Xanespy Documentation

Release 0.1.2

Mark Wolf

Jul 13, 2017

CONTENTS:

Python library for analyzing **X-Ray absorbance spectroscopy** data.

Warning: This documentation is still under development. While it is usable, the code could change at any time.

CHAPTER

ONE

INTRODUCTION

Xanespy is a toolkit for interacting with X-ray microscopy data, most likely collected at a synchrotron beamline. By collecting a set of frames at multiple X-ray energies, spectral maps are reconstructed to provide chemical insight. Multiple framesets can be collected sequentially as part of an *operando* experiment and analyzed simultaneously in python. Slow operations take advantage of multiple cores when available.

This project has the following design goals:

- Provide a python toolkit for analysis of X-ray absorbance frames.
- Store data in an open format for easy distribution.

GUI tools (eg. TXM-Wizard) exist for performing this type of analysis. While convenient, the downside to this approach is the potential inability to exactly reproduce a given set of steps. Xanespy does provide an interactive GUI for visualizing the data, but this GUI does not alter the data or export results. This way, the analysis steps are caputed either in an IPython notebook or conventional python script. These steps, together with the original data, should then be sufficient to reproduce the results exactly.

1.1 Installation

Xanespy can be installed from the **python package index (PyPI) using pip**

```
$ pip install xanespy
```

or from anaconda.org using **conda**

```
$ conda install -c canismarko xanespy
```

1.2 X-Ray Absorbance Basics

Coming soon...

1.3 Example Workflow

A typical prcocedure for interacting with microscope frame-sets involves the following parts:

- Import the raw data
- Apply corrections and align the images
- Calculate some metric and create maps of it

- Visualize the maps, staticly or interactively.

Example for a single frameset across an X-ray absorbance edge:

```
import xanespy

# Example for importing from SSRL beamline 6-2c
xanespy.import_ssrl_frameset('<data_dir>', hdf_filename='imported_data.h5')

# Load a pre-defined XAS edge or create your own subclass xanespy.Edge
edge = xanespy.k_edges['Ni_NCA']
# Now load the newly created HDF5 file and the X-ray absorbance edge
fs = xanespy.XanesFrameset(filename='imported_data.h5', edge=edge)

# Perform automatic frame alignment
fs.align_frames(passes=5)
# Fit the absorbance spectra and extract the edge position (SLOW!)
fs.fit_spectra()

# Inspect the result with the built-in Qt5 GUI
fs.qt_viewer()
```

CHAPTER

TWO

IMPORTING DATA INTO XANESPY

The first step in any Xanespy workflow will be to **import the raw data into a common format**. These importer functions are written as needed: if your preferred beamline is not here, submit an issue.

2.1 APS Beamline 8-BM-B - Energy Stack (TXRM)

Note: Currently this function can only import one XANES stack. Time-resolved measurement is not implemented. If you would find this feature valuable, please submit an issue.

The Xradia microscope can save an entire stack in one `.txrm` file. This file can be imported using the *`import_aps_8BM_xanes_file()`* function. The list of energies is automatically extracted from the file. The reference frames will then reside in a different `.txrm` file.

Example usage:

```
import xanespy as xp

xp.import_aps_8BM_xanes_file('exp1-sample-stack.txrm',
                             ref_filename='exp1-reference_stack.txrm',
                             hdf_filename='txm-data.h5',
                             groupname='experiment1')
```

2.2 APS Beamline 8-BM-B - Directory of XRM Files

In-house XANES scan scripts often save a directory full of `.xrm` files with the metadata coding in the filenames. From the Xradia TXM at sector 8-BM-B, this XANES scan script can be generated with *`sector8_xanes_script()`*, and the results can then be imported with *`import_aps_8BM_xanes_dir()`*. The list of energies is automatically extracted from the filenames. The reference frames will also be identified in the directory.

Example usage:

```
import xanespy as xp

# First a script should be created with sector_8_xanes_script()
# Once the script is done, import the data with this function
xp.import_aps_8BM_xanes_dir("opearando_exp1/",
                            hdf_filename="operando_experiments.h5")
```

2.3 SSRL Beamline 6-2c - Directory of XRM Files

In-house XANES scan scripts often save a directory full of `.xrm` files with the metadata coding in the filenames. From the Xradia TXM at SSRL beamline 6-2c, this XANES scan script can be generated with the in-house generator, and the results can then be imported with *`import_ssrl_xanes_dir()`*. The list of energies is automatically extracted from the filenames. The reference frames will also be identified in the directory.

Example usage:

```
import xanespy as xp

# First a script should be created with sector_8_xanes_script()
# Once the script is done, import the data with this function
xp.import_ssrl_xanes_dir("opearando_exp1/",
                         hdf_filename="operando_experiments.h5")
```

2.4 Ptychography from 5.3.2.1 (ALS)

The output of the nanosurveyor reconstruction algorithm at 5.3.2.1 saves the data in h5 files. `import_nanosurveyor_frameset`() copies the reconstructed images and metadata from the individual files and combines them into a new HDF5 file for XAS analysis. The original CCD images are left in their original HDF5 files, so they should not be discarded.

```
import xanespy as xp

# This function copies the reconstructed images to a new file.
xp.import_nanosurveyor_frameset('NS_160529047/')
```

Given the slow nature of ptychography experiments, it may be necessary to capture an XAS scan into multiple chunks. Passing `append=True` to the importer allows **datasets to be combined**:

```
import xanespy as xp

# The first data-set is imported like normal except that the
# groupname and filename to save under are explicit.
xp.import_nanosurveyor_frameset('NS_160529047/',
                                hdf_filename='my_ptycho_data.h5',
                                hdf_groupname='my_combined_experiment')

# Now subsequent scans get the ``append=True`` argument
xp.import_nanosurveyor_frameset('NS_160529048/',
                                hdf_filename='my_ptycho_data.h5',
                                hdf_groupname='my_combined_experiment',
                                append=True)
xp.import_nanosurveyor_frameset('NS_160529049/',
                                hdf_filename='my_ptycho_data.h5',
                                hdf_groupname='my_combined_experiment',
                                append=True)
```

It may be necessary to only import a subset of the frames collected in a given directory. For example, if the last frame drifted out of the field-of-view and was re-collected in the next set of energies. The arguments `energy_range` and `exclude_re` can be used to fine-tune the list of importable files. See the documentation for *`import_nanosurveyor_frameset()`* for more details.

2.5 Xradia Image Files (.xrm and .txrm)

Xradia microscopes use the Microsoft OLE container format, which is not easily read[1]. Individual scan files are generally not that helpful anyway. But in case you need it, there are some adapters to `.xrm` and `.txrm` files, namely *xanespy.xradia.XRMFile* and *xanespy.xradia.TXRMFile*.

Note: The specification for `.xrm` files is not public, so these classes are reverse-engineered and may not be (definitely aren't) perfect. If you encounter problems, please submit an issue.

Opening xrm or txrm files is best done via the context manager:

```
import xanespy as xp
import numpy as np

# Single-image xrm file
with xp.XRMFile('my_txm_image.xrm') as f:
    img = f.image_data()
    assert img.ndim == 2 # (row, col)

# Multi-image txrm energy stack file
with xp.TXRMFile('my_txm_stack.txrm') as f:
    # Get images one at a time by index
    img = f.image_data(idx=0)
    assert img.ndim == 2 # (row, col)

    # Get images all at once in one big array
    stack = f.image_stack()
    assert stack.ndim == 3 # (prj, row, col)
    assert np.array_equal(img, stack[0])

    # Get X-ray energies for the images
    energies = f.energies()
    assert len(energies) == stack.shape[0]
```

The *XRMFile* and *TXRMFile* classes accept an optional `flavor` keyword argument. This option affects several pieces of metadata. See the *XRMFile* documentation for details.

[1] If you're shopping for a container format for your new data storage project, I would recommend AGAINST Microsoft OLE. This format stores data in raw binary, meaning that you need to know the encoding and structure to get meaningful data out. Instead, try **HDF5**: a nice open-source, well documented, type-aware format with bindings in many languages. It even plays nicely with numpy out of the box.

CHAPTER

THREE

ANALYZING THE DATA

3.1 Forking the Data

After long computations it can be helpful to create a copy of the dataset as a sort of checkpoint. To enable this, xanespy includes the `fork_data_group()` method: it creates a copy of the HDF group. The active set can be changed by setting the `data_name` attribute on a `XanesFrameset` object. Most operations enabled by the `XanesFrameset()` class are not idempotent, so starting from a clean dataset may be necessary:

```
import xanespy
# Select an imported hdf file to use
frameset = xanespy.XanesFrameset(hdf_filename="...")
# Make sure we're starting with clean data
frameset.data_name = "imported"

# Create a copy as a checkpoint
frameset.fork_data_group("aligned")
# Do some work that we're not sure will succeed
frameset.align_frames(passes=5)

# If the alignment doesn't work right,
# we can switch back to the original data and try again
frameset.data_name = "imported"
frameset.fork_data_group("aligned")
frameset.align_frames(passes=3)
```

The *fork_data_group()* method can be slow for large datasets. Xanespy will raise exceptions for non-sensical requests for forking: trying to copy a group onto itself, using a datagroup that doesn't exist, etc.

3.2 Frame Alignment

In order to acquire reliable spectra, **it is important that the frames be aligned properly**. Thermal expansion, motor slop, sample damage and imperfect microscope alignment can all cause frames to be misaligned. **It is often necessary to align the frames before performing any of the subsequent steps.**

This is done with the `xanespy.XanesFrameset().align_frames()` method:

```
import xanespy
# Select an imported hdf file to use
frameset = xanespy.XanesFrameset(hdf_filename="...")

# Run through five passes of the default phase correlation
frameset.align_frames(passes=5, plot_results=True)
```

Fig. 3.1: With the `plot_results` argument, a box and whisker plot is generated showing the distribution of corrections needed for aligning each frame. Several passes help reduce the error.

The alignments are generally done with subpixel resolution, which gives improved accuracy, but requires interpolation. To avoid problems with accumulated error, a cumulative translation matrix is kept and applied at the end to the original data. You can add your own translation manually using the `stage_transformations()` method. If `align_frames()` is called with `commit=False`, then the alignment parameters are added to `stage_transformations` but not applied. Once all transformations are staged, the `apply_transformations()` method will apply the cumulative transformation matrix and (by default) save the result to disk.

If the starting alignment is particularly sporadic, a false minimum can result in an exception or a very small image that doesn't provide useful information. In these cases, it may be necessary to first stage a template registration then perform several passes of phase correlation:

```
fs = XanesFrameset(hdf_filename="...")
# Eg. use the 22nd energy and a range of the image as the template
template = fs.frames()[21, 110:425, 150:450]
plt.imshow(template, cmap="gray")

fs.fork_data_group('aligned')

fs.align_frames(method="template_match", template=template, commit=False)
fs.align_frames(passes=5, commit=True)
```

3.3 Subtracting Surroundings

Some microscopes show differences in the absorbance of the whole frame, including background material. This can be removed from each frame, giving a better spectrum:

```
fs = XanesFrameset(hdf_filename="...")
fs.subtract_surroundings()
```

Fig. 3.2: The effect of the `subtract_surroundings()` method.

3.4 Spectrum Fitting - K-Edge

3.5 Spectrum Fitting - L-Edge

CHAPTER

FOUR

VISUALIZATION OF RESULTS

4.1 Plotting Maps

After calculating the maps, there will be a number of options for plotting. The *xanespy.xanes_frameset.XanesFrameset* object has a number of methods for plotting the maps. Many of these method use the `map_name` argument to The *xanespy.xanes_frameset.XanesFrameset.plot_map()* method will prepare a plot of any of the maps.

4.2 Interactive (Qt) Viewer

Xanespy includes a graphical user interface that allows for interactive visualization of X-ray frames and maps. To launch the viewer, prepare a `XanesFrameset` object and run the `qt_viewer()` method:

```
fs = XanesFrameset(...)
fs.qt_viewer()
```

The data tree on the left of the window shows the possible datasets than be viewed. Choosing an entry with type "frameset" will load and plot the frames, spectra and histograms in the frame window. If a "map" entry is selected, the map window will be launchedx and the frames that went into making the map will be shown in the frame window.

In the interest of encouraging reproducibility, the **ability to export plots has been intentionally left out**. Any options selected in the GUI can be passed into the `plot_map`, `plot_histogram` or `plot_spectrum` methods of the frameset object. The name of the entry in the data tree is given as the keyword argument `representation`.

Warning: The interactive viewer has no effect on the underlying frameset or accompanying data, with one exception: changing to a different top-level branch in the data tree will change the `XanesFrameset.data_name` attribute. Before proceeding with analysis, either revert the `data_name` attribute manually or create a new `XanesFrameset` object.

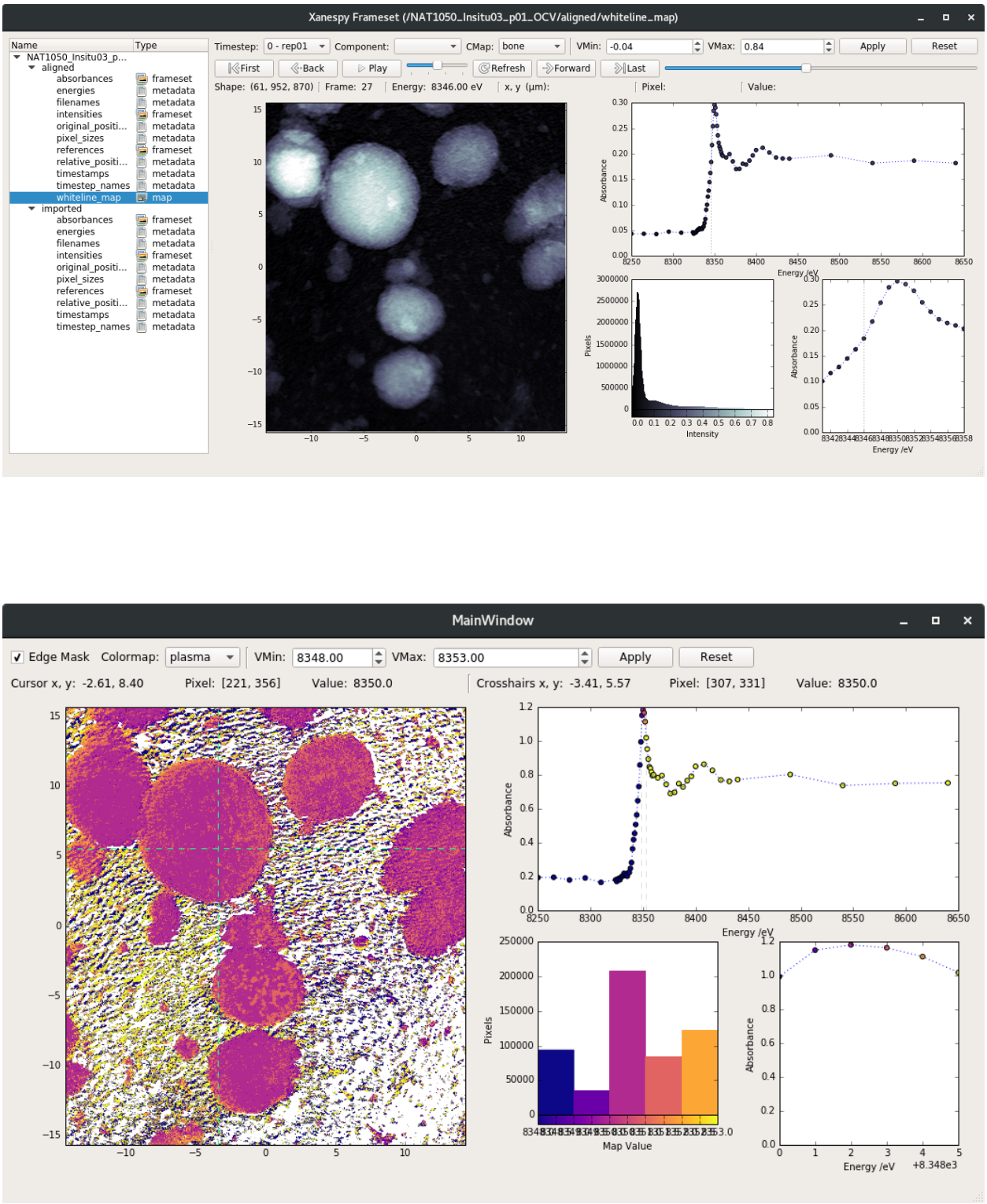

Fig. 4.1: Screenshots of the Qt viewer. Frame window (top) and map window (bottom).

CHAPTER

FIVE

ACCESSING THE DATA DIRECTLY

While the *XanesFrameset* class has methods for common tasks, sometimes it is necessary to access the data directly, as either numpy arrays or h5py datasets. The xanes_frameset has a *store()* method that returns an interface (`TXMStore`) to the underlying HDF5 file.

Warning: The `TXMStore` created by `xanes_frameset.store()` is attached to an open HDF5 file. It is strongly recommended to use the `with` statement described below. Otherwise make sure to call the store's `close()` method in a `try...except` block. File corruption is likely if not opened in this manner.

Call the following to get access to the associated datasets. Properties of the interface will return an HDF5 dataset in most cases.:

```
import xanespy as xp
frameset = xp.XanesFrameset(...)

# Open the TXMStore interface
with frameset.store() as store:
    # For example, the images are in (timestep, energy, row, column) order
    assert store.absorbances.shape == (10, 62, 1024, 1024)
    # Energies are in (timestep, energy) order
    assert store.energies.shape == (10, 62)
```

CHAPTER

SIX

XANESPY

6.1 xanespy package

6.1.1 Submodules

6.1.2 xanespy.beamlines module

Functions and classes that prepare experiments at specific synchrotron beamlines.

class xanespy.beamlines.**Detector**(*start: xanespy.beamlines.ZoneplatePoint*, *x_step: int = None*, *y_step: int = None*, *z_step: int = None*, *end: xanespy.beamlines.ZoneplatePoint = None*)

Bases: *xanespy.beamlines.Zoneplate*

A calibration object for the position of the detector.

class xanespy.beamlines.**DetectorPoint**(*x*, *y*, *z*, *energy*)

Bases: tuple

energy

Alias for field number 3

x

Alias for field number 0

y

Alias for field number 1

z

Alias for field number 2

class xanespy.beamlines.**Zoneplate**(*start: xanespy.beamlines.ZoneplatePoint*, *x_step: int = None*, *y_step: int = None*, *z_step: int = None*, *end: xanespy.beamlines.ZoneplatePoint = None*)

Bases: object

Type of focusing optic using in X-ray microscopy. It must be moved with changing energy to properly focus the beam. In order to properly predict zoneplate positions, it needs either two position-energy pairs or one position-energy pair and a step. Passing two position-energy pairs is preffered because this allows x, y and z to be set properly instead of just z.

Parameters

- **start** (*tuple*) – The first zoneplate position-energy pair.
- **z_step** (*int, optional*) – Adjustment in z-position for every positive change of 1 eV of beam energy.

- **end** (*tuple, optional*) – The second zoneplate position-energy pair.

position (*energy: float*)
Predict the x, y and z position of the zonplate for the given energy.

class xanespy.beamlines.**ZoneplatePoint** (*x*, *y*, *z*, *energy*)
Bases: tuple

energy
Alias for field number 3

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

xanespy.beamlines.**monitor_sector8** (*tsv_filename*)
Monitors a list of files and displays them as they are collected from the instrument. A matplotlib axes is displayed and is updated with each new frame that is detected. This function will block until all files in the file list are accounted for.

Parameters **tsv_filename** (*string*) – The name of the file that contains a tab-separated-values list of filenames. This file is automatically generated by the sector8_xanes_script function.

xanespy.beamlines.**sector8_xanes_script** (*dest*, *edge*, *zoneplate*, *detector*, *sample_positions*, *names*, *iterations=range(0, 1)*, *binning=1*, *exposure=30*, *abba_mode=True*)
Prepare an script file for running multiple consecutive XANES framesets on the transmission x-ray micrscope at the Advanced Photon Source beamline 8-BM-B. This function also creates a tab-separated-values (tsv) file which contains each sample filename and its associated meta-data. This can then be used for real-time processing.

Parameters

- **dest** – file-like object that will hold the resulting script
- **zoneplate** (Zoneplate) – Calibration details for the Fresnel zone-plate.
- **detector** (Detector) – Like zoneplate, but for detector.
- **edge** (KEdge) – Description of the absorption edge.
- **binning** (*int, optional*) – how many CCD pixels to combine into one image pixel (eg. 2 means 2x2 CCD pixels become 1 image pixel.
- **exposure** (*float*) – How many seconds to collect for per frame
- **sample_positions** – Locations to move the x, y (and z) axes to in order to capture the image.
- **names** – sample name to use in file names. Should match *sample_positions* in length.
- **iterations** (*iterable*) – contains an identifier for each full set of xanes location with reference.
- **abba_mode** (*str, optional*) – If True, script will locations forward and backwards to save time. Eg: reference, sample, change-energy, sample, reference, change-energy, etc. Not compatible with *frame_rest* argument. If used, the reference frame should be first or last in the order to make the process maximally efficient.

xanespy.beamlines.**ssrl6_xanes_script**(*dest, edge: edges.KEdge, zoneplate: xanespy.beamlines.Zoneplate, positions: typing.List[utilities.position], reference_position: utilities.position, iterations: typing.Iterable, iteration_rest: int = 0, frame_rest: int = 0, binning: int = 2, exposure=0.5, repetitions: int = 5, ref_repetitions: int = 10, abba_mode: bool = True*)

Prepare a script file for running multiple consecutive XANES framesets on the transmission x-ray micrscope at the Advanced Photon Source beamline 8-BM-B. Both *iteration_rest* and *frame_rest* can be used to give the material time to recover from X-ray damage.

Parameters

- **dest** – A file-like object that will hold the resulting script
- **edge** (*Edge*) – Description of the absorption edge.
- **binning** (*int, optional*) – how many CCD pixels to combine into one image pixel (eg. 2 means 2x2 CCD pixels become 1 image pixel.
- **exposure** (*float, optional*) – How many seconds to collect for per frame
- **positions** – Locations to move the x, y (and z) axes to in order to capture the image.
- **reference_position** (*tuple*) – Single x, y, z location to capture a reference frame.
- **iteration_rest** (*int, optional*) – Time (in seconds) to wait between iterations. Beam will wait at reference location before starting next XANES set.
- **frame_rest** (*int, optional*) – Time (in seconds) to wait between frames. Beam will wait at reference location before starting next energy frame.
- **zoneplate** (*Zoneplate*) – Calibration details for the Fresnel zone-plate.
- **detector** (*Detector*) – Like zoneplate, but for detector.
- **iterations** (*Iterable*) – Contains an identifier for each XANES dataset.
- **repetitions** (*int, optional*) – How many images to collect for each location/energy. These frames will then be averaged during analysis.
- **ref_repetitions** (*int, optional*) – Same as *repetitions* but for reference frames.
- **abba_mode** (*bool, optional*) – If True, script will alternate sample and reference locations first to save time. Eg: reference, sample, change-energy, sample, reference, change-energy, etc. Not compatible with *frame_rest* argument.

xanespy.beamlines.**write_scaninfo_header**(*f*, *abba_mode*, *repetitions*, *ref_repetitions*)

6.1.3 xanespy.edges module

Descriptions of X-ray energy absorption edge.

class xanespy.edges.**Edge**

Bases: object

An X-ray absorption edge. It is defined by a series of energy ranges. All energies are assumed to be in units of electron-volts. This class is intended to be extended into K-edge, L-edge, etc.

E_0

float – The energy of the absorption edge itself.

regions
list of 3-tuples – All the energy regions. Each tuple is of the form (start, end, step) and is inclusive at both ends.

name
string – A human-readable name for this edge (eg "Ni K-edge")

pre_edge
2-tuple – Energy range (start, stop) that defines points below the edge region, inclusive.

post_edge
2-tuple – Energy range (start, stop) that defines points above the edge region, inclusive.

map_range
2-tuple – Energy range (start, stop) used for normalizing maps. If not supplied, will be determine from pre- and post-edge arguments.

edge_range
2-tuple – Energy range (start, stop) used to determine the official beginning and edge of the edge itself.

E_0 = None

all_energies()
Combine all the regions into one array.

Returns **energies** – Flat array with the energies for this edge.

Return type list

edge_range = None

energies_in_range(*norm_range=None*)

map_range = None

post_edge = None

pre_edge = None

pre_edge_fit = None

regions = []

class xanespy.edges.**KEdge**
Bases: *xanespy.edges.Edge*

An X-ray absorption K-edge corresponding to a 1s transition.

annotate_spectrum(*ax*)
Draw lines on the axes to indicate the position of the edge.

mask(**args*, ***kwargs*)
Return a numpy array mask for material that's active at this edge. Calculations are done in `xanes_math.l_edge_mask()`.

class xanespy.edges.**LEdge**
Bases: *xanespy.edges.Edge*

An X-ray absorption K-edge corresponding to a 2s or 2p transition.

annotate_spectrum(*ax*)
Draw lines on the axes to indicate the position of the edge.

mask(**args*, ***kwargs*)
Return a numpy array mask for material that's active at this edge. Calculations are done in *xanes_math.l_edge_mask()*.

class xanespy.edges.**LMOMnKEdge**
Bases: *xanespy.edges.KEdge*

regions = [(6450, 6510, 20), (6524, 6542, 2), (6544, 6564, 1), (6566, 6568, 2), (6572, 6600, 4), (6610, 6650, 10), (6700, 6850,

class xanespy.edges.**NCACobaltLEdge**
Bases: *xanespy.edges.LEdge*

E_0 = 793.2

edge_range = (775, 785)

map_range = (0, 1)

post_edge = (785, 790)

pre_edge = (770, 775)

regions = [(770, 775, 1), (775, 785, 0.5), (785, 790, 1)]

class xanespy.edges.**NCANickelKEdge**
Bases: *xanespy.edges.KEdge*

E_0 = 8333

edge_range = (8341, 8358)

map_range = (8341, 8360)

post_edge = (8400, 8640)

pre_edge = (8249, 8320)

regions = [(8250, 8310, 20), (8324, 8344, 2), (8344, 8356, 1), (8356, 8360, 2), (8360, 8400, 4), (8400, 8440, 8), (8440, 8640, 5

shell = 'K'

class xanespy.edges.**NCANickelKEdge61**
Bases: *xanespy.edges.NCANickelKEdge*

regions = [(8250, 8310, 15), (8324, 8360, 1), (8360, 8400, 4), (8400, 8440, 8), (8440, 8640, 50)]

class xanespy.edges.**NCANickelKEdge62**
Bases: *xanespy.edges.NCANickelKEdge*

regions = [(8250, 8310, 15), (8324, 8360, 1), (8360, 8400, 4), (8400, 8440, 8), (8440, 8690, 50)]

class xanespy.edges.**NCANickelLEdge**
Bases: *xanespy.edges.LEdge*

E_0 = 853

edge_range = (848, 857)

map_range = (0, 1)

post_edge = (857, 862)

pre_edge = (844, 848)

regions = [(844, 848, 1), (849, 856, 0.25), (857, 862, 1)]

class xanespy.edges.**NMCNickelKEdge29**
Bases: *xanespy.edges.NCANickelKEdge*

regions = [(8250, 8310, 20), (8324, 8346, 6), (8346, 8358, 1), (8360, 8400, 10), (8400, 8440, 15), (8440, 8640, 100)]

6.1.4 xanespy.exceptions module

Define classes for more fine-grained control over exception handling.

exception xanespy.exceptions.**CreateGroupError**
Bases: ValueError

Tried to import a TXM frameset into a group but the corresponding HDF group already exists or is otherwise inaccessible.

exception xanespy.exceptions.**DataFormatError**
Bases: RuntimeError

The raw data are arranged in a way that the importers or TXM classes do not understand.

exception xanespy.exceptions.**DataNotFoundError**
Bases: FileNotFoundError

Expected a directory containing data but found none.

exception xanespy.exceptions.**DatasetExistsError**
Bases: RuntimeError

Trying to save a new dataset but one already exists with the given path.

exception xanespy.exceptions.**FileExistsError**
Bases: OSError

Tried to import a TXM frameset but the corresponding HDF file already exists.

exception xanespy.exceptions.**FilenameParseError**
Bases: ValueError

The parameters in the filename do not match the naming scheme associated with this flavor.

exception xanespy.exceptions.**FrameFileNotFound**
Bases: OSError

Expected to load a TXM frame file but it doesn't exist.

exception xanespy.exceptions.**GroupKeyError**
Bases: KeyError

Tried to load or create an HDF group but failed. Examples include: the group doesn't exist, is ambiguous or already exists when being created.

exception xanespy.exceptions.**HDFScopeError**
Bases: ValueError

Tried to pass an HDF scope that is not recognized.

exception xanespy.exceptions.**NoParticleError**
Bases: Exception

exception xanespy.exceptions.**RefinementError**
Bases: RuntimeError

exception xanespy.exceptions.**XanesMathError**
Bases: RuntimeError

6.1.5 xanespy.importers module

xanespy.importers.**decode_aps_params**(*filename*)
Accept the filename of an XRM file and return sample parameters as a dictionary.

xanespy.importers.**decode_ssrl_params**(*filename*)
Accept the filename of an XRM file and return sample parameters as a dictionary.

xanespy.importers.**import_aps_8BM_xanes_dir**(*directory*, *hdf_filename*, *quiet=False*)

xanespy.importers.**import_aps_8BM_xanes_file**(*filename*, *ref_filename*, *hdf_filename*, *groupname=None*)
Extract an entire xanes framestack from one xradia file.

A single TXRM file can contain multiple frames at different energies. This function will import such a file along with the corresponding reference frames into an HDF file. If the given `groupname` exists in `hdf_filename`, it will be overwritten and a `RuntimeWarning` will be issued.

Parameters

- **filename** (*str*) – File path to the txrm file to import.
- **ref_filename** (*str*) – File path to the txrm file that contains the white-field reference images. This will be used to calculate optical depth from transmitted intensity.
- **hdf_filename** (*str*) – File path to the destination HDF5 file that will receive the imported data.
- **groupname** (*str, optional*) – The name for the top-level HDF group. If omitted, a group name will be generated from the `filename` parameter.

xanespy.importers.**import_frameset**(*directory*, *flavor*, *hdf_filename*, *return_val=None*)
Import all files in the given directory collected at an X-ray microscope beamline.

Images are assumed to full-field transmission X-ray micrographs.

If `return_val` is "group", the return value for this function will be a data group in an **open** HDF5 file. The underlying file should be explicitly closed to avoid corruption. This is done automatically if `return_val` is None (default).

Parameters

- **directory** (*str*) – A valid path to a directory containing the frame data to import.
- **flavor** (*str*) – Indicates what type of naming conventions and data structure to assume. See documentation for `xanespy.xradia.XRMFile` for possible choice.
- **hdf_filename** (*str*) – Where to save the output to. An exception is throw if this file already exists.
- **return_val** (*str*) – Request a specific return value. - None: No return value - "group": The open HDF5 group for this experiment.

xanespy.importers.**import_nanosurveyor_frameset**(*directory: str*, *quiet=False*, *hdf_filename=None*, *hdf_groupname=None*, *energy_range=None*, *exclude_re=None*, *append=False*)
Import a set of images from reconstructed ptychography scanning microscope data.

This generates ptychography chemical maps based on data collected at ALS beamline 5.3.2.1. The arguments `energy_range` and `exclude_re` can be used to fine-tune the set of imported file. For example: passing `exclude_re='(019|017)'` will import everything except scans 019 and 017.

Parameters

- **directory** (*str*) – Directory where to look for results. It should contain .cxi files that are the output of the ptychography reconstruction."
- **quiet** (*Bool, optional*) – If truthy, progress bars will not be shown.

- **hdf_filename** (*str, optional*) – HDF File used to store computed results. If omitted or None, the *directory* basename is used
- **hdf_groupname** (*str, optional*) – Name to use for the hdf group of this dataset. If omitted or None, the *directory* basename is used. Raises an exception if the group already exists in the HDF file.
- **energy_range** (*2-tuple, optional*) – A 2-tuple with the (min, max) energy to be imported. This is useful if only a subset of the available data is usable. Values are assumed to be in electron-volts.
- **exclude_re** (*str, optional*) – Any filenames matching this regular expression will not be imported. A string or compiled re object can be given.
- **append** (*bool, optional*) – If True, any existing dataset will be added to, rather than replaced (default False)

xanespy.importers.**import_ssrl_xanes_dir**(*directory*, *hdf_filename*)

Import all files in the given directory collected at SSRL beamline 6-2c and process into framesets. Images are assumed to full-field transmission X-ray micrographs and repetitions will be averaged. Passed on to `xanespy.importers.import_frameset`

Parameters

- **directory** (*str*) – Where to look for files to import.
- **hdf_filename** (*str*) – Path to an HDF5 to receive the data.

xanespy.importers.**import_stxm_frameset**(*directory: str*, *quiet=False*, *hdf_filename=None*, *hdf_groupname=None*, *energy_range=None*, *exclude_re=None*, *append=False*)

Import a set of images from scanning microscope data.

This generates Scanning Tranmission X-ray Microscopy chemical maps based on data collected at ALS beamline 5.3.2.1

Parameters

- **directory** (*str*) – Directory where to look for results. It should contain .hdr and xim files that are the output of the ptychography reconstruction."
- **quiet** (*Bool, optional*) – If truthy, progress bars will not be shown.
- **hdf_filename** (*str, optional*) – HDF File used to store computed results. If omitted or None, the *directory* basename is used
- **hdf_groupname** (*str, optional*) – Name to use for the hdf group of this dataset. If omitted or None, the *directory* basename is used. Raises an exception if the group already exists in the HDF file.
- **energy_range** (*2-tuple, optional*) – A 2-tuple with the (min, max) energy to be imported. This is useful if only a subset of the available data is usable. Values are assumed to be in electron-volts.
- **exclude_re** (*str, optional*) – Any filenames matching this regular expression will not be imported. A string or compiled re object can be given.
- **append** (*bool, optional*) – If True, any existing dataset will be added to, rather than replaced (default False)

xanespy.importers.**magnification_correction**(*frames*, *pixel_sizes*)

Correct for changes in magnification at different energies.

As the X-ray energy increases, the focal length of the zone plate changes and so the image is zoomed-out at higher energies. This method applies a correction to each frame to make the magnification similar to that of the first frame. Some beamlines correct for this automatically during acquisition and don't need this function: APS 8-BM-B, 32-ID-C.

Parameters

- **frames** (*np.ndarray*) – Numpy array of image frames that need to be corrected.
- **pixel_sizes** (*np.ndarray*) – Numpy array of pixel sizes corresponding to entries in *frames*.

Returns **(scales2D, translations)** – An array of scale factors to use for applying a correction to each frame. Translations show how much to move each frame array by to re-center it.

Return type (np.ndarray, np.ndarray)

xanespy.importers.**read_metadata**(*filenames*, *flavor*)

Take a list of filenames and return a pandas dataframe with all the metadata.

Parameters

- **filenames** (*iterable*) – Iterable of filenames to use for extracting metadata.
- **flavor** (*str*) – Same as in `import_frameset`.

6.1.6 xanespy.plots module

Helper functions for setting up and displaying plots using matplotlib.

xanespy.plots.**big_axes**()

Return a new Axes object, but larger than the default.

xanespy.plots.**draw_colorbar**(*ax*, *cmap*, *norm*, *energies*, *orientation='vertical'*, **args*, ***kwargs*)

Draw a colorbar on the side of a mapping axes to show the range of colors used. Returns the newly created colorbar object.

Parameters

- **ax** (–) –
- **cmap** (–) – to use.
- **norm** (–) – use.
- **energies** (–) – colorbar.

xanespy.plots.**draw_histogram_colorbar**(*ax*, **args*, ***kwargs*)

Similar to *draw_colorbar()* with some special formatting options to put it along the X-axis of the axes.

xanespy.plots.**dual_axes**(*fig=None*, *longdim=13.8*, *shortdim=6.9*, *orientation='horizontal'*)

Two new axes for mapping, side-by-side.

Parameters

- **longdim** (*float*) – Size in inches for the long dimension. If orientation is "vertical", this will be the height.
- **shortdim** (*float*) – Size in inches for the short dimension. If orientation is "vertical", this will be the width.

xanespy.plots.**new_axes**(*height=5*, *width=None*)

Create a new set of matplotlib axes for plotting. Height in inches.

xanespy.plots.**new_image_axes**(*height=5*, *width=5*)
Square axes with ticks on the outside.

xanespy.plots.**plot_composite_map**(*data*, *ax=None*, *origin='upper'*, **args*, ***kwargs*)
Plot an RGB composite map on the given axes.

xanespy.plots.**plot_pixel_spectra**(*pixels*, *extent*, *spectra*, *energies*, *map_ax*, *spectra_ax=None*, *step_size=0*)
Highlight certain pixels in an already-plotted map and plot their spectra. The map should already have been plotted.

Parameters

- **pixels** (–) – to highlight and plot.
- **extent** (–) – positions to (row, column) positions.
- **spectra** (–) – in *pixels* will use this array to get spectra. Shape is assumed to be (row, column, energy).
- **energies** (–) – plotting spectra.
- **map_ax** (–) – pixels.
- **spectra_ax** (–) – None (default) a new axes will be created.
- **step_size** (–) – directly on top of each other.

xanespy.plots.**plot_txm_histogram**(*data*, *ax=None*, *norm=None*, *bins=None*, *cmap='plasma'*, *add_cbar=True*, **args*, ***kwargs*)
Take an array of data values and show a histogram with some color-coding related to normalization value.

Returns: The matplotlib axes object used for plotting.

Parameters

- **data** (*np.ndarray*) – An array of values to plot on the histogram.
- **ax** (*optional*) – Matplotlib Axes instance to receive the plot. If None, a new axes will created.
- **norm** (*optional*) – Matplotlib Normalize instance with the colormap range.
- **bins** (*optional*) – Bins to pass to the matplotlib hist() routine. If None (default), we will choose based on dtype of the data: integers will yield 1-wide bins, anything else will give 256 bins.
- **cmap** (*str, optional*) – Matplotlib colormap for coloring the bars.
- **add_cbar** (*bool, optional*) – Boolean to decide whether to add a colorbar along the bottom axis or not.
- ***args** – Positional arguments passed to matplotlib's *hist* call.
- ****kwargs** – Keyword arguments passed to matplotlib's *hist* call.

xanespy.plots.**plot_txm_intermediates**(*images*)
Accept a dictionary of images and plots them each on its own axes using matplotlib's *imshow*. This is a complement to routines that operate on a microscopy frame and optionally return all the intermediate calculated frames.

xanespy.plots.**plot_txm_map**(*data*, *edge=None*, *norm=None*, *ax=None*, *cmap='plasma'*, *origin='upper'*, *vmin=None*, *vmax=None*, **args*, ***kwargs*)

xanespy.plots.**plot_xanes_spectrum**(*spectrum*, *energies*, *norm=<matplotlib.colors.Normalize object>*, *show_fit=False*, *ax=None*, *ax2=None*, *linestyle=':'*, *color='blue'*, *cmap='plasma'*, **args*, ***kwargs*)

Plot a XANES spectrum on an axes. Applies some color formatting if *edge* is a valid XANES Edge object.

Parameters

- **spectrum** (–) –
- **energies** (–) –
- **norm** (–) – range. This will be used to annotate the plot if it is give.
- **show_fit** (–) –
- **ax** (–) – will be generated.
- **ax2** (–) – the data are complex.
- **linestyle** (–) –
- **cmap** (–) –
- **color** (–) – or "y" will decide based on the numerical value, *norm* and *cmap* arguments. Anything else will be passed as a color spec to the matplotlib commands.

xanespy.plots.**remove_extra_spines**(*ax*)

Removes the right and top borders from the axes.

xanespy.plots.**set_axes_color**(*ax*, *color*)

Set the axes, tick marks, etc of *ax* to mpl color *color*. Also, "doegreen" has special significance as the color associated with the US department of energy.

xanespy.plots.**set_outside_ticks**(*ax*)

Convert all the axes so that the ticks are on the outside and don't obscure data.

6.1.7 xanespy.qt_frame_view module

class xanespy.qt_frame_view.**FrameAnimation**(*fig*, *artists*, **args*, ***kwargs*)

Bases: matplotlib.animation.ArtistAnimation

Performs the animation for scrolling through frames arbitarily.

stop()

class xanespy.qt_frame_view.**FrameChangeSource**(*view*, **args*, ***kwargs*)

Bases: PyQt5.QtCore.QObject

add_callback(*func*, **args*, ***kwargs*)

callbacks = []

remove_callback(*func*, **args*, ***kwargs*)

start()

stop()

class xanespy.qt_frame_view.**QtFrameView**

Bases: PyQt5.QtCore.QObject

add_hdf_tree_item(*item*)

clear_axes()

connect_signals(*presenter*)

create_canvas()

create_status_bar()

disable_frame_controls(*status*)

disable_plotting_controls(*status*)

draw_frames

draw_histogram

draw_spectrum(*spectrum*, *energies*, *norm*, *cmap*, *edge_range*)

edge_ax = None

expand_hdf_tree

fig = None

frame_changed

frame_controls
Gives a list of all the UI buttons that are associated with changing the currently active frame.

hist_ax = None

hist_cb = None

plotting_controls
Gives a list of all the UI elements that are associated with changing how the frameset is plotted.

select_active_hdf_item(*item*)

set_cmap(*cmap*)

set_cmap_list(*cmap_list*)

set_component(*comp*)

set_component_list(*component_list*)

set_drawing_status(*status*)

set_slider_max(*val*)

set_status_cursor(*msg*)

set_status_energy(*msg*)

set_status_index(*msg*)

set_status_pixel(*msg*)

set_status_shape(*msg*)

set_status_unit(*msg*)

set_status_value(*msg*)

set_timestep(*idx*)

set_timestep_list(*timestep_list*)

set_ui_enabled(*enable=True*)
Turn on (default) or off the main interactive elements in the frame window. Useful for indicating blocking operations.

set_vmax(*val*)

set_vmax_decimals(*val*)

set_vmax_minimum(*val*)

set_vmax_step(*val*)

set_vmin(*val*)

set_vmin_decimals(*val*)

set_vmin_maximum(*val*)

set_vmin_step(*val*)

set_window_title(*title*)

setup()

show()

show_status_message(*message*)

spectrum_ax = None

ui = None

use_busy_cursor(*status*)

window = None

6.1.8 xanespy.qt_frameset_presenter module

class xanespy.qt_frameset_presenter.**QtFramesetPresenter**(*frameset*, *frame_view*, **args*, ***kwargs*)

Bases: PyQt5.QtCore.QObject

Presenter for showing XanesFrameset frames and maps via Qt.

app_ready
pyqtSignal – Emitted when the application has been created and is ready for drawing.

busy_status_changed
pyqtSignal – Emitted when processing has started or ended.

map_data_changed
pyqtSignal – Emitted when the map data is different and should be re-plotted.

map_data_cleared
pyqtSignal – Emitted when no map data is available and plots can be cleared.

active_component = 'real'

active_frame = 0

active_frames()

active_map()
Returns the active map array if possible. If it doesn't exist, return None.

active_representation = None

active_timestep = 0

add_frame_view(*view*, *threaded=True*)
Attach a view to this presenter.

Parameters

- **view** (*QObject*) – The view will be connected to signals that describe changes in frame data.
- **threaded** (*bool, optional*) – If true, this view will be added to its own thread before signals get connected

add_map_view(*view*, *threaded=True*)
Attach a view to this presenter.

Parameters

- **view** (*QObject*) – The view will be connected to signals that describe changes in map data.
- **threaded** (*bool, optional*) – If true, this view will be added to its own thread before signals get connected, giving a snapier UI. Disabling makes testing more straightforward.

animate_frames()

app_ready

build_hdf_tree()
Build the items and insert them into the view's HDF tree based on the structure of the frameset's HDF file.

busy_status_changed

change_cmap(*new_cmap*)

change_component(*new_comp*)

change_hdf_group(*new_item*, *old_item*)

change_map_cmap(*new_cmap*)

cmap_list_changed

component_list_changed

connect_signals()

create_app()

draw_frame_histogram()

draw_frame_spectra()

first_frame()

frame_cmap = 'copper'

frame_norm()

frame_pixel = None

hover_frame_pixel(*xy*)

last_frame()

launch()

map_cmap = 'plasma'

map_cursor_changed

map_data_changed

map_data_cleared

map_limits_changed

map_norm()

map_pixel_changed

map_spectrum_changed

mean_spectrum_changed

move_map_pixel(*vert*, *horiz*)
Move the active pixel by the given amount in each direction.

If the current pixel is not active, this is a no-op.

move_slider(*new_value*)

next_frame()

num_frames = 0

play_frames(*start*)

prepare_ui()

previous_frame()

process_events

refresh_frames()

reset_frame_range()
Reset the frame plotting vmin and vmax based on the currently selected data. VMin will be the 1nd percentile and VMax will be the 99th percentile. The results can be accessed by calling the *frame_norm()* method.

Returns

- **(vmin, vmax)** ((*float, float*)) – A 2-tuple with the new min and max range.
- *None* – If current representation is not a valid map, None will be returned.

reset_map_range()
Reset the map plotting vmin and vmax based on the currently selected data. VMin will be the 1st percentile and VMax will be the 99th percentile. This method also caches the values so they can be retrieved via the *map_norm()* method. If the current representation is not valid map data, then the cached vmin and vmax will be returned with no other changes.

Returns **(vmin, vmax)** – A 2-tuple with the current min and max range.

Return type (float, float)

set_frame_vmax(*new_value*)

set_frame_vmin(*new_value*)

set_map_cursor(*x*, *y*)

set_map_pixel(*x*, *y*)

set_map_vmax(*new_value*)

set_map_vmin(*new_value*)

set_play_speed(*new_speed*)
Change how fast the play timer ticks. Input speeds should be in the range of 0, 30 with 30 being the fastest. This is converted to timer intervals between 1ms and 1000ms on a exponential scale.

set_timestep(*new_timestep*)

show_spectrum_fit = False

timer = None

toggle_edge_mask(*state*)

toggle_spectrum_fit(*state*)

update_frame_range_limits()
Check that the (min, max, step) of the frame spinboxes are reasonable. This should be called when the spinbox values change.

update_map_limits()

update_maps()
Send the current mapping data to the *map_data_changed* signal.

This method should be called after anything changes the visual representation of the map. Examples:

- Changing the active representation
- Changing the colormap
- Changing the map normalization limits
- Changing whether the XAS edge mask is applied

update_spectra()
Get the most recent data for mean and single-pixel spectra and send them out to the signals *mean_spectrum_changed* and *pixel_sepctrum_changed*.

update_status_frame(*new_frame*)
Create a string (and send it to the UI) that indicates the energy of the requested frame.

update_status_shape()

update_status_unit()

update_status_value()

use_edge_mask = False

6.1.9 xanespy.qt_map_view module

class xanespy.qt_map_view.**QtMapView**
Bases: PyQt5.QtCore.QObject

A Qt view for a frameset map. It should be controlled by a presenter.

cmap_changed
signal – Fires when the user changes the colormap via the UI

limits_applied
signal – Fires when the user requests that data be redrawn with new limits.

limits_reset
signal – Fires when the user asks that the norm limits be reset to the data.

cmap_changed

connect_presenter(*presenter*)
Connect to signals for changed presenter state.

create_canvas()

crosshairs = None

edge_mask_toggled

fig = None

hide()

keyboard_nav(*event*)

latest_cmap = 'plasma'

limits_applied

limits_reset

map_clicked

map_hovered

map_moved

map_vmax_changed

map_vmin_changed

mouse_clicked_canvas(*mouse_event*)

mouse_in_canvas(*mouse_event*)

plot_histogram_data(*map_data*, *norm*, *cmap*, *extent*)

plot_map_data(*map_data*, *norm*, *cmap*, *extent*)

plot_spectrum(*spectrum*, *fitted_spectrum*, *norm*, *cmap*, *edge_range*)

redraw_canvas()

redraw_crosshairs(*xy*)
Draw a set of crosshairs on the map at location given by *xy*.

set_cmap_list(*new_list*)

set_map_limits(*vmin*, *vmax*, *step*, *decimals*)

setup_ui()

show()

spectrum_fit_toggled

ui = None

update_crosshair_labels(*xy*, *pixel*, *value*)

update_cursor_labels(*xy*, *pixel*, *value*)

window = None

6.1.10 xanespy.txmstore module

Tools for accessing TXM data stored in an HDF5 file.

class xanespy.txmstore.**TXMStore**(*hdf_filename: str*, *parent_name: str*, *data_name=None*, *mode='r'*)

Bases: object

Wrapper around HDF5 file that stores TXM data. It has a series of properties that return the corresponding HDF5 dataset object; the TXMStore().attribute.value pattern can be used to get pure numpy arrays. These objects should be used as a context manager to ensure that the file is closed, especially if using a writing mode:

with TXMStore() as store: # Do stuff with store here

Parameters

- **hdf_filename** (*str*) – Path to the HDF file to be used.
- **parent_name** (*str*) – Name of the top-level HDF5 group.
- **data_name** (*str*) – Name of the second level HDF5 group, used for specific data iterations (eg. imported, aligned)
- **mode** (*str*) – Eg. 'r' for read-only, 'r+' for read-write. Passed directly to h5py.File constructor.

VERSION = 1

absorbance_mean

absorbances

close()

cluster_map

data_group()
Retrieve the currently active second-level HDF5 group object for this file and groupname. Ex. "imported" or "aligned_frames".

data_name

data_tree()
Create a tree of the possible groups this store could access. The first level is samples, then data_groups (ie. same sample but different analysis status), then representations. Maps are not included in this tree.

energies

filenames

fit_parameters

fork_data_group(*dest*, *src=None*)
Turn on different active data group for this store. This method deletes the existing group and copies symlinks from the current one.

get_dataset(*name*)
Attempt to open the requested dataset.

Returns **data** – An open HDF5 dataset

Return type hyp5.Dataset

Raises exceptions.GroupKeyError – If the dataset does not exist in the file.

get_frames(*name*)
Get a set of frames, specified by the value of *name*.

get_map(*name*)
Get a map of the frames, specified by the value of *name*.

has_dataset(*name*)
Return a boolean indicated whether this dataset exists in the HDF file.

intensities

latest_data_name

original_positions

parent_group()
Retrieve the top-level HDF5 group object for this file and groupname.

particle_labels

pixel_sizes

pixel_unit

references

relative_positions
(x, y, z) position values for each frame.

replace_dataset(*name*, *data*, *context=None*, *attrs={}*, *compression=None*, **args*, ***kwargs*)
Wrapper for h5py.create_dataset that removes the existing dataset if it exists.

Parameters

- **name** (*str*) – HDF5 groupname name to give this dataset.
- **data** (*np.ndarray*) – Numpy array of data to be saved.
- **context** (*str, optional*) – Specifies what kind of data is stored. Eg. "frameset", "metadata", "map".
- **attrs** (*dict, optional*) – Dictionary containing HDF5 metadata attributes to be set on the resulting dataset.
- ***args** – Arguments to pass to h5py's `create_dataset` method.
- ****kwargs** – Keyword arguments to pass to h5py's `create_dataset` method.

set_frames(*name*, *val*)
Set data for a set of frames, specificied by the value of *name*.

signal_map

signal_method
String describing how the previously extracted signals were calculated.

signal_weights
Get the pixel weights of the previously extracted signals using any one of a variety of decomposition methods, saved as *signal_method*.

signals
Get the previously extracted signals using any one of a variety of decomposition methods, saved as *signal_method*.

timestamps

timestep_names

whiteline_fit

whiteline_max

6.1.11 xanespy.utilities module

A collection of classes and functions that arent' specific to any one type of measurement. Also, it defines some namedtuples for describing coordinates.

xanespy.utilities.**Extent**
alias of extent

class xanespy.utilities.**Pixel**(*vertical*, *horizontal*)
Bases: tuple

horizontal
Alias for field number 1

vertical
Alias for field number 0

xanespy.utilities.**broadcast_reverse**(*array*, *shape*, **args*, ***kwargs*)
Take the array and extends it as much as possible to match *shape*. Similar to numpy's broadcast_to function, but starts with the most significant axis. For example, if *array* has shape (7, 29), it can be broadcast to (7, 29, 1024, 1024).

xanespy.utilities.**foreach**(*f*, *l*, *threads=4*, *return_=False*)
Apply f to each element of l, in parallel

xanespy.utilities.**get_component**(*data*, *name*)
If complex, turn to given component, otherwise return original data.

xanespy.utilities.**is_kernel**()
Detect whether or not we're running inside an IPython kernel. NB: This does not distinguish between eg IPython notebook and IPython QtConsole.

xanespy.utilities.**parallel_map**(*f*, *l*, *threads=4*)

xanespy.utilities.**pixel_to_xy**(*pixel*, *extent*, *shape*)
Take an xy location on an image and convert it to a pixel location suitable for numpy indexing.

class xanespy.utilities.**position**(*x*, *y*, *z*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

z
Alias for field number 2

xanespy.utilities.**prog**(*iterable=None*, *leave=None*, *dynamic_ncols=True*, **args*, ***kwargs*)
A progress bar for displaying how many iterations have been completed.

This is mostly just a wrapper around the tqdm library. args and kwargs are passed directly to either tqdm.tqdm or tqdm.tqdm_notebook. This function also takes into account the value of USE_PROG defined in this module. If tqdm is no installed, then calls to prog will just return the iterable again.

Parameters

- **iterable** – Iterable to decorate with a progressbar. See `tqdm.tqdm` documentation for more details.
- **leave** (*bool, optional*) – Whether to leave the progress bar in the stream after it's completed. If omitted, will depend on terminal used.
- **dynamic_ncols** (*bool, optional*) – Whether to adapt dynamically to the width of the environment.
- **args** – Positional arguments passed directly to `tqdm` or `tqdm_notebook`.
- **kwargs** – Keyword arguments passed directly to `tqdm` or `tqdm_notebook`.

class xanespy.utilities.**shape**(*rows*, *columns*)
Bases: tuple

columns
Alias for field number 1

rows
Alias for field number 0

xanespy.utilities.**xy_to_pixel**(*xy*, *extent*, *shape*)
Take an xy location on an image and convert it to a pixel location suitable for numpy indexing.

class xanespy.utilities.**xycoord**(*x*, *y*)
Bases: tuple

x
Alias for field number 0

y
Alias for field number 1

6.1.12 xanespy.xanes_frameset module

Class definitions for working with a whole stack of X-ray microscopy frames. Each frame is a micrograph at a different energy. A frameset then is a three-dimenional dataset with of dimenions (energy, row, column).

class xanespy.xanes_frameset.**PtychoFrameset**(*filename*, *edge*, *groupname=None*)
Bases: *xanespy.xanes_frameset.XanesFrameset*

A set of images ("frames") at different energies moving across an absorption edge. The individual frames should be generated by ptychographic reconstruction of scanning transmission X-ray microscopy (STXM) to produce an array complex intensity values. This class does *not* include any code responsible for the collection and reconstruction of such data, only for the analysis in the context of X-ray absorption near edge spectroscopy.

representations()
Retrieve a list of valid representations for these data, such as modulus or phase data for ptychography.

class xanespy.xanes_frameset.**XanesFrameset**(*filename*, *edge*, *groupname=None*)
Bases: object

A collection of TXM frames at different energies moving across an absorption edge. Iterating over this object gives the individual Frame() objects. The class assumes that the data have been imported into an HDF file.

active_group = ''

align_frames(*reference_frame='mean'*, *blur=None*, *method: str = 'cross_correlation'*, *template=None*, *passes=1*, *commit=True*, *component='modulus'*, *plot_results=True*)
Use cross correlation algorithm to line up the frames. All frames will have their sample position set to (0, 0) since we don't know which one is the real position. This operation will interpolate between pixels so

introduces error. If multiple passes are performed, the translations are saved and combined at the end so this error is only introduced once. Using the *commit=False* argument allows for multiple different types of registration to be performed in sequence, since uncommitted translations will be applied before the next round of registration.

Parameters

- **(int, stror None)** (*reference_frame*) – which all other frames should be aligned. If None, the frame of highest intensity will be used. If "mean" (default) or "median", the average or median of all frames will be used. If "max", the frame with highest absorbance is used. This attribute has no effect if template matching is used.
- **blur** (*A type of filter to apply to each frame of the data*) – before attempting registration. Choices are "median" or None
- **method** (*Which technique to use to calculate the translation*) –
  – "cross_correlation" (default)
  – "template_match"
  (If "template_match" is used, the *template* argument should also be provided.)
- **passes** (*How many times this alignment should be done. Default: 1.*) –
- **template** (*Image data that should be matched if the*) – *template_match* method is used.
- **commit** (*If truthy (default), the final translation will be*) – applied to the data stored on disk by calling *self.apply_translations(crop=True)* after all passes have finished.
- **component** (*What component of the data to use: 'modulus',*) – 'phase', 'imag' or 'real'.
- **plot_results** (*If truthy (default), plot the root-mean-square of the*) – translation distance for each pass.

apply_internal_reference(*plot_background=True*, *ax=None*)

Use a portion of each frame for internal reference correction. The result is the complex refraction for each pixel: the real component describes the phase shift, and the imaginary component is exponential decay, ie. absorbance.

Parameters

- **plot_background** (*If truthy, the values of I_0 are plotted as*) – a function of energy.
- **ax** (The axes to use for plotting if *plot_background* is) – truthy.

apply_transformations(*crop=True*, *commit=True*)

Take any transformations staged with *self.stage_transformations()* and apply them. If commit is truthy, the staged transformations are reset.

Returns: Transformed array of the absorbances frames.

Parameters

- **crop** (–) –
- **so there are not edges. If falsy, the images will** (*translated,*) –
- **wrapped.** (*be*) –

- **commit** (–) – store for absorbances, intensities and references, and the staged transformations will be cleared. Otherwise, only the absorbance data will be transformed and returned.
- **frames_name** (–) – transformation too (eg. 'absorbances')

calculate_maps(*fit_spectra=False*)

Generate a set of maps based on pixel-wise Xanes spectra: whiteline position, particle labels.

Parameters **-fit_spectra** (*If truthy, the whiteline will be found by*) – fitting curves, instead of the default of taking the direct maximum. This is likely to be very slow.

calculate_signals(*n_components=2*, *method='nmf'*, *frame_source='absorbances'*, *edge_mask=True*)

Extract signals and assign each pixel to a group, then save the resulting RGB cluster map.

Parameters

- **n_components** (*int, optional*) – The number of signals and number of clusters into which the data will be separated.
- **method** (*str, optional*) – The technique to use for extracting signals. Currently only "nmf" is supported.
- **frame_source** (*str, optional*) – Name of the frame-set to use as the input data.
- **edge_mask** (*bool, optional*) – If truthy (default), only those pixels passing the edge filter will be considered.

calculate_whitelines(*edge_mask=False*)

Calculate and save a map of the whiteline position of each pixel by calculating the energy of simple maximum absorbance.

Parameters **edge_mask** (–) – be fit and the remaning pixels will be set to a default value. This can help reduce computing time.

clear_caches()

Clear cached function values so they will be recomputed with fresh data

cmap = 'plasma'

components()

Retrieve a list of valid representations for these data.

data_name

data_tree()

Wrapper around the TXMStore.data_tree() method.

edge_mask

Calculate a mask for what is likely active material at this edge.

Parameters

- **sensitivity** (*float*) – A multiplier for the otsu value to determine the actual threshold.
- **min_size** (*int*) – Objects below this size (in pixels) will be removed. Passing zero (default) will result in no effect.

endtime(*timeidx=None*)

Determine the latest timestamp amongst all of the frames.

Parameters **timeidx** (*int, optional*) – Which timestep to use for finding the end time. If omitted or None (default), all timesteps will be checked.

Returns **start_time** – Naive datetime representing the latest known frame for this time index. Timezone will always be UTC no matter where the data were collected.

Return type np.datetime64

energies

Return the array of beam energies for the given time index.

Returns **energies** – A 1-dimensional array with the energy for each frame.

Return type np.ndarray

extent

Determine physical dimensions for axes values.

If an index is given, it will first be applied to the frames array. For any remaining dimensions besides the last two, the median will be taken. For an array of extents for each frame, use the `extent_array` method.

Parameters

- **representation** (*str, optional*) – Name for which dataset to use.
- **idx** (*int, optional*) – Index for choosing a frame. Any valid numpy index is allowed, eg. `...` (default) uses all frame.

Returns **extent** – The spatial extent for the frame with order specified by `utilities.Extent`

Return type tuple

extent_array(*representation='intensities'*)

fit_spectra(*edge_mask=True*)

Fit a series of curves to the spectrum at each pixel.

For anything other than trivially small data-sets, this method can take a very long time. To speed up processing, MPI calls are used. Calling this function with mpiexec will allow for more parallel processing.

Parameters **edge_mask** (*bool, optional*) – If true, only pixels passing the edge_mask will be fit and the remaning pixels will be set to a default value. This can help reduce computing time.

fitted_spectrum(*energies=None*, *pixel=None*, *index=0*, *representation='fit_parameters'*)

Take the previously calculated fitting parameters from *fit_spectra* and predict the XANES spectrum.

Parameters

- **energies** (*np.ndarray, optional*) – The input values to give to the predictor function. If omitted, a np.linspace will be created covering the range of energies.
- **pixel** (*2-tuple, optional.*) – The specific pixel to use for getting fit parameters. If omitted, the average for each parameter calculated.
- **index** (*int, optional*) – The timeindex to use for retrieving fit parameters.
- **representation** (*str, optional*) – Which dataset name to use for accessing the fit parameters. The default value ("fit_parameters") is the one used by `fit_spectra` to save the results.

Returns **fit** – The predicted spectrum based on the previously calculated fit parameters.

Return type pd.Series

fork_data_group(*dest*, *src=None*)
Turn on different active data for this frameset's store object. Similar to *switch_data_group* except that this method deletes the existing group and copies symlinks from the current one.

Parameters

- **dest** (*str*) – Name for the data group.
- **src** (*str*) – String with the name of the data group to copy from. If None (default), the current data group will be used.

frame_shape(*representation='intensities'*)
Return the shape of the individual energy frames.

frames
Return the frames for the given time index.

If *representation* is really mapping data, then the source frames will be returned.

Parameters

- **timeidx** (*int*) – Index for the first dimension of the combined data array.
- **representation** (*str*) – The group name for these data. Eg "absorbances", "white-line_map", "intensities"

Returns **frames** – A 3-dimensional array with the form (energy, row, col).

Return type np.ndarray

has_representation(*representation*)

hdf_path(*representation=None*)
Return the hdf path for the active group.

Parameters **representation** (*str, optional*) – Name of third-level group to use. If omitted, the path to the parent group will be given.

Returns **path** – The path to the current group in the HDF5 file. Returns an empty string if the representation does not exists.

Return type str

label_particles(*min_distance=20*)
Use watershed segmentation to identify particles.

Parameters **min_distance** (–) – grouping areas into particles. Lower numbers means more particles, but might split large particles into two.

map_data
Return map data for the given time index and representation.

If *representation* is really mapping data, then the result will have more dimensions than expected.

Parameters

- **timeidx** (*int*) – Index for the first dimension of the combined data array. If the underlying map data has only 2 dimensions, this parameter is ignored.
- **representation** (*str*) – The group name for these data. Eg "absorbances", "white-line_map", "intensities"

Returns **map_data** – A 2-dimensional array with the form (row, col).

Return type np.ndarray

mean_frame(*representation='absorbances'*)
Return the mean value with the same shape as an individual frame.

normalize(*plot_fit=False*, *new_name='normalized'*)
Correct for background material not absorbing at this edge. Uses method described in DOI 10.1038/ncomms7883: fit line against material that fails edge_jump_filter and use this line to correct entire frame.

Parameters

- **plot_fit** (*–*) –
- **line.** (*best-fit*) –

num_energies

num_timesteps

particle_regions(*intensity_image=None*, *labels=None*)
Return a list of regions (1 for each particle) sorted by area. (largest first). This requires that the *label_particles* method be called first.

Parameters

- **intensity_image** (*np.ndarray, optional*) – 2D array passed on to the skimage *regionprops* function to determine what shows up in the image for each particle.
- **labels** (*np.ndarray, optional*) – Array of the same shape as the map, with the particles segmented. If None (default), the *particle_labels* attribute of the TXM store will be used.

particle_series(*map_name='whiteline_max'*)
Generate an array of values from map_name averaged across each particle.

Returns: A 2D array where the first dimension is particles and the second is the first dimension of the map dataset (usually time).

pixel_unit()
Return the unit of measure for the size of a pixel.

plot_frame(*idx*, *ax=None*, *cmap='gray'*, **args*, ***kwargs*)
Plot the frame with given index as an image.

plot_histogram(*plotter=None*, *timeidx=None*, *ax=None*, *vmin=None*, *vmax=None*, *goodness_filter=False*, *representation='whiteline_fit'*, *component='real'*, *active_pixel=None*, *bins='energies'*, **args*, ***kwargs*)
Use a default frameset plotter to draw a map of the chemical data.

plot_map(*ax=None*, *map_name='whiteline_fit'*, *timeidx=0*, *vmin=None*, *vmax=None*, *median_size=0*)
Prepare data and plot a map of whiteline positions.

Parameters **median_size** (*int*) – Kernel size for the median rank filter.

plot_map_pixel_spectra(*pixels*, *map_ax=None*, *spectra_ax=None*, *map_name='whiteline_map'*, *timeidx=0*, *step_size=0*, **args*, ***kwargs*)
Plot the frameset's map and highlight some pixels on it then plot those pixel's spectra on another set of axes.

Parameters

- **pixels** (*iterable*) – An iterable of 2-tuples indicating which (row, column) pixels to highlight.

- **map_ax** (*optional*) – A matplotlib axes object to put the map onto. If None, a new 2-wide subplot will be created for both map_ax and spectra_ax.
- **spectra_ax** (*optional*) – A matplotlib axes to be used for plotting spectra. Will only be used if `map_ax` is not None.
- **map_name** (*str, optional*) – Name of the map to use for plotting. It will be passed to the TXM store object and retrieved from the hdf5 file. If falsy, no map will be plotted.
- **timeidx** (*int, optional*) – Index of which timestep to use (default: 0).
- **kwargs** (*args,*) – Passed to plots.plot_pixel_spectra()

plot_mean_image(*ax=None*, *component='modulus'*, *cmap='gray'*, **args*, ***kwargs*)

plot_signal_map(*ax=None*, *signals_idx=None*, *interpolation=None*)

Plot the map of signal strength for signals extracted from self.calculate_signals().

Parameters

- **ax** (–) – axes object is created.
- **signals_idx** (–) –
- **as a numpy array index. Special value None(default)** (*passed*) –
- **first three signals will be plotted.** (*means*) –
- **interpolation** (–) – How to smooth the image when plotting.

plot_signals(*cmap='viridis'*)

Plot the signals from the previously extracted data. Requires that self.store().signals and self.store().signal_weights be set.

plot_xanes_spectrum(*ax=None*, *pixel=None*, *norm_range=None*, *normalize=False*, *representation='absorbances'*, *show_fit=False*, *edge_jump_filter=False*, *linestyle=':'*, **args*, ***kwargs*)

Calculate and plot the xanes spectrum for this field-of-view.

Parameters

- **– matplotlib axes object on which to draw** (*ax*) –
- **– Coordinates of a specific pixel on the image to plot.** (*pixel*) –
- **– If truthy, will set the pre-edge at zero and the** (`normalize`) – post-edge at 1.
- **– If truthy, will use the edge object to fit the data** (*show_fit*) – and plot the resulting fit line.
- **– If truthy, will only include those values** (*edge_jump_filter*) – that show a strong absorbtion jump across this edge.

qt_viewer()

spectra(*edge_filter=False*)

Return a two-dimensional array of spectra for all the pixels in shape of (pixel, energy).

Parameters (boolor str) (*edge_jump_filter*) – truthy, only pixels that pass the edge jump filter are used to calculate the spectrum. If "inverse" is given, then the edge jump filter is logically not-ted and calculated with a more conservative threshold.

spectrum(*pixel=None*, *edge_jump_filter=False*, *representation='absorbances'*, *index=0*)
Collapse the frameset down to an energy spectrum.

Any dimensions (besides the energy dimension) that remain after applying the arguments below, will be averaged to give the final intensity at each energy.

Returns **spectrum** – A pandas Series with the spectrum.

Return type pd.Series

Parameters

- **pixel** (*tuple, optional*) – A 2-tuple that causes the returned series to represent the spectrum for only 1 pixel in the frameset. If None, a larger part of the frame will be used, depending on the other arguments.
- **edge_jump_filter** (*bool or str, optional*) – If truthy, only pixels that pass the edge jump filter are used to calculate the spectrum. If "inverse" is given, then the edge jump filter is logically not-ted and calculated with a more conservative threshold.
- **representation** (*str, optional*) – What kind of data to use for creating the spectrum. This will be passed to TXMStore.get_map()
- **index** (*int, optional*) – Which step in the frameset to use. When used to index store().absorbances, this should return a 3D array like (energy, rows, columns).

stage_transformations(*translations=None*, *rotations=None*, *center=(0, 0)*, *scales=None*)
Allows for deferred transformation of the frame data.

Since each transformation introduces interpolation error, the best results occur when the translations are saved up and then applied all in one shot. Takes a combination of arrays of translations (x, y), rotations and/or scales and saves them for later application. This method should be used in conjunction apply_transformations().

All three arguments should have shapes that are compatible with the frame data, though this is not strictly enforced for now. Rotation will necessarily have one less degree of freedom than translation/scale values.

Example Shapes:

Frames	Translations	Rotations	Scales
(10, 48, 1024, 1024)	(10, 48, 2)	(10, 48, 1)	(10, 48, 2)
(10, 48, 1024, 1024, 1024)	(10, 48, 3)	(10, 48, 2)	(10, 48, 3)

Parameters

- **translations** (*np.ndarray*) – How much to move each axis (x, y[, z]).
- **rotations** (*np.ndarray*) – How much to rotate around the origin (0, 0) pixel.
- **center** (*2-tuple*) – Where to set the origin of rotation. Default is the first pixel (0, 0).
- **scales** (*np.ndarray*) – How much to scale the image by in each dimension (x, y[, z]).

starttime(*timeidx=None*)
Determine the earliest timestamp amongst all of the frames.

Parameters **timeidx** (*int, optional*) – Which timestep to use for finding the start time. If omitted or None (default), all timesteps will be checked.

Returns **start_time** – Naive datetime representing the earliest known frame for this timeidx. Timezone will always be UTC no matter where the data were collected.

Return type np.datetime64

store(*mode='r'*)
Get a TXM Store object that saves and retrieves data from the HDF5 file. The mode argument is passed to h5py as is. This method should be used as a context manager, especially if mode is something writeable:

```
# The 'r+' creates the file or appends if one exists
with self.store(mode='r+') as store:
    # Do stuff with the store...
    img = store.absorbances[0,0]
```

subtract_surroundings()
Use the edge mask to separate "surroundings" from "sample", then subtract the average surrounding absorbance from each frame. This effectively removes effects where the entire frame is brighter from one energy to the next.

6.1.13 xanespy.xanes_math module

Module containing all the computationally demanding functions. This allows for easy optimization of parallelizable algorithms. Most functions will operate on large arrays of data.

class xanespy.xanes_math.**KEdgeParams**(*scale*, *voffset*, *E0*, *sigw*, *bg_slope*, *ga*, *gb*, *gc*)
Bases: tuple

E0
Alias for field number 2

bg_slope
Alias for field number 4

ga
Alias for field number 5

gb
Alias for field number 6

gc
Alias for field number 7

scale
Alias for field number 0

sigw
Alias for field number 3

voffset
Alias for field number 1

xanespy.xanes_math.**apply_internal_reference**(*intensities*, *out=None*)
Apply a reference correction to complex data to convert intensities into refractive index. I_0 is determined by separating the pixels into background and foreground using Otsu's method.

Arrays *intensities* and *out* must all have the same shape where the last two dimensions are image rows and column.

xanespy.xanes_math.**apply_mosaic_reference**(*intensity*, *reference*)
Use a single reference frame to calculate the reference for a mosaic of intensity frames.

Returns - out – Same shape as intensity (I) but with optical depth: ln(I/I0) where I0 is the reference frame.

Return type ndarray

Parameters

- **intensity** (–) – Intensity data image of the sample. It's shape must be a whole multiple of the shape of `reference`.
- **reference** (–) – A reference image with no sample that will be applied.

`xanespy.xanes_math.`**apply_references**(*intensities*, *references*, *out=None*)

Apply a reference correction to convert intensity values to optical depth.

The formula $-ln\frac{intensities}{references}$ is used to calculate the new values. Arrays `intensities`, `references` and `out` must all have the same shape where the last two dimensions are image rows and columns.

Parameters

- **intensities** (*np.ndarray*) – Sample input signal data.
- **references** (*np.ndarray*) – Background input signal data. Must be the same shape as intensities.
- **out** (*np.ndarray, optional*) – Array to receive the results.

`xanespy.xanes_math.`**direct_whitelines**(*spectra*, *energies*, *edge*)

Takes an array of X-ray absorbance spectra and calculates the positions of maximum intensities over the near-edge region.

Parameters

- **spectra** (*np.array*) – 2D numpy array of absorbance spectra where the last dimension is energy.
- **energies** (*np.array*) – Array of X-ray energies in electron-volts. Must be broadcastable to the shape of spectra.
- **edge** – An XAS Edge object that describes the absorbance edge in question.

Returns **out** – Array with the whiteline position of each spectrum.

Return type np.ndarray

`xanespy.xanes_math.`**extract_signals_nmf**(*spectra*, *n_components*, *nmf_kwargs=None*, *mask=None*)

Extract the signal components present in the given spectra using non-negative matrix factorization. Input data can be negative, but it will be shifted up, processed, then shifted down again.

Parameters

- **spectra** (–) –
- **n_components** (–) –
- **nmf_kwargs** (–) – the constructor of the estimator.

Returns

Return type 2-tuple of arrays (*components*, weights)

`xanespy.xanes_math.`**fit_kedge**(*spectra*, *energies*, *p0*)

Use least squares to fit a set of curves to the data. Currently this is a line for the baseline absorbance decreasing at higher energies, plus a sigmoid for the edge and a gaussian for the whiteline.

Returns an array with a similar shape to spectra but the last axis is replaced with fitting parameters, describe by the named tupled *KParams* defined in this module.

Parameters

- **spectra** (–) – index is energy. This can be a multi-dimensional array, which allows calculation of image frames, etc. The last axis should be X-ray energy.
- **energies** (–) –
- **p0** (–) – described by kedge_params.
- **out** (–) –
- **be created.** (*will*) –

xanespy.xanes_math.**fit_kedge_mpi**(*spectra*, *energies*, *p0*)

Use least squares to fit a set of curves to the data. Very similar to `fit_k_edge()` except using message passing interface (MPI) for parallel processing.

Returns an array with a similar shape to spectra but the last axis is replaced with fitting parameters, described by the named tupled `KParams` defined in this module.

Parameters

- **spectra** (*np.ndarray*) – An array containing absorbance data. Assumes that the index is energy. This can be a multi-dimensional array, which allows calculation of image frames, etc. The last axis should be X-ray energy.
- **energies** (*np.ndarray*) – Array of X-ray energies. Must have same shape as *spectra*.
- **p0** (*tuple*) – A tuple with the initial guess. The correct order is described by kedge_params.
- **out** (*np.ndarray*) – To hold the results. If omitted, a new array will be created.

xanespy.xanes_math.**guess_kedge**(*spectrum*, *energies*, *edge*)

Guess initial starting parameters for a k-edge curve. This will give a rough estimate, appropriate for giving to the fit_kedge function as the starting parameters, p0.

Parameters

- **spectrum** (–) – K-edge spectrum. Only 1-dimensional data are currently accepted.
- **energies** (–) – the points in *spectrum*. Must have the same shape as *spectrum*.
- **edge** (–) – actual edge energy itself.
- **Returns** (*A named tuple with the estimated parameters (see*) – .KEdgeParams for definition)

xanespy.xanes_math.**iter_indices**(*data*, *leftover_dims=1*, *desc=None*)

Accept an array of frames, indices, etc. and generate slices for each frame. Assumes the last two dimensions of *data* are rows and columns. All other dimensions will be iterated over.

- leftover_dims : Integer describing which dimensions should not be iterated over. Eg. if data is 3D array and leftover_dims == 1, only first two dimenions will be iterated.
- desc : String to put in the progress bar.

xanespy.xanes_math.**k_edge_jump**(*frames: numpy.ndarray*, *energies: numpy.ndarray*, *edge*)

Determine what the difference is between the post_edge and the pre_edge.

xanespy.xanes_math.**k_edge_mask**(*frames: numpy.ndarray*, *energies: numpy.ndarray*, *edge*, *sensitivity: float = 1*, *min_size=0*)

Calculate a mask for what is likely active material at this edge. This is done by comparing the edge-jump to the standard deviation. Foreground material will be identified when the edge-jump accounts for most of the standard deviation.

Parameters

- **frames** (*numpy.ndarray*) – Array with images at different energies.
- **energies** (*numpy.ndarray*) – X-ray energies corresponding to images in `frames`. Must have the same shape along the first dimenion as `frames`.
- **edge** (`KEdge`) – A *xanespy.edges.KEdge* object that contains a description of the elemental edge being studied.
- **sensitivity** (*float, optional*) – A multiplier for the otsu value to determine the actual threshold.
- **min_size** (*int, optional*) – Objects below this size (in pixels) will be removed. Passing zero (default) will result in no effect.

Returns **mask** – A boolean mask with the same shape as the last two dimensions of `frames` where True pixels are likely to be background material.

Return type numpy.ndarray

xanespy.xanes_math.**l_edge_mask**(*frames: numpy.ndarray*, *energies: numpy.ndarray*, *edge*, *sensitivity: float = 1*, *frame_dims=2*, *min_size=0*)

Calculate a mask for what is likely active material at this edge. This is done by comparing each spectrum to the overall spectrum using the dot product. A normalization is first applied to mitigate differences in total intensity.

Parameters

- **frames** (–) –
- **energies** (–) – *frames*. Must have the same shape along the first dimenion as *frames*.
- **edge** (–) – elemental edge being studied.
- **sensitivity** (–) – the actual threshold.
- **frame_dims** (–) – means each frame is a two-dimensional image.
- **min_size** (–) – removed. Passing zero (default) will result in no effect.

Returns

- *- A boolean mask with the same shape as the last two dimensions of*
- *frames* where True pixels are likely to be background material.

xanespy.xanes_math.**particle_labels**(*frames: numpy.ndarray*, *energies: numpy.ndarray*, *edge*, *min_distance=20*)

Prepare a map by segmenting the images into particles.

Parameters **frames** (*numpy.ndarray*) – An array of images, each one at a different energy. These will be merged and used for segmentation.

xanespy.xanes_math.**predict_edge**(*energies*, **params*)

Defines the curve function that gets fit to the data for an absorbance K-edge.

The predicted curve is a combination of a straight line (for background), an arctan function (for the edge), and a gaussian peak (for the whiteline).

Parameters

- **energies** (*np.ndarray*) – Array with energy values to be predicted.
- ***params** (*tuple(int)*) – The curve parameters that should be used for the prediction. Their order is described by kedge_params variable.

Returns **curve** – The predicted absorbance values based on the input parameters. Shape will match *energies*.

Return type np.ndarray

xanespy.xanes_math.**register_correlations**(*frames*, *reference*, *upsample_factor=10*, *desc='Registering'*)

Calculate the relative translation between the reference image and a series of frames.

This uses phase correlation through scikit-image's *register_translation* function.

Parameters

- **frames** (*np.ndarray*) – Array where the last two dimensions are (column, row) of images to be registered.
- **reference** (*np.ndarray*) – Image frame against which to align the entries in *frames*.
- **upsample_factor** (*int, optional*) – Factor controls subpixel registration via scikit-image.
- **desc** (*str, optional*) – Description for putting in the progress bar.

Returns **translations** – Array with same dimensions as 0-th axis of *frames* containing (x, y) translations for each frame.

Return type np.ndarray

xanespy.xanes_math.**register_template**(*frames*, *reference*, *template*, *desc='Registering'*)

Calculate the relative translation between the reference image and a series of frames.

This uses template cross correlation through scikit-image's *match_template* function.

The *register_correlations* algorithm is simpler to use in most cases but sometimes results in unreasonable results; in those cases, this method can be more reliable to achieve a first approximation.

Parameters

- **frames** (*np.ndarray*) – Array where the last two dimensions are (column, row) of images to be registered.
- **reference** (*np.ndarray*) – Image frame against which to align the entries in *frames*.
- **template** (*np.ndarray*) – A 2D array (smaller than frames and reference) that will be identified in each frame and used for alignment.
- **desc** (*str, optional*) – Description for putting in the progress bar.

Returns **translations** – Array with same dimensions as 0-th axis of *frames* containing (x, y) translations for each frame.

Return type np.ndarray

xanespy.xanes_math.**transform_images**(*data*, *transformations*, *out=None*, *mode='median'*)

Takes image data and applies the given translation matrices.

It is assumed that the first dimension of *data* is the same as the length of *transformations*. The transformation matrices can be generated from translation, rotation and scale parameters via the `xanespy.xanes_math.transformation_matrics()` function. Data will be written to *out* if given, otherwise returned as a new array.

Parameters

- **data** (*np.ndarray*) – Numeric array with frames to transform. Last two dimensions are assumed to be (row, columns).
- **transformations** (*np.ndarray*) – A numeric array shaped compatibally with *data*. The last two dimensions are assumed to be (3, 3) and each (3, 3) encodes a transformation matrix for the corresponding frame in *data*.

- **out** (*np.ndarray, optional*) – A numeric array with same shape as *data* that will hold the transformed data.
- **mode** (*str, optional*) – Describes how to deal with edges. See scikit-image documentation for options. Special value "median" (default), takes the median pixel intensity of that frame and uses it as the constant value.

Returns **out** – A new array with similar dimensions to *data* but with transformations applied and converted to float datatype.

Return type np.ndarray

xanespy.xanes_math.**transformation_matrices**(*translations=None*, *rotations=None*, *scales=None*, *center=(0, 0)*)

Takes array of operations and calculates (3, 3) transformation matrices.

This function operates by calculating an AffineTransform similar to that described in the scikit-image package.

All three arguments (*translations*, *rotations*, and *scales*, should have shapes that are compatible with the frame data, though this is not strictly enforced for now. Rotation will necessarily have one less degree of freedom than translation/scale values.

Example Shapes:

Frames	Translations	Rotations	Scales
(10, 48, 1024, 1024)	(10, 48, 2)	(10, 48, 1)	(10, 48, 2)
(10, 48, 1024, 1024, 1024)	(10, 48, 3)	(10, 48, 2)	(10, 48, 3)

Parameters

- **translations** (*np.ndarray, optional*) – How much to move each axis (x, y[, z]).
- **rotations** (*np.ndarray, optional*) – How much to rotate around the origin (0, 0) pixel.
- **center** (*np.ndarray, optional*) – Where to set the origin of rotation. Default is the first pixel (0, 0).
- **scales** (*np.ndarray, optional*) – How much to scale the image by in each dimension (x, y[, z]).

Returns **new_transforms** – Resulting transformation matrices. Will have the same shape as the input arrays but with the last dimension replaced by (3, 3).

Return type np.ndarray

6.1.14 xanespy.xradia module

Tools for importing X-ray microscopy frames in formats produced by Xradia instruments.

class xanespy.xradia.**TXRMFile**(*filename*, *flavor: str*)

Bases: *xanespy.xradia.XRMFile*

Similar to *XRMFile* but contains multiple images in a data-set.

energies()

image_stack()

class xanespy.xradia.**XRMFile**(*filename*, *flavor: str*)
Bases: object

Single X-ray micrscopy frame created using XRadia XRM format.

Formats from different beamlines have subtly different storage patterns. The `flavor` argument controls this parameter. The following metadata are affected by this coice:

- Energy: SSRL 6-2c does not store the X-ray beam energy in the file so it must be extracted from the filename.
- starttime, endtime : The XRMFile does not store timezone info, so we have to guess based on beamline.
- pixel_size : The APS microscope automatically corrects magnification when changing energy, the SSRL microscope does not.

Parameters

- **filename** (*str*) – The path to the .xrm file
- **flavor** (*str*) – The variety of data represented in the xrm file. Valid choices are ['ssrl', 'aps', 'aps-old1']. These choices should line up with whatever is generated using the scripts in beamlines moudles.

aps_old1_regex = re.compile('(\d{8})_([a-zA-Z0-9_]+)_([a-zA-Z0-9]+)_(\d{4}).xrm')

binning()
Binning mode of the image.

Binning reduces pixel density but improves signal/noise ratio by combining adjacent pixels. For example, a 2048 x 2048 CCD with binning 4 would produce a 512 x 512 image.

close()
Close original XRM (ole) file on disk.

endtime()
Retrieve a datetime object representing when this frame was finished collecting. Duration is decided by exposure time of the frame and the start time.

energy()
Beam energy in electronvoltes.

image_data(*idx=0*)
TXM Image frame.

image_dtype()

image_shape()

is_valid()
Check that the XRM file has valid data.

num_images()

ole_value(*stream*, *fmt=None*, *as_array=False*)
Get arbitrary data from the ole file and convert from bytes.

print_ole()

sample_position()

starttime()
Return the earliest timestamp for the collected frames.

starttimes()
Retrieve all datetime objects representing when these frames were collected. Timezone is inferred from flavor (eg. ssrl -> california time).

um_per_pixel()
Describe the size of a pixel in microns. If this is an SSRL frame, the pixel size is dependent on energy. For APS frames, the pixel size is uniform and assumes a 40μm field-of-view.

6.1.15 Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

X

INDEX

D

E

F

G

H

I

K

L

M

N

T

U

V

W

X

Y

Z