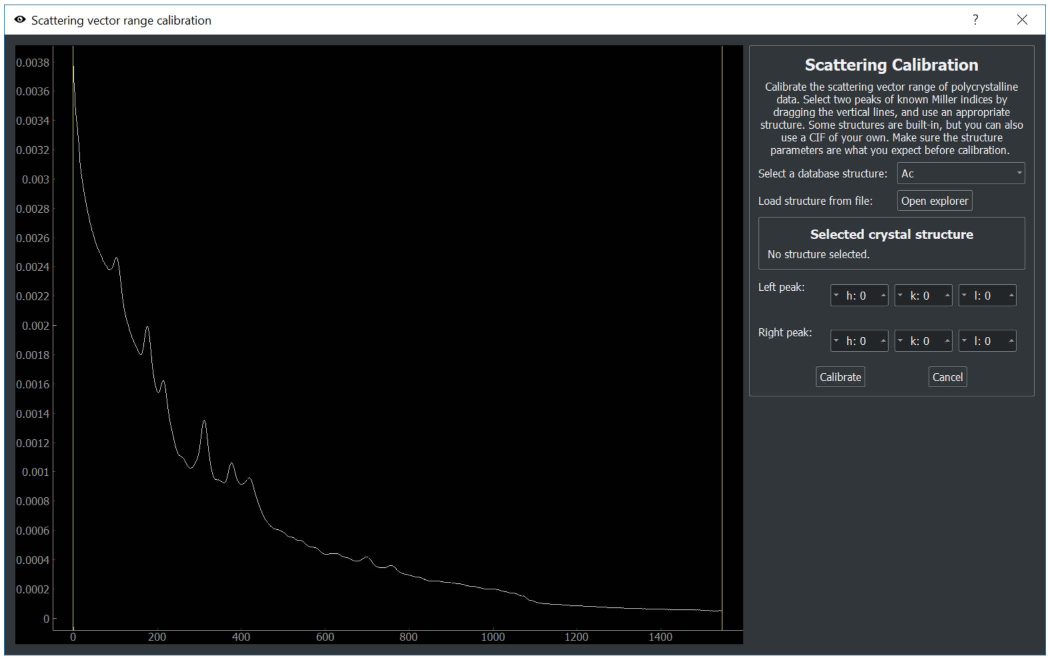
aligned	False
angular_bounds	(0, 360)
camera_length	0.23
center	(758, 1003)
chunks	True

2.2.7 Polycrystalline scattering vector calibration

On the above images, the scattering vector range might not be right. To calibrate the scattering vector range based on a known structure, select the 'Calibrate scattering vector' option from the 'Dataset' menu.


Iris - UED data exploration
File Dataset Help
Process raw data
Symmetrize data (beta)
Calculate angular average
Update dataset metadata
Recompute angular average
Calibrate scattering vector
0.0008
0.0006

This opens the calibration dialog.



Scattering Calibration

Calibrate the scattering vector range of polycrystalline data. Select two peaks of known Miller indices by dragging the vertical lines, and use an appropriate structure. Some structures are built-in, but you can also use a CIF of your own. Make sure the structure parameters are what you expect before calibration.

Select a database structure: Ac

Load structure from file: Open explorer

Selected crystal structure

No structure selected.

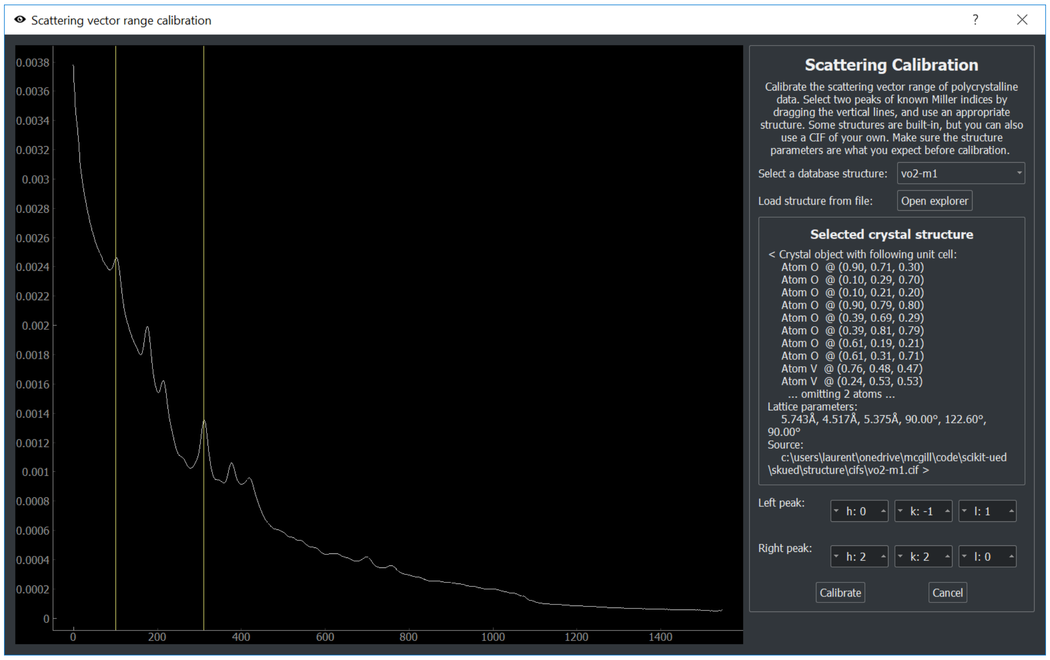
Left peak: h: 0 k: 0 l: 0

Right peak: h: 0 k: 0 l: 0

Calibrate Cancel

You must either select a structure file (CIF) or one of the built-in structures. Once a structure is selected, it's description will be printed on the screen. Make sure this is the crystal structure you expect.

Then, drag the left and right yellow bars on two diffraction peaks with known Miller indices. Click 'Calibrate' to calibrate the scattering vector range.



Scattering Calibration

Calibrate the scattering vector range of polycrystalline data. Select two peaks of known Miller indices by dragging the vertical lines, and use an appropriate structure. Some structures are built-in, but you can also use a CIF of your own. Make sure the structure parameters are what you expect before calibration.

Select a database structure: vo2-m1

Load structure from file: Open explorer

Selected crystal structure

```
< Crystal object with following unit cell:
    Atom O  @ (0.90, 0.71, 0.30)
    Atom O  @ (0.10, 0.29, 0.70)
    Atom O  @ (0.10, 0.21, 0.20)
    Atom O  @ (0.90, 0.79, 0.80)
    Atom O  @ (0.39, 0.69, 0.29)
    Atom O  @ (0.39, 0.81, 0.79)
    Atom O  @ (0.61, 0.19, 0.21)
    Atom O  @ (0.61, 0.31, 0.71)
    Atom V  @ (0.76, 0.48, 0.47)
    Atom V  @ (0.24, 0.53, 0.53)
      ... omitting 2 atoms ...
Lattice parameters:
    5.743Å, 4.517Å, 5.375Å, 90.00°, 122.60°,
90.00°
Source:
    c:\users\laurent\onedrive\mcgill\code\scikit-ued
\skued\structure\cifs\vo2-m1.cif >
```

Left peak: h: 0 k: -1 l: 1

Right peak: h: 2 k: 2 l: 0

Calibrate Cancel

2.3 Datasets in Iris

2.3.1 The `DiffractionDataset` object

The *`DiffractionDataset`* object is the basis for `iris`'s interaction with ultrafast electron diffraction data. *`DiffractionDataset`* objects are simply HDF5 files with a specific layout, and associated methods:

```
from iris import DiffractionDataset
import h5py

assert issubclass(DiffractionDataset, h5py.File)     # yep
```

You can take a look at h5py's documentation to familiarize yourself with `h5py.File`.

You can also use other HDF5 bindings to inspect *`DiffractionDataset`* instances.

Creating a `DiffractionDataset`

An easy way to create a DiffractionDataset is through the *`DiffractionDataset.from_collection()`* method, which saves diffraction patterns and metadata:

`classmethod` `DiffractionDataset.`**`from_collection`**(*patterns*, *filename*, *time_points*, *metadata*, *valid_mask=None*, *dtype=None*, *ckwargs=None*, *callback=None*, ***kwargs*)

Create a DiffractionDataset from a collection of diffraction patterns and metadata.

Parameters

- **`patterns`** (*`iterable of ndarray or ndarray`*) – Diffraction patterns. These should be in the same order as `time_points`. Note that the iterable can be a generator, in which case it will be consumed.
- **`filename`** (*`str or path-like`*) – Path to the assembled DiffractionDataset.
- **`time_points`** (*`array_like, shape (N,)`*) – Time-points of the diffraction patterns, in picoseconds.
- **`metadata`** (*`dict`*) – Valid keys are contained in `DiffractionDataset.valid_metadata`.
- **`valid_mask`** (*`ndarray or None, optional`*) – Boolean array that evaluates to True on valid pixels. This information is useful in cases where a beamblock is used.
- **`dtype`** (*`dtype or None, optional`*) – Patterns will be cast to `dtype`. If None (default), `dtype` will be set to the same data-type as the first pattern in `patterns`.
- **`ckwargs`** (*`dict, optional`*) – HDF5 compression keyword arguments. Refer to `h5py`'s documentation for details. Default is to use the *lzf* compression pipeline.
- **`callback`** (*`callable or None, optional`*) – Callable that takes an int between 0 and 99. This can be used for progress update when `patterns` is a generator and involves large computations.
- **`kwargs`** – Keywords are passed to `h5py.File` constructor. Default is file-mode 'x', which raises error if file already exists. Default libver is 'latest'.

Returns dataset

Return type *DiffractionDataset*

The required metadata that must be passed to `DiffractionDataset.from_collection()` is also listed in `DiffractionDataset.valid_metadata`. Metadata not listed in `DiffractionDataset.valid_metadata` will be *ignored*.

An other possibility is to create a `DiffractionDataset` from a `AbstractRawDataset` subclass using the `DiffractionDataset.from_raw()` method :

classmethod `DiffractionDataset.`**`from_raw`**(*raw*, *filename*, *exclude_scans=None*, *valid_mask=None*, *processes=1*, *callback=None*, *align=True*, *normalize=True*, *ckwargs=None*, *dtype=None*, ***kwargs*)

Create a DiffractionDataset from a subclass of AbstractRawDataset.

Parameters

- **`raw`** (*`AbstractRawDataset instance`*) – Raw dataset instance.
- **`filename`** (*`str or path-like`*) – Path to the assembled DiffractionDataset.
- **`exclude_scans`** (*`iterable of ints or None, optional`*) – Scans to exclude from the processing. Default is to include all scans.
- **`valid_mask`** (*`ndarray or None, optional`*) – Boolean array that evaluates to True on valid pixels. This information is useful in cases where a beamblock is used.
- **`processes`** (*`int or None, optional`*) – Number of Processes to spawn for processing. Default is number of available CPU cores.
- **`callback`** (*`callable or None, optional`*) – Callable that takes an int between 0 and 99. This can be used for progress update.
- **`align`** (*`bool, optional`*) – If True (default), raw images will be aligned on a per-scan basis.
- **`normalize`** (*`bool, optional`*) – If True, images within a scan are normalized to the same integrated diffracted intensity.
- **`ckwargs`** (*`dict or None, optional`*) – HDF5 compression keyword arguments. Refer to `h5py`'s documentation for details.
- **`dtype`** (*`dtype or None, optional`*) – Patterns will be cast to `dtype`. If None (default), `dtype` will be set to the same data-type as the first pattern in `patterns`.
- **`kwargs`** – Keywords are passed to `h5py.File` constructor. Default is file-mode 'x', which raises error if file already exists.

Returns **dataset**

Return type *DiffractionDataset*

See also:

`open_raw()` open raw datasets by guessing the appropriate format based on available plug-ins.

Raises IOError : If the filename is already associated with a file.

Important Methods for the `DiffractionDataset`

The following three methods are the bread-and-butter of interacting with data. See the API section for a complete description.

`DiffractionDataset.`**`diff_data`**(*timedelay*, *relative=False*, *out=None*)

Returns diffraction data at a specific time-delay.

Parameters

- **`timdelay`** (*`float or None`*) – Timedelay [ps]. If None, the entire block is returned.
- **`relative`** (*`bool, optional`*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **`out`** (*`ndarray or None, optional`*) – If an out ndarray is provided, h5py can avoid making intermediate copies.

Returns **arr** – Time-delay data. If `out` is provided, `arr` is a view into `out`.

Return type ndarray

Raises `ValueError` – If timedelay does not exist.

`DiffractionDataset.`**`diff_eq`**()

Returns the averaged diffraction pattern for all times before photoexcitation. In case no data is available before photoexcitation, an array of zeros is returned. The result of this function is cached to minimize overhead.

Time-zero can be adjusted using the `shift_time_zero` method.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,)

`DiffractionDataset.`**`time_series`**(*rect*, *relative=False*, *out=None*)

Integrated intensity over time inside bounds.

Parameters

- **`rect`** (*`4-tuple of ints`*) – Bounds of the region in px. Bounds are specified as [row1, row2, col1, col2]
- **`relative`** (*`bool, optional`*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **`out`** (*`ndarray or None, optional`*) – 1-D ndarray in which to store the results. The shape should be compatible with `(len(time_points),)`

Returns **out**

Return type ndarray, ndim 1

2.3.2 The `PowderDiffractionDataset` object

For polycrystalline data, we can define more data structures and methods. A *`PowderDiffractionDataset`* is a strict subclass of a *`DiffractionDataset`*, and hence all methods previously described are also available.

Specializing a *`DiffractionDataset`* object into a *`PowderDiffractionDataset`* is done as follows:

```
from iris import PowderDiffractionDataset
dataset_path = 'C:\\path_do_dataset.hdf5'    # DiffractionDataset already exists

with PowderDiffractionDataset.from_dataset(dataset_path, center) as dset:
    # Do computation
```

Important Methods for the `PowderDiffractionDataset`

The following methods are specific to polycrystalline diffraction data. See the API section for a complete description.

`PowderDiffractionDataset.`**`powder_eq`**`()`

Returns the average powder diffraction pattern for all times before photoexcitation. In case no data is available before photoexcitation, an array of zeros is returned.

Parameters **`bgr`** (*`bool`*) – If True, background is removed.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,)

`PowderDiffractionDataset.`**`powder_data`**(*timedelay*, *bgr=False*, *relative=False*, *out=None*)

Returns the angular average data from scan-averaged diffraction patterns.

Parameters

- **`timdelay`** (*`float or None`*) – Time-delay [ps]. If None, the entire block is returned.
- **`bgr`** (*`bool, optional`*) – If True, background is removed.
- **`relative`** (*`bool, optional`*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **`out`** (*`ndarray or None, optional`*) – If an out ndarray is provided, h5py can avoid making intermediate copies.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,) or (N,M)

`PowderDiffractionDataset.`**`powder_calq`**(*crystal*, *peak_indices*, *miller_indices*)

Determine the scattering vector q corresponding to a polycrystalline diffraction pattern and a known crystal structure.

For best results, multiple peaks (and corresponding Miller indices) should be provided; the absolute minimum is two.

Parameters

- **`crystal`** (*`skued.Crystal instance`*) – Crystal that gave rise to the diffraction data.
- **`peak_indices`** (*`n-tuple of ints`*) – Array index location of diffraction peaks. For best results, peaks should be well-separated. More than two peaks can be used.
- **`miller_indices`** (*`iterable of 3-tuples`*) – Indices associated with the peaks of `peak_indices`. More than two peaks can be used. E.g. `indices = [(2,2,0), (-3,0,2)]`

Raises

- ValueError : if the number of peak indices does not match the number of Miller indices.
- ValueError : if the number of peaks given is lower than two.

`PowderDiffractionDataset.`**`compute_baseline`**(*first_stage*, *wavelet*, *max_iter=50*, *level=None*, ***kwargs*)

Compute and save the baseline computed based on the dual-tree complex wavelet transform. All keyword arguments are passed to scikit-ued's *baseline_dt* function.

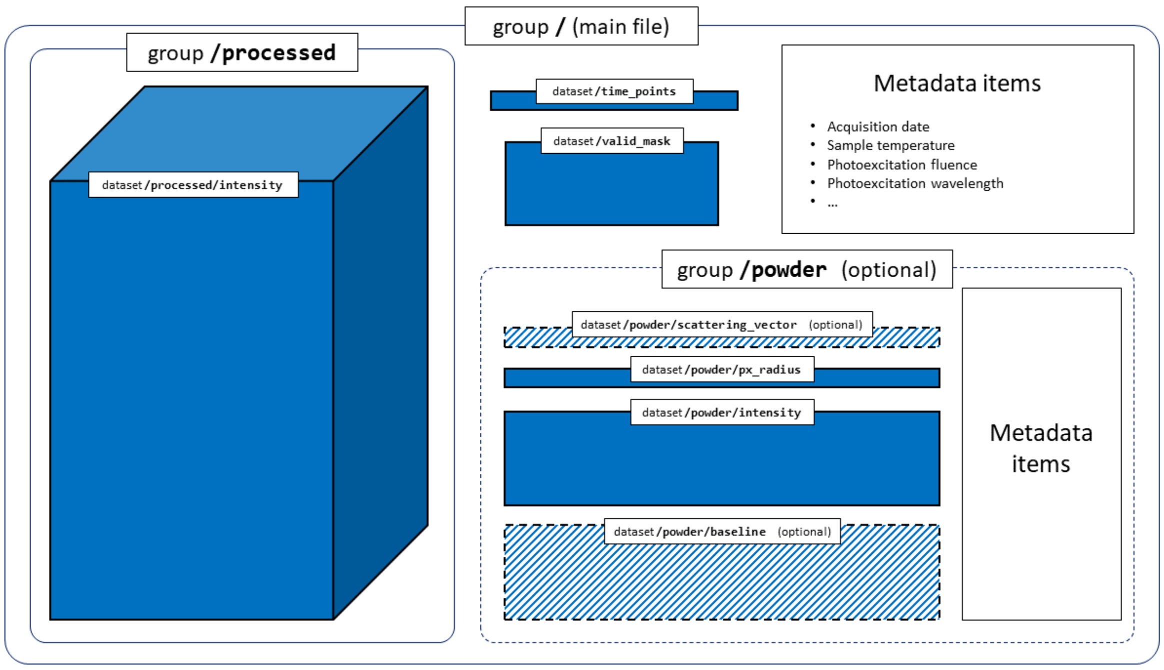
Parameters

- **`first_stage`** (*`str, optional`*) – Wavelet to use for the first stage. See `skued.available_first_stage_filters()` for a list of suitable arguments

- **wavelet** (*str, optional*) – Wavelet to use in stages > 1. Must be appropriate for the dual-tree complex wavelet transform. See `skued.available_dt_filters()` for possible values.
- **max_iter** (*int, optional*) –
- **level** (*int or None, optional*) – If None (default), maximum level is used.

2.3.3 HDF5 layout

DiffractionDataset instances (and by extension, *PowderDiffractionDataset* instances) are a specialization of HDF5 files. Therefore, it is possible to inspect and manipulate instances with any other tool that has bindings to the HDF5 libraries. The HDF5 layout is presented below.



2.4 Dataset Plug-ins

To use your own raw data with `iris`, a plug-in functionality is made available.

Plug-ins are Python modules that implement a subclass of *AbstractRawDataset*, and should be placed in `~/iris_plugins` (`C:\Users\UserName\iris_plugins` on Windows). Subclasses of *AbstractRawDataset* are automatically detected by `iris` and can be used via the GUI.

Installed plug-ins can be imported from `iris.plugins`:

```
from iris.plugins import DatasetSubclass
```

which would work if the `DatasetSubclass` is defined in the file `~/iris_plugins/<anything>.py`. Example plug-ins is available here. Plug-ins used by members of the Siwick research group are visible here.

2.4.1 Installing a plug-in

To install a plug-in that you have written in a file named `~/myplugin.py`:

```
import iris
iris.install_plugin('~/myplugin.py')
```

Installing a plug-in in the above makes it immediately available.

install_plugin(path)	Install and load an iris plug-in.

iris.install_plugin

`iris.`**`install_plugin`**`(`*path*`)`

Install and load an iris plug-in. Installed plug-ins are loaded at every iris start-up.

New in version 5.0.4.

Parameters **`path`** (*`path-like`*) – Path to the plug-in. This plug-in file will be copied.

2.5 Subclassing AbstractRawDataset

To take advantage of `iris`'s *`DiffractionDataset`* and *`PowderDiffractionDataset`*, an appropriate subclass of *`AbstractRawDataset`* must be implemented. This subclass can then be fed to *`DiffractionDataset.from_raw()`* to produce a *`DiffractionDataset`*.

2.5.1 How to assemble a AbstractRawDataset subclass

Ultrafast electron diffraction experiments typically have multiple *scans*. Each scan consists of a time-delay sweep. You can think of it as one scan being an experiment, and so each dataset is composed of multiple, equivalent experiments.

To subclass *`AbstractRawDataset`*, the method *`AbstractRawDataset.raw_data()`* must minimally implemented. It must follow the following specification:

`AbstractRawDataset.`**`raw_data`**`(`*timedelay*, *scan=1*, ***kwargs*`)`

Returns an array of the image at a timedelay and scan.

Parameters

- **`timdelay`** (*`float`*) – Acquisition time-delay.
- **`scan`** (*`int, optional`*) – Scan number. Default is 1.
- **`kwargs`** – Keyword-arguments are ignored.

Returns **arr**

Return type *~numpy.ndarray*, ndim 2

Raises

- ValueError : if `timedelay` or `scan` are invalid / out of bounds.
- IOError : Filename is not associated with an image/does not exist.

For better performance, or to tailor data reduction to your data acquisition scheme, the following method can also be overloaded:

`AbstractRawDataset.`**`reduced`**(*exclude_scans=None*, *align=True*, *normalize=True*, *mask=None*, *processes=1*, *dtype=<class 'float'>*)

Generator of reduced dataset. The reduced diffraction patterns are generated in order of time-delay.

This particular implementation normalizes diffracted intensity of pictures acquired at the same time-delay while rejecting masked pixels.

Parameters

- **exclude_scans** (*iterable or None, optional*) – These scans will be skipped when reducing the dataset.
- **align** (*bool, optional*) – If True (default), raw diffraction patterns will be aligned using the masked normalized cross-correlation approach. See *skued.align* for more information.
- **normalize** (*bool, optional*) – If True (default), equivalent diffraction pictures (e.g. same time-delay, different scans) are normalized to the same diffracted intensity.
- **mask** (*array-like of bool or None, optional*) – If not None, pixels where `mask = True` are ignored for certain operations (e.g. alignment).
- **processes** (*int or None, optional*) – Number of Processes to spawn for processing.
- **dtype** (*numpy.dtype or None, optional*) – Reduced patterns will be cast to `dtype`.

Yields **pattern** (*~numpy.ndarray*, ndim 2)

2.5.2 AbstractRawDataset metadata

`AbstractRawDataset` subclasses automatically include the following metadata:

- `date` (*str*): Acquisition date. Date format is up to you.
- `energy` (*float*): Electron energy in keV.
- `pump_wavelength` (*int*): photoexcitation wavelength in nanometers.
- `fluence` (*float*): photoexcitation fluence $\mathrm{mJ}/\mathrm{cm} * *2$.
- `time_zero_shift` (*float*): Time-zero shift in picoseconds.
- `temperature` (*float*): sample temperature in Kelvins.
- `exposure` (*float*): picture exposure in seconds.
- `resolution` (2-*tuple*): pixel resolution of pictures.
- `time_points` (*tuple*): time-points in picoseconds.
- `scans` (*tuple*): experimental scans.
- `camera_length` (*float*): sample-to-camera distance in meters.
- `pixel_width` (*float*): pixel width in meters.
- `notes` (*str*): notes.

Subclasses can add more metadata or override the current metadata with new defaults.

All proper subclasses of *`AbstractRawDataset`* are automatically added to the possible raw dataset formats that can be loaded from the GUI.

2.6 Reference/API

2.6.1 Opening raw datasets

To open any raw dataset, take a look at the `open_raw()` function.

`iris.`**`open_raw`**(*path*)

Open a raw data item, guessing the AbstractRawDataset instance that should be used based on available plug-ins.

This function can also be used as a context manager:

```
with open_raw('.') as dset:
    ...
```

Parameters **path** (*path-like*) – Path to the file/folder containing the raw data.

Returns **raw** – The raw dataset. If no format could be guessed, an RuntimeError is raised.

Return type AbstractRawDataset instance

Raises RuntimeError : if the data format could not be guessed.

2.6.2 Raw Dataset Classes

AbstractRawDataset([source, metadata])	Abstract base class for ultrafast electron diffraction data set.

iris.AbstractRawDataset

class `iris.`**`AbstractRawDataset`**(*source=None*, *metadata=None*)

Abstract base class for ultrafast electron diffraction data set. AbstractRawDataset allows for enforced metadata types and values, as well as a standard interface. For example, AbstractRawDataset implements the context manager interface.

Minimally, the following method must be implemented in subclasses:

- raw_data

It is suggested to also implement the following magic method:

- __init__
- __exit__

Optionally, the `display_name` class attribute can be specified.

For better results or performance during reduction, the following methods can be specialized:

- reduced

A list of concrete implementations of AbstractRawDatasets is available in the `implementations` class attribute. Subclasses are automatically added.

The call signature must remain the same for all overwritten methods.

Parameters

- **source** (*object*) – Data source, for example a directory or external file.

- **metadata** (*dict or None, optional*) – Metadata and experimental parameters. Dictionary keys that are not valid metadata, they are ignored. Metadata can also be set directly later.

Raises TypeError : if an item from the metadata has an unexpected type.

__init__(*source=None*, *metadata=None*)

Parameters

- **source** (*object*) – Data source, for example a directory or external file.
- **metadata** (*dict or None, optional*) – Metadata and experimental parameters. Dictionary keys that are not valid metadata, they are ignored. Metadata can also be set directly later.

Raises TypeError : if an item from the metadata has an unexpected type.

Methods

__init__([source, metadata])	**param source** Data source, for example a directory or external file.
iterscan(scan, **kwargs)	Generator function of diffraction patterns as part of a scan, in time-delay order.
raw_data(timedelay[, scan])	Returns an array of the image at a timedelay and scan.
reduced([exclude_scans, align, normalize, . . .])	Generator of reduced dataset.
update_metadata(metadata)	Update metadata from a dictionary.

__exit__(**exc*)
Raise any exception triggered within the runtime context.

__init__(*source=None*, *metadata=None*)

Parameters

- **source** (*object*) – Data source, for example a directory or external file.
- **metadata** (*dict or None, optional*) – Metadata and experimental parameters. Dictionary keys that are not valid metadata, they are ignored. Metadata can also be set directly later.

Raises TypeError : if an item from the metadata has an unexpected type.

__repr__()
Return repr(self).

iterscan(*scan*, ***kwargs*)
Generator function of diffraction patterns as part of a scan, in time-delay order.

Parameters

- **scan** (*int*) – Scan from which to yield the data.
- **kwargs** – Keyword-arguments are passed to `raw_data` method.

Yields **data** (~*numpy.ndarray*, ndim 2)

See also:

`itertime()` generator of diffraction patterns for a single time-delay, in scan order

`itertime`(*timedelay*, *exclude_scans=None*, ***kwargs*)

Generator function of diffraction patterns of the same time-delay, in scan order.

Parameters

- **`timedelay`** (*`float`*) –
- **`from which to yield the data.`** (*`Scan`*) –

exclude_scans [iterable or None, optional] These scans will be skipped.

kwargs Keyword-arguments are passed to `raw_data` method.

Yields data (~*numpy.ndarray*, ndim 2)

See also:

`iterscan()` generator of diffraction patterns for a single scan, in time-delay order

`metadata`

Experimental parameters and dataset metadata as a dictionary.

`raw_data`(*timedelay*, *scan=1*, ***kwargs*)

Returns an array of the image at a timedelay and scan.

Parameters

- **`timdelay`** (*`float`*) – Acquisition time-delay.
- **`scan`** (*`int, optional`*) – Scan number. Default is 1.
- **`kwargs`** – Keyword-arguments are ignored.

Returns arr

Return type ~*numpy.ndarray*, ndim 2

Raises

- ValueError : if `timedelay` or `scan` are invalid / out of bounds.
- IOError : Filename is not associated with an image/does not exist.

`reduced`(*exclude_scans=None*, *align=True*, *normalize=True*, *mask=None*, *processes=1*, *dtype=<class 'float'>*)

Generator of reduced dataset. The reduced diffraction patterns are generated in order of time-delay.

This particular implementation normalizes diffracted intensity of pictures acquired at the same time-delay while rejecting masked pixels.

Parameters

- **`exclude_scans`** (*`iterable or None, optional`*) – These scans will be skipped when reducing the dataset.
- **`align`** (*`bool, optional`*) – If True (default), raw diffraction patterns will be aligned using the masked normalized cross-correlation approach. See *skued.align* for more information.
- **`normalize`** (*`bool, optional`*) – If True (default), equivalent diffraction pictures (e.g. same time-delay, different scans) are normalized to the same diffracted intensity.

- **mask** (*array-like of bool or None, optional*) – If not None, pixels where `mask = True` are ignored for certain operations (e.g. alignment).
- **processes** (*int or None, optional*) – Number of Processes to spawn for processing.
- **dtype** (*numpy.dtype or None, optional*) – Reduced patterns will be cast to `dtype`.

Yields **pattern** (*~numpy.ndarray*, ndim 2)

update_metadata (*metadata*)

Update metadata from a dictionary. Only appropriate keys are used; irrelevant keys are ignored.

Parameters **metadata** (*dictionary*) – See `AbstractRawDataset.valid_metadata` for valid keys.

2.6.3 Diffraction Dataset Classes

`DiffractionDataset`(name[, mode, driver, . . .])	Abstraction of an HDF5 file to represent diffraction datasets.
`PowderDiffractionDataset`(*args, **kwargs)	Abstraction of HDF5 files for powder diffraction datasets.

iris.DiffractionDataset

class iris.**DiffractionDataset** (*name*, *mode=None*, *driver=None*, *libver=None*, *userblock_size=None*, *swmr=False*, *rdcc_nslots=None*, *rdcc_nbytes=None*, *rdcc_w0=None*, *track_order=None*, ***kwds*)

Abstraction of an HDF5 file to represent diffraction datasets.

Create a new file object.

See the h5py user guide for a detailed explanation of the options.

name Name of the file on disk, or file-like object. Note: for files created with the 'core' driver, HDF5 still requires this be non-empty.

mode r Readonly, file must exist r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise (default)

driver Name of the driver to use. Legal values are None (default, recommended), 'core', 'sec2', 'stdio', 'mpio'.

libver Library version bounds. Supported values: 'earliest', 'v108', 'v110', and 'latest'. The 'v108' and 'v110' options can only be specified with the HDF5 1.10.2 library or later.

userblock Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr Open the file in SWMR read mode. Only used when mode = 'r'.

rdcc_nbytes Total size of the raw data chunk cache in bytes. The default size is 1024**2 (1 MB) per dataset.

rdcc_w0 The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75.

rdcc_nslots The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521.

track_order Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

Additional keywords Passed on to the selected file driver.

`__init__`(*name*, *mode=None*, *driver=None*, *libver=None*, *userblock_size=None*, *swmr=False*, *rdcc_nslots=None*, *rdcc_nbytes=None*, *rdcc_w0=None*, *track_order=None*, ***kwds*)

Create a new file object.

See the h5py user guide for a detailed explanation of the options.

name Name of the file on disk, or file-like object. Note: for files created with the 'core' driver, HDF5 still requires this be non-empty.

mode r Readonly, file must exist r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise (default)

driver Name of the driver to use. Legal values are None (default, recommended), 'core', 'sec2', 'stdio', 'mpio'.

libver Library version bounds. Supported values: 'earliest', 'v108', 'v110', and 'latest'. The 'v108' and 'v110' options can only be specified with the HDF5 1.10.2 library or later.

userblock Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr Open the file in SWMR read mode. Only used when mode = 'r'.

rdcc_nbytes Total size of the raw data chunk cache in bytes. The default size is 1024**2 (1 MB) per dataset.

rdcc_w0 The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75.

rdcc_nslots The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521.

track_order Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

Additional keywords Passed on to the selected file driver.

`__repr__`()

Return repr(self).

`compression_params`

Compression options in the form of a dictionary

diff_apply(*func*, *callback=None*, *processes=1*)

Apply a function to each diffraction pattern possibly in parallel. The diffraction patterns will be modified in-place.

> **Warning:** This is an irreversible in-place operation.

New in version 5.0.3.

Parameters

- **func** (`callable`) – Function that takes in an array (diffraction pattern) and returns an array of the exact same shape, with the same data-type.
- **callback** (`callable or None, optional`) – Callable that takes an int between 0 and 99. This can be used for progress update.
- **processes** (`int or None, optional`) – Number of parallel processes to use. If `None`, all available processes will be used. In case Single Writer Multiple Reader mode is not available, `processes` is ignored.

  New in version 5.0.6.

Raises TypeError : if *func* is not a proper callable

diff_data(*timedelay*, *relative=False*, *out=None*)

Returns diffraction data at a specific time-delay.

Parameters

- **timdelay** (`float or None`) – Timedelay [ps]. If None, the entire block is returned.
- **relative** (`bool, optional`) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **out** (`ndarray or None, optional`) – If an out ndarray is provided, h5py can avoid making intermediate copies.

Returns **arr** – Time-delay data. If `out` is provided, `arr` is a view into `out`.

Return type ndarray

Raises `ValueError` – If timedelay does not exist.

diff_eq

Returns the averaged diffraction pattern for all times before photoexcitation. In case no data is available before photoexcitation, an array of zeros is returned. The result of this function is cached to minimize overhead.

Time-zero can be adjusted using the `shift_time_zero` method.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,)

classmethod from_collection(*patterns*, *filename*, *time_points*, *metadata*, *valid_mask=None*, *dtype=None*, *ckwargs=None*, *callback=None*, ***kwargs*)

Create a DiffractionDataset from a collection of diffraction patterns and metadata.

Parameters

- **patterns** (`iterable of ndarray or ndarray`) – Diffraction patterns. These should be in the same order as `time_points`. Note that the iterable can be a generator, in which case it will be consumed.

- **`filename`** (*`str or path-like`*) – Path to the assembled DiffractionDataset.
- **`time_points`** (*`array_like, shape (N,)`*) – Time-points of the diffraction patterns, in picoseconds.
- **`metadata`** (*`dict`*) – Valid keys are contained in `DiffractionDataset.valid_metadata`.
- **`valid_mask`** (*`ndarray or None, optional`*) – Boolean array that evaluates to True on valid pixels. This information is useful in cases where a beamblock is used.
- **`dtype`** (*`dtype or None, optional`*) – Patterns will be cast to `dtype`. If None (default), `dtype` will be set to the same data-type as the first pattern in `patterns`.
- **`ckwargs`** (*`dict, optional`*) – HDF5 compression keyword arguments. Refer to `h5py`'s documentation for details. Default is to use the *lzf* compression pipeline.
- **`callback`** (*`callable or None, optional`*) – Callable that takes an int between 0 and 99. This can be used for progress update when `patterns` is a generator and involves large computations.
- **`kwargs`** – Keywords are passed to `h5py.File` constructor. Default is file-mode 'x', which raises error if file already exists. Default libver is 'latest'.

Returns dataset

Return type *DiffractionDataset*

`classmethod from_raw`(*raw*, *filename*, *exclude_scans=None*, *valid_mask=None*, *processes=1*, *callback=None*, *align=True*, *normalize=True*, *ckwargs=None*, *dtype=None*, ***kwargs*)

Create a DiffractionDataset from a subclass of AbstractRawDataset.

Parameters

- **`raw`** (*`AbstractRawDataset instance`*) – Raw dataset instance.
- **`filename`** (*`str or path-like`*) – Path to the assembled DiffractionDataset.
- **`exclude_scans`** (*`iterable of ints or None, optional`*) – Scans to exclude from the processing. Default is to include all scans.
- **`valid_mask`** (*`ndarray or None, optional`*) – Boolean array that evaluates to True on valid pixels. This information is useful in cases where a beamblock is used.
- **`processes`** (*`int or None, optional`*) – Number of Processes to spawn for processing. Default is number of available CPU cores.
- **`callback`** (*`callable or None, optional`*) – Callable that takes an int between 0 and 99. This can be used for progress update.
- **`align`** (*`bool, optional`*) – If True (default), raw images will be aligned on a per-scan basis.
- **`normalize`** (*`bool, optional`*) – If True, images within a scan are normalized to the same integrated diffracted intensity.
- **`ckwargs`** (*`dict or None, optional`*) – HDF5 compression keyword arguments. Refer to `h5py`'s documentation for details.
- **`dtype`** (*`dtype or None, optional`*) – Patterns will be cast to `dtype`. If None (default), `dtype` will be set to the same data-type as the first pattern in `patterns`.
- **`kwargs`** – Keywords are passed to `h5py.File` constructor. Default is file-mode 'x', which raises error if file already exists.

Returns dataset

Return type *DiffractionDataset*

See also:

`open_raw()` open raw datasets by guessing the appropriate format based on available plug-ins.

Raises IOError : If the filename is already associated with a file.

`invalid_mask`
Array that evaluates to True on invalid pixels (i.e. on beam-block, hot pixels, etc.)

`metadata`
Dictionary of the dataset's metadata. Dictionary is sorted alphabetically by keys.

`resolution`
Resolution of diffraction patterns (px, px)

`shift_time_zero`(*shift*)
Insert a shift in time points. Reset the shift by setting it to zero. Shifts are not consecutive, so that calling *shift_time_zero(20)* twice will not result in a shift of 40ps.

Parameters **`shift`** (`float`) – Shift [ps]. A positive value of *shift* will move all time-points forward in time, whereas a negative value of *shift* will move all time-points backwards in time.

`symmetrize`(*mod*, *center*, *kernel_size=None*, *callback=None*, *processes=1*)
Symmetrize diffraction images based on n-fold rotational symmetry.

Warning: This is an irreversible in-place operation.

Parameters

- **`mod`** (`int`) – Fold symmetry number.
- **`center`** (`array-like, shape (2,) or None`) – Coordinates of the center (in pixels). If None, the data is symmetrized around the center of the images.
- **`kernel_size`** (`float or None, optional`) – If not None, every diffraction pattern will be smoothed with a gaussian kernel. *kernel_size* is the standard deviation of the gaussian kernel in units of pixels.
- **`callback`** (`callable or None, optional`) – Callable that takes an int between 0 and 99. This can be used for progress update.
- **`processes`** (`int or None, optional`) – Number of parallel processes to use. If `None`, all available processes will be used. In case Single Writer Multiple Reader mode is not available, `processes` is ignored.

  New in version 5.0.6.

Raises ValueError: if `mod` is not a divisor of 360.

See also:

`diff_apply()` apply an operation to each diffraction pattern one-by-one

time_series(*rect*, *relative=False*, *out=None*)
Integrated intensity over time inside bounds.

Parameters

- **rect** (*4-tuple of ints*) – Bounds of the region in px. Bounds are specified as [row1, row2, col1, col2]
- **relative** (*bool, optional*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **out** (*ndarray or None, optional*) – 1-D ndarray in which to store the results. The shape should be compatible with `(len(time_points),)`

Returns out

Return type ndarray, ndim 1

valid_mask
Array that evaluates to True on valid pixels (i.e. not on beam-block, not hot pixels, etc.)

iris.PowderDiffractionDataset

class iris.**PowderDiffractionDataset**(**args*, ***kwargs*)
Abstraction of HDF5 files for powder diffraction datasets.

__init__(**args*, ***kwargs*)
Create a new file object.

See the h5py user guide for a detailed explanation of the options.

name Name of the file on disk, or file-like object. Note: for files created with the 'core' driver, HDF5 still requires this be non-empty.

mode r Readonly, file must exist r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise (default)

driver Name of the driver to use. Legal values are None (default, recommended), 'core', 'sec2', 'stdio', 'mpio'.

libver Library version bounds. Supported values: 'earliest', 'v108', 'v110', and 'latest'. The 'v108' and 'v110' options can only be specified with the HDF5 1.10.2 library or later.

userblock Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr Open the file in SWMR read mode. Only used when mode = 'r'.

rdcc_nbytes Total size of the raw data chunk cache in bytes. The default size is 1024**2 (1 MB) per dataset.

rdcc_w0 The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75.

rdcc_nslots The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least

10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521.

track_order Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

Additional keywords Passed on to the selected file driver.

`__init__`(**args*, ***kwargs*)
Create a new file object.

See the h5py user guide for a detailed explanation of the options.

name Name of the file on disk, or file-like object. Note: for files created with the 'core' driver, HDF5 still requires this be non-empty.

mode r Readonly, file must exist r+ Read/write, file must exist w Create file, truncate if exists w- or x Create file, fail if exists a Read/write if exists, create otherwise (default)

driver Name of the driver to use. Legal values are None (default, recommended), 'core', 'sec2', 'stdio', 'mpio'.

libver Library version bounds. Supported values: 'earliest', 'v108', 'v110', and 'latest'. The 'v108' and 'v110' options can only be specified with the HDF5 1.10.2 library or later.

userblock Desired size of user block. Only allowed when creating a new file (mode w, w- or x).

swmr Open the file in SWMR read mode. Only used when mode = 'r'.

rdcc_nbytes Total size of the raw data chunk cache in bytes. The default size is 1024**2 (1 MB) per dataset.

rdcc_w0 The chunk preemption policy for all datasets. This must be between 0 and 1 inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks to flush from cache. A value of 0 means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of 1 means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to 1. Otherwise, this should be set lower depending on how often you re-read or re-write the same data. The default value is 0.75.

rdcc_nslots The number of chunk slots in the raw data chunk cache for this file. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in rdcc_nbytes bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is 521.

track_order Track dataset/group/attribute creation order under root group if True. If None use global default h5.get_config().track_order.

Additional keywords Passed on to the selected file driver.

`compute_angular_averages`(*center=None*, *normalized=False*, *angular_bounds=None*, *trim=True*, *callback=None*)
Compute the angular averages.

Parameters

- **`center`** (*`2-tuple or None, optional`*) – Center of the diffraction patterns. If None (default), the dataset attribute will be used instead.
- **`normalized`** (*`bool, optional`*) – If True, each pattern is normalized to its integral.

- **angular_bounds** (*2-tuple of float or None, optional*) – Angle bounds are specified in degrees. 0 degrees is defined as the positive x-axis. Angle bounds outside [0, 360) are mapped back to [0, 360).
- **trim** (*bool, optional*) – If True, leading/trailing zeros - possibly due to masks - are trimmed.
- **callback** (*callable or None, optional*) – Callable of a single argument, to which the calculation progress will be passed as an integer between 0 and 100.

compute_baseline (*first_stage*, *wavelet*, *max_iter=50*, *level=None*, ***kwargs*)

Compute and save the baseline computed based on the dual-tree complex wavelet transform. All keyword arguments are passed to scikit-ued's *baseline_dt* function.

Parameters

- **first_stage** (*str, optional*) – Wavelet to use for the first stage. See `skued.available_first_stage_filters()` for a list of suitable arguments
- **wavelet** (*str, optional*) – Wavelet to use in stages > 1. Must be appropriate for the dual-tree complex wavelet transform. See `skued.available_dt_filters()` for possible values.
- **max_iter** (*int, optional*) –
- **level** (*int or None, optional*) – If None (default), maximum level is used.

classmethod from_dataset (*dataset*, *center*, *normalized=True*, *angular_bounds=None*, *callback=None*)

Transform a DiffractionDataset instance into a PowderDiffractionDataset. This requires computing the azimuthal averages as well.

Parameters

- **dataset** (`DiffractionDataset`) – DiffractionDataset instance.
- **center** (*2-tuple or None, optional*) – Center of the diffraction patterns. If None (default), the dataset attribute will be used instead.
- **normalized** (*bool, optional*) – If True, each pattern is normalized to its integral. Default is False.
- **angular_bounds** (*2-tuple of float or None, optional*) – Angle bounds are specified in degrees. 0 degrees is defined as the positive x-axis. Angle bounds outside [0, 360) are mapped back to [0, 360).
- **callback** (*callable or None, optional*) – Callable of a single argument, to which the calculation progress will be passed as an integer between 0 and 100.

Returns **powder**

Return type *PowderDiffractionDataset*

powder_baseline (*timedelay*, *out=None*)

Returns the baseline data.

Parameters

- **timdelay** (*float or None*) – Time-delay [ps]. If None, the entire block is returned.
- **out** (*ndarray or None, optional*) – If an out ndarray is provided, h5py can avoid making intermediate copies.

Returns **out** – If a baseline hasn't been computed yet, the returned array is an array of zeros.

Return type ndarray

powder_calq(*crystal*, *peak_indices*, *miller_indices*)

Determine the scattering vector q corresponding to a polycrystalline diffraction pattern and a known crystal structure.

For best results, multiple peaks (and corresponding Miller indices) should be provided; the absolute minimum is two.

Parameters

- **crystal** (*skued.Crystal instance*) – Crystal that gave rise to the diffraction data.
- **peak_indices** (*n-tuple of ints*) – Array index location of diffraction peaks. For best results, peaks should be well-separated. More than two peaks can be used.
- **miller_indices** (*iterable of 3-tuples*) – Indices associated with the peaks of `peak_indices`. More than two peaks can be used. E.g. `indices = [(2,2,0), (-3,0,2)]`

Raises

- ValueError : if the number of peak indices does not match the number of Miller indices.
- ValueError : if the number of peaks given is lower than two.

powder_data(*timedelay*, *bgr=False*, *relative=False*, *out=None*)

Returns the angular average data from scan-averaged diffraction patterns.

Parameters

- **timdelay** (*float or None*) – Time-delay [ps]. If None, the entire block is returned.
- **bgr** (*bool, optional*) – If True, background is removed.
- **relative** (*bool, optional*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.
- **out** (*ndarray or None, optional*) – If an out ndarray is provided, h5py can avoid making intermediate copies.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,) or (N,M)

powder_eq

Returns the average powder diffraction pattern for all times before photoexcitation. In case no data is available before photoexcitation, an array of zeros is returned.

Parameters **bgr** (*bool*) – If True, background is removed.

Returns **I** – Diffracted intensity [counts]

Return type ndarray, shape (N,)

powder_time_series(*rmin*, *rmax*, *bgr=False*, *relative=False*, *units='pixels'*, *out=None*)

Average intensity over time. Diffracted intensity is integrated in the closed interval [rmin, rmax]

Parameters

- **rmin** (*float*) – Lower scattering vector bound [1/A]
- **rmax** (*float*) – Higher scattering vector bound [1/A].
- **bgr** (*bool, optional*) – If True, background is removed. Default is False.
- **relative** (*bool, optional*) – If True, data is returned relative to the average of all diffraction patterns before photoexcitation.

- **units** (*str, {'pixels', 'momentum'}*) – Units of the bounds rmin and rmax.
- **out** (*ndarray or None, optional*) – 1-D ndarray in which to store the results. The shape should be compatible with (len(time_points),)

Returns **out** – Average diffracted intensity over time.

Return type ndarray, shape (N,)

px_radius
Pixel-radius of azimuthal average

scattering_vector
Array of scattering vector norm $|q|$ [1/]

shift_time_zero(**args*, ***kwargs*)
Shift time-zero uniformly across time-points.

Parameters **shift** (*float*) – Shift [ps]. A positive value of *shift* will move all time-points forward in time, whereas a negative value of *shift* will move all time-points backwards in time.

2.7 What's new

2.7.1 5.2.0 (development)

- Official support for Linux.
- Plug-ins installed via the GUI can now be used right away. No restarts required.
- Added the *iris.plugins.load_plugin* function to load plug-ins without installing them. Useful for testing.
- Plug-ins can now have the `display_name` property which will be displayed in the GUI. This is optional and backwards-compatible.
- Siwick Research Group-specific plugins were removed. They can be found here: https://github.com/Siwick-Research-Group/iris-ued-plugins
- Switched to Azure Pipelines for continuous integration builds;
- Added cursor information (position and image value) for processed data view;
- Fixed an issue where very large relative differences in datasets would crash the GUI displays;
- Fixed an issue where time-series fit would not display properly in fractional change mode;

2.7.2 5.1.3

- Added logging support for the GUI component. Logs can be reached via the help menu
- Added an update check. You can see whether an update is available via the help menu, as well as via the status bar.
- Added the ability to view time-series dynamics in absolute units AND relative change.
- Pinned dependency to scikit-ued, to prevent upgrade to scikit-ued 2.0 unless appropriate.
- Pinned dependency to npstreams, to prevent upgrade to npstreams 2.0 unless appropriate.

2.7.3 5.1.2

- Fixed an issue where the QDarkStyle internal imports were absolute.

2.7.4 5.1.1

- Fixed an issue where data reduction would freeze when using more than one CPU;
- Removed the auto-update mechanism. Update checks will run in the background only;
- Fixed an issue where the in-progress indicator would freeze;
- Moved tests outside of source repository;
- Updated GUI stylesheet to QDarkStyle 2.6.6;

2.7.5 5.1.0

- Added explicit support for Python 3.7;
- Usability tweaks, for example more visible mask controls;
- Added the ability to create standalone executables via PyInstaller;
- Added the ability to create Windows installers;

2.7.6 5.0.5.1

- Due to new forced image orientation, objects on screens were not properly registered (e.g. diffraction center finder).

2.7.7 5.0.5

- Added the ability to fit exponentials to time-series;
- Added region-of-interest text bounds for easier time-series exploration
- Enforced PyQtGraph to use row-major image orientation
- Datasets are now opened in read-only mode unless absolutely necessary. This should make it safer to handler multiple instances of iris at the same time.

2.7.8 5.0.4

- Better plug-in handling and command-line interface.

2.7.9 5.0.3

The major change in this version is the ability to guess raw dataset formats using the *iris.open_raw* function. This allows the possibility to start the GUI and open a dataset at the same time.

2.7.10 5.0.2

The package now only has dependencies that can be installed through *conda*

2.7.11 5.0.1

This is a minor bug-fix release that also includes user interface niceties (e.g. link to online documentation) and user experience niceties (e.g. confirmation message if you forget pixel masks).

2.7.12 5.0.0

This new version includes a completely rewritten library and GUI front-end. Earlier datasets will need to be reprocessed. New features:

- Faster performance thanks to better data layout in HDF5;
- Plug-in architecture for various raw data formats;
- Faster performance thanks to npstreams package;
- Easier to extend GUI skeleton;
- Online documentation accessible from the GUI;
- Continuous integration.

CHAPTER 3

Authors

- Laurent P. René de Cotret (McGill)

Index