
Xanespy Documentation

Release 0.1.7

Mark Wolfman

Mar 11, 2020

Contents:

1	Introduction	3
1.1	Installation	3
1.2	X-Ray Absorbance Basics	3
1.3	Example Workflow	4
2	Importing Data into Xanespy	5
2.1	APS Beamline 32-ID-C	5
2.2	SSRL Beamline 6-2c - Directory of XRM Files	6
2.3	Ptychography from 5.3.2.1 (ALS)	6
2.4	Xradia Image Files (.xrm and .txrm)	7
2.5	APS Beamline 8-BM-B - Energy Stack (TXRM)	8
2.6	APS Beamline 8-BM-B - Directory of XRM Files	8
3	Analyzing the Data	11
3.1	Defining an Absorbance Edge	11
3.2	Forking the Data	12
3.3	Frame Alignment	12
3.4	Median Filtering	13
3.5	Subtracting Surroundings	14
3.6	Calculating Maps	14
3.7	Fitting Spectra	14
4	Fitting X-ray Absorbance Spectra	15
4.1	Linear Combination Fitting	15
4.2	Fitting K- and L- Spectral Edges	16
4.3	Rolling Your Own Fit Function	16
5	Visualization of Results	19
5.1	Plotting Maps	19
5.2	Interactive (Qt) Viewer	19
6	Accessing the Data Directly	21
7	xanespy	23
7.1	xanespy package	23
8	Indices and tables	73

Python Module Index	75
Index	77

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

Warning: This documentation and the code are under active development. We make every effort to ensure the code is usable, but make no guarantees.

CHAPTER 1

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 captured 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
```

1.2 X-Ray Absorbance Basics

Coming soon...

1.3 Example Workflow

A typical procedure 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, statically 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()
```

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 32-ID-C

The `energy_scan` script at 32-ID-C saves source data as an HDF file. Xanespy preserves the original (“source”) file and saves imported and processed data in a second (“desination”) HDF file to be used in later analysis. The source file can be easily imported:

```
import_aps32idc_xanes_file(filename='source_xanes_example_scan_001.h5',
                           hdf_filename='xanespy_destination.h5',
                           hdf_groupname='example_scan_001',
                           downsample=1, square=True)
```

The camera at the beamline captures micrographs of 2048x2448. In most cases this is over-kill since the focusing power of the 60 nm zone-plate is not strong enough to take advantage of this. The extra pixel density can be converted into an improved contrast-to-noise ratio by the `downsample` parameter. This parameter controls how many pixels (to a power of two) are combined. In the previous example, `downsample=1` combines to $2^1 = 2 \times 2$ blocks and results in a (1024, 1224) image. `downsample=2` combines $2^2 = 4 \times 4$ blocks to produce a (512, 612) image.

There are sometimes corner artifacts in the camera, independent of sample position: this can cause poor alignment of the imported data. An easy solution is to import the data with `square=True`. This will crop the imported images to be square, discarding the corner artifacts.

Operando experiments generate one HDF file per energy scan. Using the above approach would continually overwrite the imported data, leaving only the last scan in the import data store. To overcome this, the `timestep` and `total_timesteps` parameters should be used with `import_aps32idc_xanes_file()`, or the related `import_aps32idc_xanes_files()` function.

Option 1: Loop through the files explicitly:

```
# Populate a list of filenames, eg. with listdir() from the os module
filenames = []
# Loop through the files and import then one at a time
for idx, fname in enumerate(filenames):
    import_aps32idc_xanes_file(fname, hdf_filename='my_XANES_data.h5',
                              hdf_groupname='operando_001',
                              timestep=idx, total_timesteps=len(filenames))
```

Option 2: Convenience function.

```
# Populate a list of filenames, eg. with listdir() from the os module
filenames = []
# The function basically does what is shown in option 1
import_aps32idc_xanes_files(filenames, hdf_filename='my_XANES_data.h5',
                            hdf_groupname='operando_002')
```

Any additional parameters given to the convenience function `import_aps32idc_xanes_files()` will be passed directly to the inner `import_aps32idc_xanes_file()` function. If using option 1, it is important that parameters controlling the data shape are consistent across calls: `total_timesteps`, `append`, `downsample`, `square` and `exclude`.

2.2 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("operando_exp1/",
                        hdf_filename="operando_experiments.h5")
```

2.3 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.

2.4 Xradia Image Files (.xrm and .txrm)

Xradia microscopes use the Microsoft OLE container format, which is not easily read¹. 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()
```

(continues on next page)

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.

(continued from previous page)

```
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.

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

Note: The X-ray microscope that was temporarily at beamline 8-BM has been returned to NSLS-II. These functions are retained for compatibility with previously collected data.

The Xradia microscope can save an entire stack in one `.txrm` file. This file can be imported using the `import_aps8bm_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_aps8bm_xanes_file('expl-sample-stack.txrm',
                           ref_filename='expl-reference_stack.txrm',
                           hdf_filename='txm-data.h5',
                           groupname='experiment1')
```

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](#).

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

Note: The X-ray microscope that was temporarily at beamline 8-BM has been returned to NSLS-II. These functions are retained for compatibility with previously collected data.

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_aps8bm_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 sector8_xanes_script()
```

(continues on next page)

(continued from previous page)

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


CHAPTER 3

Analyzing the Data

After importing the data, you will likely need to process and analyze the frame set. This is done through the `xanespy.xanes_frameset.XanesFrameset` class. Most **processing and analysis steps are provided as methods on this class**, so the first step is to create a frameset object.

```
import xanespy as xp

# Use the same HDF file and groupname as when importing
fs = xp.XanesFrameset(filename='my_analysis.h5',
                      groupname='experiment1',
                      edge=None)
```

3.1 Defining an Absorbance Edge

Some steps in the analysis process require knowledge of the X-ray absorbance edge to function properly. To use these features, you'll need to define and absorbance edge for the element in question. You can also use one of the pre-defined edges, either directly or by the shortcuts `xanespy.edges.k_edges` or `xanespy.edges.l_edges`.

```
import xanespy as xp

# Using a pre-made edge by name
fs = xp.XanesFrameset(edge=xp.k_edges['Ni'], ...)

# Using a pre-made edge by reference
fs = xp.XanesFrameset(edge=xp.edges.NCANickelKEdge(), ...)

# Defining your own edge
class CuKEdge(KEdge):
    name = 'Cu'
    E_0 = 8978.9
    shell = "K"
    pre_edge = (8940, 8970)
```

(continues on next page)

(continued from previous page)

```
post_edge = (9010, 9200)
edge_range = (8970, 9010)
map_range = (8970, 9010)
fs = xp.XanesFrameset(edge=CuKEdge(), ...)
```

3.2 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, the `xanespy.xanes_frameset.XanesFrameset` class 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.

3.3 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 `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. 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)
```

3.4 Median Filtering

There are three options for applying a median filter, with each one having a different purpose. The larger the size of the kernel given, the longer it will take to apply the filter.

3.4.1 Filter When Importing

Area detectors often have some number of **bad pixels**, either hot pixels or dead pixels. Applying a mild median filter when using one of the importers in `xanespy.importers`, the raw data can fix most of these problems. Some beamline importers apply this by default. The **3D filter** can also include the energy dimension, but this is not recommended since the frames haven't been aligned yet:

```
import xanespy as xp

xp.import_aps32idc_file(median_filter_size=(1, 5, 5))
```

3.4.2 Filter When Aligning

When aligning frames with `align_frames()`, it may be helpful to apply an **aggressive median filter to blur each image** before registration so that noise and fine details have less impact. This **2D filter** is only applied to the images in memory, so does not apply to the final result.

```
import xanespy as xp

fs = xp.XanesFrameset(...)
fs.align_frames(median_filter_size=(5, 5))
```

3.4.3 Filter After Aligning

Depending on the scientific question being addressed, a **final median filter after aligning** maybe desirable. This **4D filter**, applied with `apply_median_filter()`, provides a trade-off between temporal, spatial and energy resolutions: The larger the kernel along one dimension, the less resolution you'll be able to see but the higher the signal-to-noise in the other dimensions.

```
import xanespy as xp

fs = xp.XanesFrameset(...)
fs.align_frames(...)
kernel = (3, 3, 5, 5) # (time, energy, row, col)

fs.median_filter(kernel)
```

3.5 Subtracting Surroundings

Sometimes there are differences in the absorbance of the whole frame, including background material. This can be removed from each frame using `subtract_surroundings()`, giving a better spectrum:

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

Fig. 2: The effect of the `subtract_surroundings()` method.

3.6 Calculating Maps

Several basic maps can be create with the `calculate_maps()` method. These maps will be saved in the HDF5 file alongside the frames.

```
import matplotlib.pyplot as plt
from xanespy import xp

fs = xp.XanesFrameset()
fs.calculate_maps()

# Visualize one of the newly created
fs.plot_map(map_name='optical_depths_mean')
```

More fine-grained mapping is planned and will be available soon.

3.7 Fitting Spectra

When numerical methods are insufficient, it may be necessary to fit the pixel spectra with a model function and extract parameters from the model. A comprehensive guide can be found on the page [Fitting X-ray Absorbance Spectra](#).

Fitting X-ray Absorbance Spectra

When numerical methods are insufficient, it may be necessary to fit the pixel spectra with a model function and extract parameters from the model. The core *XanesFrameset* class has methods for common fitting-related use-cases, such as using curves to approximate L₃ and K edges, and linear combination fitting of standard spectra. If the pre-rolled options are not enough, arbitrary callables can be created and fit against the data. The following is an example for fitting a single L-edge spectrum:

```
Es = np.linspace(845, 865, num=1000)
obs = ... # Load your data
l3 = xp.fitting.L3Curve(Es, num_peaks=2)

# Create initial guess, matching ``l3.param_names``
p0 = (1.1, 853, 0.6, 1, 855, 0.6, 0.15, 854, 10, 3)

# Now do the fitting
result = xp.fitting.fit_spectra(observations=obs, func=l3, p0=p0)
params, residuals = result

# Plot the resulting fit and original data
predicted = l3(*params)
plt.plot(Es, obs)
plt.plot(Es, predicted, linestyle=':')
```

Todo: Come up with a better illustration for fitting.

4.1 Linear Combination Fitting

Often times the observed spectrum is a linear combination of spectral sources from known standards. This works best when the standards is stable and have been isolated and measured on the same instrument as the observed data.

In order to fit linear combinations of source, use the `fit_linear_combinations()` method:

```
# Load your previously imported data
fs = xp.XanesFrameset(...)

# Prepare the sources for fitting
source1 = ...
source2 = ...

# The sources must have the same number of points as the data
assert len(source1) == fs.num_energies
assert len(source2) == fs.num_energies

# Now execute the fitting
results = fs.fit_linear_combinations(sources=[source1, source2])
fits, residuals = results
```

Todo: Create a figure to illustrate LC fitting.

This method will create three new HDF5 datasets:

- `linear_combination_parameters` (maps)
- `linear_combination_residuals` (maps)
- `linear_combination_sources` (arrays)

The naming prefix can be controlled by passing the name parameter to `fit_linear_combinations()` method. If more control is needed, the `xanespy.fitting.LinearCombination` class can be subclassed and given to the `fit_spectra()` method as described *below*.

4.2 Fitting K- and L- Spectral Edges

A common use case is to fit the spectra with either a K-edge or L-edge shape. This can be done easily with the `fit_l_edge()` or `fit_k_edge()` methods.

4.3 Rolling Your Own Fit Function

If none of the options suit your needs, you can create a callable that produces the curve you wish to fit given a number of parameters, then pass this to the `fit_spectra()` method. In the simplest case this can be a simple function:

```
import numpy as np
import xanespy as xp

# Define the function we wish to fit against
def sin_curve(scale, frequency, phase):
    theta = np.linspace(0, 2*np.pi, num=100)
    out = scale * np.sin(frequency * theta(phase))
    return out

fs = xp.XanesFrameset(...)
# Come up with an initial guess
pnames = ('scale', 'frequency', 'phase')
```

(continues on next page)

(continued from previous page)

```
p0 = (0, 1, 0)
fs.fit_spectra(func=sin_curve, p0=p0, pnames=pnames, name='sin_curve')
```

In many cases, static information (such as the list of energies) is needed to construct the curve. This can be given to a class's constructor and the algorithm itself placed in the `__call__` method. This is illustrated below by fitting a variable number of sine waves, making a sort of horribly inefficient fourier transform. Since the number of sine waves is not known at import-time, the use of star-arguments makes the result more dynamic. Adding the `param_names` saves us the trouble of passing it in every time.

```
# Define a new callable for passing to the fitting function
class SineCurves():
    def __init__(self, theta, num_sines=1):
        self.theta = theta
        self.num_sines = num_sines

    def __call__(self, *params):
        out = np.zeros_like(self.theta)
        # Iterate on the parameters in groups of 3
        for i in xrange(0, len(params), 3):
            scale, freq, phase = params[i:i+3]
            # Add another sin wave to the total curve
            out += scale * np.sin((self.theta-phase) * frequency)
        return out

    @property
    def param_names(self):
        # Build a list of 2 params for each sine wave
        names = []
        for num in range(self.num_sines):
            names.append('scale%d' % num)
            names.append('frequency%d' % num)
            names.append('phase%d' % num)
        return names

# Create the actual callable object
theta = np.linspace(0, 2*pi, num=100)
sines = SineCurves(theta=theta, num_curves=3)
# Load the data
fs = xp.XanesFrameset(...)
p0 = []
for i in range(3):
    p0.append(1, 2*i+1, 0)
fs.fit_spectra(func=sines, p0=p0, name='sine_curve_fit')
```

Visualization of Results

5.1 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.

5.2 Interactive (Qt) Viewer

Xanespy includes a graphical user interface that allows for interactive visualization of X-ray frames and maps. The viewer is launched from the command line and takes the path to a processed HDF file as its input. For some extra functionality, you can give the name of a metal K- or L- edge using the `-k` or `-l` argument respectively. See `xanespy-viewer --help` for a list of K and L edges available.

```
$ xanespy-viewer results/beamtime-analysis.h5 -k Ni
```

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 launched 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`.

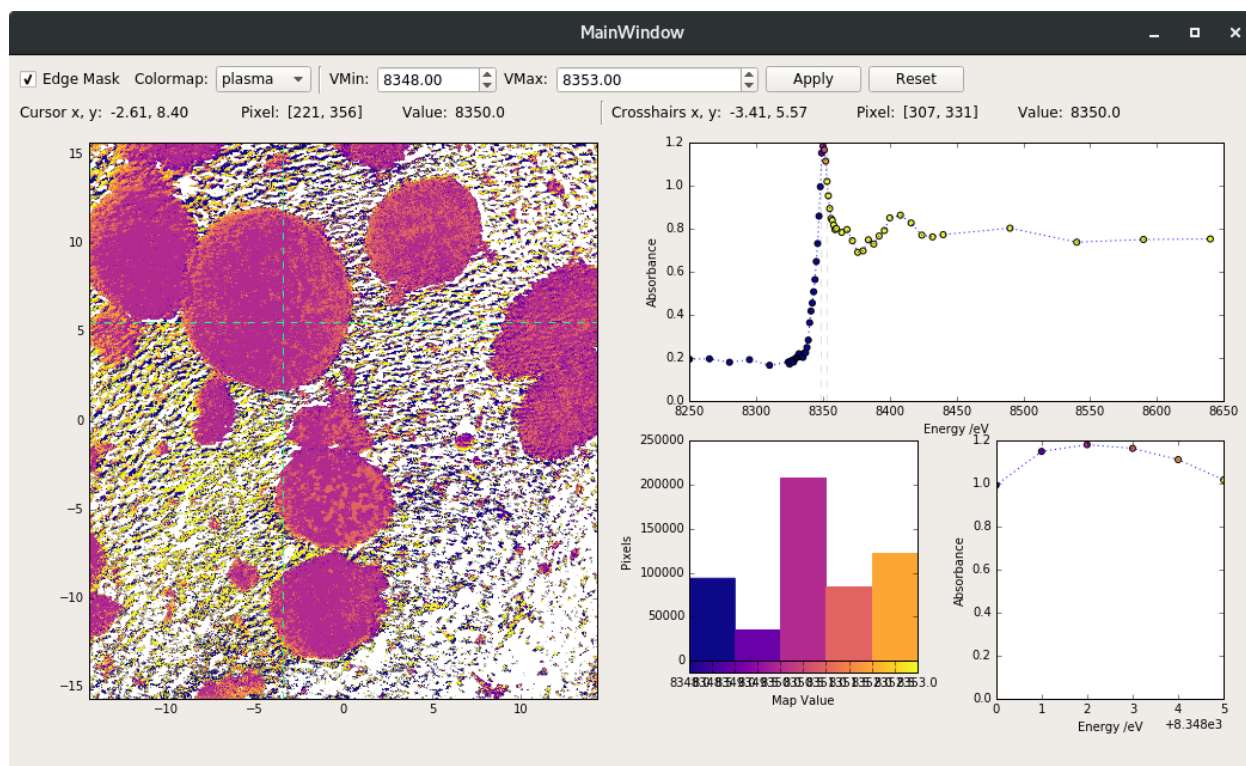
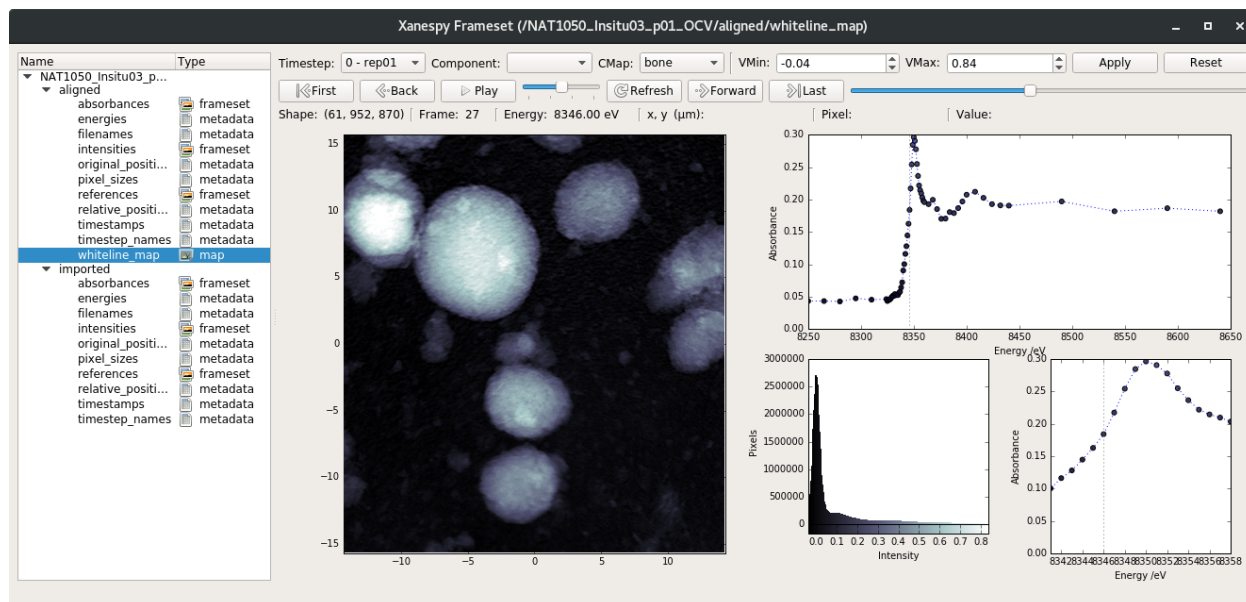


Fig. 1: Screenshots of the Qt viewer. Frame window (top) and map window (bottom).

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)
```


7.1 xanespy package

7.1.1 Submodules

7.1.2 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 preferred 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 zoneplate 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 microscope 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.ssr16_xanes_script (dest, edge: edges.KEdge, zoneplate:
xanespy.beamlines.Zoneplate, positions:
List[utilities.position], reference_position: utilities.position, iterations: Iterable[T_co], 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 microscope 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)
```

7.1.3 xanespy.edges module

Descriptions of X-ray energy absorption edge.

```
class xanespy.edges.CuKEdge
    Bases: xanespy.edges.KEdge
```

```
    E_0 = 8978.9
    edge_range = (8970, 9010)
    map_range = (8970, 9010)
    name = 'Cu'
    post_edge = (9010, 9200)
    pre_edge = (8940, 8970)
    regions = []
    shell = 'K'
```

```
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
The energy of the absorption edge itself.

Type float

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

Type list of 3-tuples

name
A human-readable name for this edge (eg “Ni K-edge”)

Type string

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

Type 2-tuple

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

Type 2-tuple

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

Type 2-tuple

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

Type 2-tuple

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.FeKEdge
    Bases: xanespy.edges.KEdge

    E_0 = 7100.0
    edge_range = (7115, 7140)
    map_range = (7100, 7200)
    name = 'Fe'
    post_edge = (7150, 7250)
    pre_edge = (7100, 7108)
    regions = [(7100, 7110, 3), (7110, 7117, 1), (7117, 7130, 5), (7130, 7200, 5)]
    shell = 'K'

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

    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().

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

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().

    shell = 'L'

```

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

    name = 'Mn_LMO'
    regions = [(6450, 6510, 20), (6524, 6542, 2), (6544, 6564, 1), (6566, 6568, 2), (6572,

class xanespy.edges.NCACobaltKEdge
    Bases: xanespy.edges.KEdge

    E_0 = 7712
    edge_range = (7715, 7740)
    map_range = (7725, 7735)
    name = 'Co_NCA'
    post_edge = (7780, 7900)
    pre_edge = (7600, 7715)
    shell = 'K'

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

    E_0 = 793.2
    edge_range = (775, 785)
    map_range = (0, 1)
    name = 'Co_NCA'
    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)
    name = 'Ni_NCA'
    post_edge = (8350, 8500)
    pre_edge = (8249, 8320)
    regions = [(8250, 8310, 20), (8324, 8344, 2), (8344, 8356, 1), (8356, 8360, 2), (8360,
    shell = 'K'

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

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

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

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



```

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

    E_0 = 853

    edge_range = (848, 857)

    map_range = (0, 1)

    name = 'Ni_NCA'

    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

```

7.1.4 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.FrameSourceError`

Bases: `KeyError`

The frame-source attribute is not present.

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`

7.1.5 xanespy.fitting module

A collection of callables that can be used for fitting spectra against.

`xanespy.fitting.prepare_p0` (*p0*, *frame_shape*, *num_timesteps*=1)

Create an initial parameter guess for fitting.

Takes a starting guess (*p0*) and returns a numpy array with this initial guess that matches the frameset.

For example, if a frameset has 12 timesteps and (1024, 2048) frames, then a 5-tuple input for *p0* will result in a return value with shape (12, 5, 1024, 2048)

`xanespy.fitting.fit_spectra` (*observations*, *func*, *p0*, *nonnegative*=False, *quiet*=False)

Fit a function to a series of observations.

The shapes of *observations* and *p0* parameters must match in the first dimension, and the callable *func* should take a series of parameters (the exact number is determined by the last dimension of *p0*) and return a set of observations (the length of which is determined by the last dimension of *observations*).

Parameters

- **observations** (*np.ndarray*) – A 1- or 2-dimensional array of observations against which to fit the function *func*.
- **func** (*callable*, *str*) – The function that will be used for fitting. It should match *func*(*p0*, *p1*, ...) where *p0*, *p1*, etc are the fitting parameters. Some useful functions can be found in the `xanespy.fitting` module.
- **p0** (*np.ndarray*) – Initial guess for parameters, with similar dimensions to a frameset. Example, fitting 3 sources (plus offset) for a (1, 40, 256, 256) 40-energy frameset requires *p0* to be (1, 4, 256, 256).
- **nonnegative** (*bool*, *optional*) – If true (default), negative parameters will be avoided. This can also be a tuple to allow for fine-grained control. Eg: (True, False) will only punish negative values in the first of the two parameters.
- **quiet** (*bool*, *optional*) – Whether to suppress the progress bar, etc.

Returns

- **params** (*numpy.ndarray*) – The fit parameters (as frames) for each source.
- **residuals** (*numpy.ndarray*) – Residual error after fitting, as maps.

```
class xanespy.fitting.Curve
```

Bases: object

Base class for a callable Curve.

```
name = 'curve'
```

```
param_names = ()
```

```
class xanespy.fitting.Line(x)
```

Bases: *xanespy.fitting.Curve*

```
class xanespy.fitting.LinearCombination(sources)
```

Bases: *xanespy.fitting.Curve*

Combines other curves into one callable.

The constructor accepts the keyword argument `sources`, which should be a list of numpy arrays. The resulting object can then be called with parameters for the weight of each function plus an offset. For example, with two sources, the object is called as

```
# Prepare the separate sources
x = np.linspace(0, 2*np.pi, num=361)
sources = [np.sin(x), np.sin(2*x)]

# Produce a combo with 0.5*sin(x) + 0.25*sin(2x) + 2
lc = LinearCombination(sources=sources)
out = lc(0.5, 0.25, 2)
```

The final output will have the same shape as the sources, which should all be the same shape as each other.

```
name = 'linear_combination'
```

```
param_names
```

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

```
class xanespy.fitting.Gaussian(x)
```

Bases: *xanespy.fitting.Curve*

A Gaussian curve.

Mathematically:

$$y = ae^{\frac{-(x-b)**2}{2c^2}}$$

Parameters `x` (*np.ndarray*) – Array of x-values to input into the Gaussian function.

```
name = 'gaussian'
```

```
param_names = ('height', 'center', 'width')
```

```
class xanespy.fitting.L3Curve(energies, num_peaks=2)
```

Bases: *xanespy.fitting.Curve*

An L₃ absorption edge.

This function is a combination of two Gaussian peaks and a step function. The first 3 parameters give the height, position and width of one peak, and parameters 3:6 give the same for a second peak. Parameters 6:9 are height, position and width of an arctan step function. Parameter 9 is a global offset.

Parameters `peaks` (*int*, *optional*) – How many peaks to fit across the edge.

`name = 'L3-gaussian'`

param_names

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

class `xanespy.fitting.KCurve` (*energies*)

Bases: `xanespy.fitting.Curve`

A K absorption edge.

Fit Parameters:

scale Overall scale factor for curve

voffset Overall vertical offset for the curve

E0 Edge position as energy of maximum in second derivative at edge

sigw Sharpness of the edge sigmoid

bg_slope Linear increase/-decrease in background optical depth

ga Height parameter for Gaussian whiteline peak

gb Center parameter in eV (relative to E0) for Gaussian whiteline peak

gc Width parameter for Gaussian whiteline peak

guess_params (*intensities*, *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

- **intensities** (`np.ndarray`) – An array containing optical_depth data that represents a K-edge spectrum. Only 1-dimensional data are currently accepted.
- **edge** (`xanespy.edges.KEdge`) – An X-ray Edge object, will be used for estimating the actual edge energy itself.

Returns `p0` – A named tuple with the estimated parameters (see `KEdgeParams` for definition)

Return type tuple

`name = 'K-edge-curve'`

`param_names = ('scale', 'voffset', 'E0', 'sigw', 'bg_slope', 'ga', 'gb', 'gc')`

7.1.6 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_aps32idc_xanes_file(filename, hdf_filename, hdf_groupname,
                                              timestep=0, total_timesteps=1, ap-
                                              pend=False, downsample=1, square=True,
                                              exclude=[], median_filter_size=(1, 3, 3),
                                              dark_idx=slice(None, None, None))
```

Import XANES data from a HDF5 file produced at APS beamline 32-ID-C.

This is used for importing a single XANES dataset.

Parameters

- **filename** (*str*) – The path to the HDF5 file containing the source data.
- **hdf_filename** (*str*) – The path to the HDF5 file that will receive the imported data. Will be created if it doesn't exist.
- **hdf_groupname** (*str, optional*) – A description of the dataset that will be used to form the HDF data group.
- **timestep** (*int, optional*) – Which timestep index to use for saving data.
- **total_timesteps** (*int, optional*) – How many timesteps to use for creating new datasteps. Only meaningful if append is truthy.
- **append** (*bool, optional*) – If true, existing datasets will be saved and only the timestep will be overwritten.
- **downsample** (*int, optional*) – Improves signal-to-noise at the expense of spatial resolution. Applied to intensities, flat-field, etc before converting to optical_depth.
- **square** (*bool, optional*) – If true (default), the edges will be cut to make a square array. Eg (2048, 2448) becomes (2048, 2048).
- **exclude** (*iterable, optional*) – Indices of frames to exclude from importing if, for example, the frame contains artifacts or is otherwise problematic.
- **median_filter_size** (*float or tuple*) – If not None, apply a median rank filter to each flat and data frame. The value of this parameters matches the size parameter to `scipy.ndimage.filters.median_filter()`, for example using (1, 3, 3) will filter only along the *x* and *y* axes, and not the energy axis. Median filtering takes places after downsampling.
- **dark_idx** (*slice, optional*) – A slice object (or an index) for which dark-field images to use. Must be the same for all timesteps in an experiment. Useful if some dark field images are not usable.

```
xanespy.importers.import_aps32idc_xanes_files(filename, hdf_filename, hdf_groupname,
                                              *args, **kwargs)
```

Import XANES data from a HDF5 file produced at APS beamline 32-ID-C.

This is used for importing a full operando experiment at once.

Parameters

- **filenames** (*str*) – List of paths to the HDF5 files containing the source data.
- **hdf_filename** (*str*) – The path to the HDF5 file that will receive the imported data. Will be created if it doesn't exist.
- **hdf_groupname** (*str, optional*) – A description of the dataset that will be used to form the HDF data group.
- **kwargs** (*args,*) – Passed to `import_aps32idc_xanes_file`

```
xanespy.importers.import_aps4idc_sxstm_files (filenames, hdf_filename, hdf_groupname,  
                                             shape, energies, flux_correction=True)
```

Import scanning X-ray tunneling microscopy absorbance frames.

These frames are STM images from APS 4-ID-C with incident X-rays at increasing energies, providing pixel-resolved spectral data. If the files are all in one directory with no other data, then the `filenames` argument can be the directory name, otherwise it should be a list of the filenames to import. It is assumed that the fast axis is energy, then the two spatial axes.

Parameters

- **filenames** (*list, str*) – List of filenames to import relative to working directory. Alternately, it can be a string with a directory path and all files will be imported.
- **hdf_filename** (*str*) – Path to a filename to use for saving imported data. If it doesn't exist, it will be created.
- **hdf_groupname** (*str*) – HDF groupname to use for saving data. If it exists, it will be overwritten.
- **shape** (*2-tuple*) – Shape for the resulting maps.
- **energies** (*iterable*) – Incident beam energies for each frame, in electron-volts.
- **flux_correction** (*bool, optional*) – If true, certain channels will be corrected to account for changing beam flux.

```
xanespy.importers.import_aps8bm_xanes_dir (directory, hdf_filename, groupname=None,  
                                           *args, **kwargs)
```

```
xanespy.importers.import_aps8bm_xanes_file (filename, ref_filename, hdf_filename, group-  
                                             name=None, quiet=False)
```

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. '{}' can be included and will receive the position name using the `format()` method. The '{}' is required if more than one field of view exists and a groupname is given.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar, etc.

```
xanespy.importers.import_frameset (directory, flavor, hdf_filename, groupname=None, re-  
                                   turn_val=None, quiet=False)
```

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. Will overwrite previous data-sets with the same name.
- **groupname** (*str, optional*) – What to use as the name for HDF group storing the data. If omitted, guess the name from the directory path. ‘{’ can be included and will receive the position name using the `format()` method. The ‘{’ is required if more than one field of view exists and a groupname is given.
- **return_val** (*str, optional*) – Request a specific return value. - None: No return value (default) - “group”: The open HDF5 group for this experiment.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar

```
xanespy.importers.import_nanosurveyor_frameset(directory: str, quiet=False,
                                              hdf_filename=None,
                                              hdf_groupname=None, energy_range=None,
                                              exclude_re=None, append=False, frame_shape=None)
```

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)
- **frame_shape** (*2-tuple, optional*) – If given, images will be trimmed to this shape. Must be smaller than the smallest frame. This may be useful if the frames are slightly different shapes. Does not apply to STXM images.

```
xanespy.importers.import_ssrl_xanes_dir(directory, hdf_filename, groupname=None, *args,
                                         **kwargs)
```

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.
- **groupname** (*str*, *optional*) – HDF group name to use for saving these data. If omitted, try to guess from directory path.
- **kwargs** (*args*,) – Arguments and keyword arguments passed on to
- **import_frameset** . –

```
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 Transmission 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*, *quiet=False*)

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`.
- **quiet** (*bool*, *optional*) – Whether to suppress the progress bar, etc.

7.1.7 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=None*, *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** – Matplotlib axes object against which to plot.
- **cmap** (*str*) – String or mpl Colormap instance indicating which colormap to use.
- **norm** (*matplotlib.Normalize*) – Describes the range of values to use.
- **energies** (*iterable*, *optional*) – Values to put as the tick marks on the colorbar. If not given, 3 points across *norm* will be used.
- **orientation** (*str*, *optional*) – “horizontal” or “vertical” (default)

`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.latexify` ()

Set some custom options for saving matplotlib graphics in PGF format.

`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_kedge_fit(energies, params)`

Plot the fit based on the given k-edge params. The params will be given to `xanespy.xanes_math.KEdgeParams` for conversion.

energies [np.ndarray] A 1D array with the x values to plot.

params [array-like] Fitted K-edge parameters.

`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_spectra(spectra, energies, ax=None)`

Take an iterable of spectra and plot them one above the next.

Parameters

- **spectra** (*iterable*) – Each entry is an array with intensity values.
- **energies** (*np.ndarray*) – Energy values for the points in each entry of *spectra*.
- **ax** (*matplotlib.Axes, optional*) – The Axes object to receive the plots. If omitted, a new Axes will be created.

Returns *artists* – A list of the matplotlib artists used to draw the spectra.

Return type *list*

`xanespy.plots.plot_spectra_as_map(spectra, energies, ax=None, extent=None, **kwargs)`

Take an iterable of spectra and plot them as a heat map.

This function takes energies so it can put ticks on the x-axis, however if they are not equally spaced, their distance on the map will not properly capture they're distance numerically.

Parameters

- **spectra** (*iterable*) – Each entry is an array with intensity values.
- **energies** (*np.ndarray*) – Energy values for the points in each entry of *spectra*.
- **ax** (*matplotlib.Axes, optional*) – The Axes object to receive the plots. If omitted, a new Axes will be created.
- **extent** (*Matplotlib extent.*) –
- ****kwargs** (*Get passed on to matplotlib imshow()*) –

Returns *artists* – A list of the matplotlib artists used to draw the spectra.

Return type *list*

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

Plot an energy spectrum on an axes.

Applies some color formatting if *edge* is a valid XANES Edge object.

Parameters

- **spectrum** (*np.ndarray*) – Array of intensity values.
- **energies** (*np.ndarray*) – Array of energy values.
- **norm** (*optional*) – Matplotlib Normalize() object that shows the map range. This will be used to annotate the plot if it is give.
- **show_fit** (*bool, optional*) – Whether to plot lines showing the best fit.
- **ax** (*mpl.Axes, optional*) – Matplotlib Axes on which to plot. If not given, a new axes will be generated.
- **ax2** (*mpl.Axes, optional*) – A second y-axes for plotting the imaginary component if the data are complex.
- **linestyle** (*optional*) – Passed on to matplotlib.
- **cmap** (*str, optional*) – Colormap, passed on to matplotlib
- **color** (*optional*) – Specifies the color for the circles plotted. Either “x” 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.
- **polar_coords** (*bool, optional*) – If truthy, the spectrum will be plotted as modulus-phase instead of real-imag. For purely real data this is equivalent to taking the absolute value.

```
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 be 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.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.

7.1.8 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 arbitrarily.

`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)`

`animate_frames(frames, energies, norm, cmap, extent)`

`back_button_clicked`

`build_hdf_tree(tree)`

`change_vrange(vmin, vmax, step, decimals)`

Change the range of frame map colors.

Does not affect the rendered frames.

`clear_axes()`

`connect_presenter(presenter)`

`connect_signals(presenter)`

`create_canvas()`

`create_status_bar()`

`draw_histogram` (*data*, *energies*, *norm*, *cmap*)

`draw_spectrum` (*spectrum*, *fitted_spectrum*, *norm*, *cmap*, *edge_range=None*)

`edge_ax` = None

`fig` = None

`figure_hovered`

`first_button_clicked`

`forward_button_clicked`

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

`frame_slider_moved`

`hist_ax` = None

`hist_cb` = None

`last_button_clicked`

`move_slider` (*val*)

`new_cmap_requested`

`new_component_requested`

`new_hdf_file_requested`

`new_hdf_group_requested`

`new_timestep_requested`

`new_vrange_requested`

`open_hdf_file` ()

`play_button_clicked`

`play_speed_requested`

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

`presenter_frame_changed`

`request_new_vrange` (*bool*)

`reset_vrange_requested`

`select_active_hdf_item` (*item*)

`set_busy_mode` (*status*)
Either enable or disable the busy mode.
Changes the mouse pointer and disables/enables controls.
Parameters **status** (*bool*) – If truthy, the UI is disabled, otherwise the UI is re-enabled.

`set_cmap_list` (*cmap_list*)

`set_component_list` (*component_list*)

`set_slider_max` (*val*)

`set_status_energy` (*energy*)

```
set_status_index(msg)
set_status_metadata(frames, energies, norm, cmap, extent)
set_status_unit(msg)
set_timestep(idx)
set_timestep_list(timestep_list)
set_window_title(hdf_path)
setup_ui()
show_status_message(message)
spectrum_ax = None
ui = None
update_figure_pixel(xy, xy_unit, px, value)
window = None
```

7.1.9 xanespy.qt_frameset_presenter module

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

Bases: PyQt5.QtCore.QObject

Presenter for showing XanesFrameset frames and maps via Qt.

app_ready

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

Type pyqtSignal

busy_status_changed

Emitted when processing has started or ended.

Type pyqtSignal

map_data_changed

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

Type pyqtSignal

map_data_cleared

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

Type pyqtSignal

active_energy_changed

active_frame = 0

active_frame_changed

active_frame_component = 'real'

active_frames()

active_map()

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

active_map_component = 'real'

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 snappier UI. Disabling makes testing more straightforward.

app_ready

build_hdf_tree (*expand_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_active_frame (*new_idx*)

change_frame_cmap (*new_cmap*)

change_frame_component (*new_comp*)

change_hdf_file (*filename*)

change_hdf_group (*new_item*, *old_item*)

change_map_cmap (*new_cmap*)

change_map_component (*new_comp*)

cmap_list_changed

component_list_changed

create_app ()

first_frame ()

frame_cmap = ''

frame_data_changed

frame_data_cleared

frame_hover_changed

frame_norm ()

frame_pixel = None

frame_vrange_changed

`hdf_path_changed`
`hdf_tree_changed`
`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.

`next_frame` ()

`num_frames` = 0

`play_frames` (*start*)

`prepare_ui` (*expand_tree=True*)

`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 1st 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_vrange` (*vmin, vmax*)


```

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 an exponential scale.
set_timestep (new_timestep)
show_spectrum_fit = False
timer = None
timestep_list_changed
toggle_edge_mask (state)
toggle_spectrum_fit (state)
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 active complex component (real, imag, etc)
        • 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_spectrum_changed.
update_timestep_list ()
use_edge_mask = False

```

7.1.10 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
    Fires when the user changes the colormap via the UI
    Type signal
limits_applied
    Fires when the user requests that data be redrawn with new limits.
    Type signal

```

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

Type signal

cmap_changed

component_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_component_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)
update_spectrum_ui(spectrum, fitted_spectrum, norm, cmap, edge_range)
window = None

```

7.1.11 xanespy.txmstore module

Tools for accessing TXM data stored in an HDF5 file.

```

class xanespy.txmstore.TXMDataset(name, context=None, dtype=None)
    Bases: object

```

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

```

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 descriptors and properties that return the corresponding HDF5 dataset object; the `TXMStore().attribute.value` pattern can be used to get pure numpy arrays directly. 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

```

```

close()

```

```

cluster_fit

```

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

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.

edge_mask

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

energies

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

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.

frame_source (*name*)

Get the name of the frames that went into creating a map.

frameset_names()

Returns a list of all the valid map representations.

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*)

Return the source frame data for the given data name.

This is similar to `get_dataset` except that if the data are a map, then get the frames that went into making it.

Parameters **name** (*str*) – The dataset name for which to retrieve frames.

Returns dataset – The requested frameset. If the dataset is actually a map, as determined by the “context” attribute, then the related frame source attribute will be retrieved.

Return type h5py.Dataset

has_dataset (*name*)

Return a boolean indicated whether this dataset exists in the HDF file.

intensities

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

intensity_mean

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

latest_data_name

linear_combination_parameters

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

linear_combination_residuals

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

linear_combination_sources

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.

- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

map_names ()

Returns a list of all the valid map representations.

open_file (*filename, mode*)

optical_depth_mean

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

optical_depths

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

original_positions

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

parent_group ()

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

particle_labels

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

pixel_sizes

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.

- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

pixel_unit

references

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

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.
- **compression** (*str, optional*) – What type of compression to use. See HDF5 documentation for options.
- ***args** – Arguments to pass to h5py’s create_dataset method.
- ****kwargs** – Keyword arguments to pass to h5py’s create_dataset method.

segments

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

signal_map

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str, optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype, optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

signal_method

String describing how the previously extracted signals were calculated.

signal_weights

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

signals

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

timestamps**timestep_names**

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

validate_parent_group (*name*)

Retrieve the real parent group name for a possible parent_group.

If *name* is None and only one group exists in the file, then that group name will be returned. If *name* is in the file, then *name* will be returned. If *name* is not in the file, a `GroupKeyError` will be raised.

whiteline_fit

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.
- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

whiteline_max

Data descriptor for accessing HDF datasets.

Parameters

- **name** (*str*) – The dataset name in the HDF file.
- **context** (*str*, *optional*) – Type of dataset this is: frameset, map, metadata, etc.

- **dtype** (*np.dtype*, *optional*) – The data-type to use when saving new data to disk. Using lower precision datatypes can save significant disk space.

7.1.12 xanespy.utilities module

A collection of classes and functions that aren't specific to any one type of measurement. Also, it defines some namedtuples for describing coordinates.

`xanespy.utilities.Extent`

alias of `xanespy.utilities.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.

Parameters

- **data** – Numerical (presumably complex-valued) data to be reduced to a component.
- **name** (*str*) – One of ('modulus', 'phase', 'real', 'imag')

Returns Input data converted to requested component.

Return type 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, smoothing=0.01, *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

7.1.13 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-dimensional dataset with of dimenions (energy, row, column).

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', method: str = 'cross_correlation', template=None, passes=1, median_filter_size=None, commit=True, component='modulus', plot_results=True, results_ax=None, quiet=False*)

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

- **reference_frame** (*2-tuple, str, optional*) – The index of the frame to 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 optical_depth is used. Otherwise, a 2-tuple with (timestep, energy) should be provided. This attribute has no effect if template matching is used.
- **method** (*str, optional*) – 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.)
- **template** (*np.ndarray, optional*) – Image data that should be matched if the *template_match* method is used.
- **passes** (*int, optional*) – How many times this alignment should be done. Default: 1.
- **median_filter_size** (*int, optional*) – If provided, a median filter will be applied to each frame. The value of this parameter determines how large the kernel is: 3 creates a (3, 3) kernel; (3, 5) creates a (3, 5) kernel; etc.
- **commit** (*bool, optional*) – 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.
 - results_ax** : optional If `plot_results` is true, this axes will be used to receive the plot.
 - quiet** : bool, optional Whether to suppress the progress bar, etc.
- **component** (*str, optional*) – For complex data, which component to use: modulus, phase, real, imag.
- **plot_results** (*bool, optional*) – If true, a boxplot will be plot with the RMS shifts for each pass.
- **results_ax** (*mpl.Axes, optional*) – If `plot_results` is true, this Axes will receive the plot. If omitted, a new axes will be created.
- **quiet** (*bool, optional*) – Suppress the progress bar

apply_internal_reference()

Use a portion of each frame for internal reference correction.

This function will extract an I_0 from the background by thresholding. Then calculate the optical depth by

$$OD = \ln\left(\frac{I_0}{I}\right)$$

This method is compatible with complex intensity data.

apply_median_filter(size, representation='optical_depths')

Permanently apply a median filter to a frameset.

Parameters

- **size** (*4-tuple*) – Dimensions of the kernel to use for filtering in order of (time, energy, row, col).
- **representation** (*str, optional*) – Which frameset representation to use for filtering.

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

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

Parameters

- **crop** (*bool, optional*) – If truthy, the images will be cropped after being translated, so there are not edges. If falsy, the images will be wrapped.
- **commit** (*bool, optional*) – If truthy, the changes will be saved to the HDF5 store for optical depths, intensities and references, and the staged transformations will be cleared. Otherwise, only the optical_depth data will be transformed and returned.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar, etc.

Returns **out** – Transformed array of the optical depth frames.

Return type np.ndarray

calculate_maps ()

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

This method does not do any advanced analysis, so something like *~xanespy.xanes_frameset.XanesFrameset.fit_spectra* may be necessary.

calculate_mean_frames ()

calculate_signals (*n_components=2, method='nmf', frame_source='optical_depths', 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 optical_depth.

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 = None

edge_mask

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

Parameters

- **sensitivity** (*optional*) – A multiplier for the otsu value to determine the actual threshold. Higher values give better statistics, but might include the active material, lower values give worse statistics but are less likely to include active material.
- **min_size** (*optional*) – 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

export_beamer_file (*basename, map_names=None, frameset_names=None, time_idx=None, groupby='dataset'*)

Plot maps and create a beamer file.

The resulting beamer file won't be compilable by itself, but should be included in a beamer slide-deck with the `\input{}` directive. The pgf package must be included with `\usepackage{pgf}` in the preamble.

Parameters

- **basename** (*str*) – Filename of the beamer output. Will be overwritten if it exists. Pgf figures will use this as a prefix.
- **map_names** (*list, optional*) – Representations to use for plotting maps. If omitted, anything resembling a map will be used.
- **frameset_names** (*list, optional*) – Representations to use for plotting spectra. If omitted, anything resembling a frameset will be used.
- **time_idx** (*list, optional*) – Indices of timesteps to use for plotting maps. If omitted, all timesteps will be used.
- **dataset** (*str, optional*) – Determines which dimension changes quickly. Options are “dataset” and “timestep”.

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

fit_linear_combinations (*sources*, *component='real'*, *name='linear_combination'*, *representation='optical_depths'*, **args*, ***kwargs*)

Take a set of sources and fit the spectra with them.

Saves to the representation “linear_combinations”. Also creates “linear_combination_sources” and “linear_combination_residuals” datasets.

Parameters

- **sources** (*numpy.ndarray*) – Sources to use for fitting the combinations.
- **component** (*str*, *optional*) – Complex component to use before fitting.
- **name** (*str*, *optional*) – What to call the resulting dataset in the hdf file.
- **representation** (*str*, *optional*) – What dataset to use as input for fitting.
- **kwargs** (*args*,) – Passed on to `self.fit_spectra`.

Returns

- **fits** (*numpy.ndarray*) – The weights (as frames) for each source.
- **residuals** (*numpy.ndarray*) – Residual error after fitting, as maps.

fit_spectra (*func*, *p0*, *pnames=None*, *name=None*, *edge_filter=True*, *nonnegative=False*, *component='real'*, *representation='optical_depths'*, *dtype=None*, *quiet=False*)

Fit a given function to the spectra at each pixel.

The fit parameters will be saved in the HDF dataset “{name}_params” based on the parameter `name`. RMS residuals for each pixel will be saved in “{name}_residuals”.

Parameters

- **func** (*callable*, *str*) – The function that will be used for fitting. It should match `func(p0, p1, ...)` where `p0`, `p1`, etc are the fitting parameters. Some useful functions can be found in the `xanespy.fitting` module.
- **p0** (*np.ndarray*) – Initial guess for parameters, with similar dimensions to a frameset. Example, fitting 3 sources (plus offset) for a (1, 40, 256, 256) 40-energy frameset requires `p0` to be (1, 4, 256, 256).
- **pnames** (*str*, *optional*) – An object with `__str__` that will be saved as metadata giving the parameters’ names.
- **name** (*str*, *optional*) – What to call this fit in the HDF5 file. Use this to allow subsequent fits against the same dataset to be saved. If `None`, we will attempt look for `func.name`, then lastly we’ll use “fit”.

- **edge_filter** (*bool, optional*) – If true, only compute pixels that pass an edge filter.
- **nonnegative** (*bool, optional*) – If true (default), negative parameters will be avoided. This can also be a tuple to allow for fine-grained control. Eg: (True, False) will only punish negative values in the first of the two parameters.
- **component** (*str, optional*) – What to use for complex-valued functions.
- **representation** (*str, optional*) – Which set of frames to use for fitting.
- **dtype** (*numpy.dtype, optional*) – Specify a datatype to convert all values to. This helps avoid fit failure due to precision errors. If omitted, the function will also check for `func.dtype`.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar, etc.

Returns

- **params** (*numpy.ndarray*) – The fit parameters (as frames) for each source.
- **residuals** (*numpy.ndarray*) – Residual error after fitting, as maps.

fitting_param_names (*representation='fit_parameters'*)

Get the human-readable names of the fit parameters.

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='optical_depths'*)

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 “optical_depths”, “whiteline_map”, “intensities”

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

Return type `np.ndarray`

gui_viewer ()

Launch a Qt GUI for inspecting the data.

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 (*int, optional*) – Controls how selective the algorithm is at grouping areas into particles. Lower numbers means more particles, but might split large particles into two.

line_spectra (*xy0, xy1, representation='optical_depths', timestep=0*)

Return an array of spectra on a line between two points.

This is effectively nearest neighbor interpolation between two (x, y) pairs on the frames.

Returns spectra – An array of spectra, one for each point on the line.

Return type np.ndarray

Parameters

- **xy0** (*2-tuple*) – Starting point for the line.
- **xy1** (*2-tuple*) – Ending point for the line.

map_data

Return map data for the given time index and representation.

If *representation* is not 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 “optical_depths”, “whiteline_map”, “intensities”

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

Return type np.ndarray

mean_frame (*representation='optical_depths'*)

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

parent_name = None

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_size (*representation='optical_depths', timestep=0*)

Return the size of the pixel (with units set by `pixel_unit`).

pixel_unit ()

Return the unit of measure for the size of a pixel.

plot_beamer_maps (*basename, map_names=None, time_idx=None, groupby='dataset', tight_layout=True*)

Export maps to pgf and prepare a jinja context.

This method is similar to `export_beamer_file` except that it doesn't actually create the .tex file. This can be useful if you desire more control over the beamer layout.

Parameters

- **basename** (*str*) – The base name to be used for generating the .pgf files.
- **map_names** (*list, optional*) – Representations to use for plotting maps. If omitted, anything resembling a map will be used.
- **time_idx** (*list, optional*) – Indices of timesteps to use for plotting maps. If omitted, all timesteps will be used.
- **groupby** (*str, optional*) – Determines which dimension changes quickly. Options are “dataset” and “timestep”.
- **tight_layout** (*bool, optional*) – Whether to call the `matplotlib.tight_layout()` function, useful for things with colorbars, etc.

Returns context – Context dictionary appropriate for passing into the jinja2 template engine.

Return type dict

plot_beamer_spectra (*basename, frameset_names=None, time_idx=None, groupby='dataset'*)

Export spectra to pgf and prepare a jinja context.

This method is similar to `export_beamer_file` except that it doesn't actually create the .tex file. This can be useful if you desire more control over the beamer layout.

Parameters

- **basename** (*str*) – The base name to be used for generating the .pgf files.
- **frameset_names** (*list, optional*) – Representations to use for plotting maps. If omitted, anything resembling a map will be used.
- **time_idx** (*list, optional*) – Indices of timesteps to use for plotting maps. If omitted, all timesteps will be used.
- **groupby** (*str, optional*) – Determines which dimension changes quickly. Options are “dataset” and “timestep”.

Returns context – Context dictionary appropriate for passing into the jinja2 template engine.

Return type dict

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_line_scans (*representation='optical_depths'*, *direction='horizontal'*, *idx=None*, *ax=None*, *form='lines'*, *time_idx=0*)

Plot spectrum for each point on a line.

The *direction* and *idx* parameters control which lines are scanned. Eg. *direction="vertical"* and *idx=3* will plot a spectrum for each position in the 4th column.

Parameters

- **representation** (*str*, *optional*) – Which frameset to use for plotting.
- **direction** (*str*, *optional*) – Whether to do lines scans in “horizontal” (default) or “vertical” orientation.
- **idx** (*int*, *optional*) – Which row or column to use for line scans. If omitted, frames will be averaged across the unchanging dimension.
- **ax** (*mpl.Axes*, *optional*) – Matplotlib axes to use for plotting. If omitted, a new Axes will be created.
- **form** (*str*, *optional*) – How to plot the data: ‘lines’ or ‘map’.
- **timeidx** (*int*, *optional*) – Which time step to use for plotting line scans.

Returns List of line of image artists return from plotting command

Return type artists

plot_map (*ax=None*, *map_name='whiteline_fit'*, *timeidx=0*, *vmin=None*, *vmax=None*, *median_size=0*, *component='real'*, **args*, ***kwargs*)

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', representation='optical_depths', 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_spectrum (*ax=None, pixel=None, norm_range=None, normalize=False, representation='optical_depths', show_fit=False, edge_jump_filter=False, linestyle=':', timeidx=0, *args, **kwargs*)

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

Parameters

- **ax** (*optional*) – matplotlib axes object on which to draw
- **pixel** (*2-tuple, optional*) – Coordinates of a specific pixel on the image to plot.
- **normalize** (*bool, optional*) – If truthy, will set the pre-edge at zero and the post-edge at 1.
- **show_fit** (*bool, optional*) – If truthy, will use the edge object to fit the data and plot the resulting fit line.
- **edge_jump_filter** (*bool, optional*) – If truthy, will only include those values that show a strong absorption jump across this edge.
- **kwargs** (*args,*) – Passed to plotting functions.

segment_materials (*thresholds, representation='optical_depths', component='real'*)

Split the frames into different materials based on mean values.

This is most useful with a metric that is unique to a given material. For example, the phase representation of ptychography data can be used to determine which material is which. Results will be stored in `XanesFrameset().store().segments`.

Parameters

- **thresholds** (*tuple*) – The boundary values to use for segmentation. If N thresholds are given, the data will be split into N+1 segments.
- **representation** (*str, optional*) – What kind of data the thresholds represent.
- **component** (*str, optional*) – Complex-value representation to use. Default is real value. For ptychography data, 'phase' is probably better.

spectra (*edge_filter=False*)

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

Parameters (**bool or 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='optical_depths', index=0, derivative=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_dataset()`
- **index** (*int, optional*) – Which step in the frameset to use. When used to index `store().optical_depths`, this should return a 3D array like (energy, rows, columns).
- **derivative** (*int, optional*) – Calculate a derivative of the spectrum before returning it. If less than 1 (default), no derivative is calculated.

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.optical_depths[0,0]
```

subtract_surroundings (*sensitivity: float = 1.0*)

Use the edge mask to separate “surroundings” from “sample”, then subtract the average surrounding optical_depth from each frame. This effectively removes effects where the entire frame is brighter from one energy to the next.

Parameters **sensitivity** (*optional*) – A multiplier for the otsu value to determine the actual threshold. Higher values give better statistics, but might include the active material, lower values give worse statistics but are less likely to include active material.

timestamps ()

Retrieve an array with the timestamp for each scan.

This will be a numpy array with the *datetime64* dtype. Each frame has both a start and end time-stamp. Depending on the beamline, the individual energy frames might have timestamps based on the whole XANES scan. These timestamps are in UTC since numpy’s *datetime64* has no timezone support.

Returns **timestamps** – The timestamps for each energy frame with a shape described as (num_timesteps, num_energies, 2), where the last axis is for the start and end of the frame.

Return type *np.ndarray*

timestep_names

7.1.14 xanespy.xanes_math module

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

class *xanespy.xanes_math.FramesPool* (*processes=None, initializer=None, initargs=(), max_tasksperchild=None, context=None*)

Bases: *multiprocessing.pool.Pool*

A multiprocessing pool that iterates over frames even if there are extra dimensions.

Parameters **processes** (*int, optional*) – How many processes to spawn. If omitted, twice the number of CPUs will be used.

map (*func, frames, spatial_dims=2, chunksize=None, desc=None*)

Apply the function to each frame in *frames* and return the result.

Parameters

- **func** (*callable*) – Function to be applied to each frame.
- **frames** (*np.ndarray*) – Array of frames to operate on. The last dimensions are considered spatial dimensions.
- **spatial_dims** (*int, optional*) – How many of the last dimensions are spatial frame dimensions, 2D, 3D, etc.
- **chunksize** (*int, optional*) – How many frames to process at a time.
- **desc** (*str, optional*) – If provided, a progress bar will be displayed.

Returns result – The result of all calculations. Similar in shape to *frames* except that the spatial dimensions are replaced by whatever is returned by *func*.

Return type *np.ndarray*

xanespy.xanes_math.apply_internal_reference (*intensities, out=None, quiet=False*)

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: $\ln(I/I_0)$ where I_0 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, quiet=False*)

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

The formula $-\ln \frac{\text{intensities}}{\text{references}}$ 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.
- **quiet** (*bool, optional*) – Whether to suppress progress bar, etc.

xanespy.xanes_math.direct_whitelines (*spectra, energies, edge, quiet=False*)

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

Parameters

- **spectra** (*np.array*) – 2D numpy array of optical_depth 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 absorption edge in question.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar, etc.

Returns out – Array with the whiteness position of each spectrum.

Return type np.ndarray

`xanespy.xanes_math.downsample_array(arr, factor, method='mean', axis=None)`

Reduced the shape of an image in powers of 2.

Increases in the signal-to-noise can be achieved by sacrificing spatial resolution: a (1024, 1024) image can be converted to (512, 512) by taking the mean of each (2, 2) block. The `factor` parameter controls how aggressive this conversion is. Extra pixels get dropped: eg (1025, 1024) -> (512, 512).

arr	factor	block	out
(1024, 1024) (1024, 1024) (1024, 1024) (1024, 1024)	0 1 2 3	N/A (2, 2) (4, 4) (8, 8)	(1024, 1024) (512, 512) (256, 256) (128, 128)

Parameters

- **arr** (*np.ndarray*) – Input array to be downsampled.
- **factor** (*int*) – Downsampling factor. A value of 0 returns the original array.
- **method** (*str, optional*) – How to convert a (eg 2x2) block to a single value. Valid choices are 'mean' (default), 'median', 'sum'.
- **axis** (*tuple, optional*) – Which axes to use for reduction. If omitted, all axes will be used.

Raises

- `ValueError` – Negative downsampling factors.
- `ValueError` – `method` parameter is not one of the valid options.

Returns out – The downsample array.

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** (*numpy.ndarray*) – A numpy array of observations where the last axis is energy.
- **n_components** (*int*) – How many components to extract from the data.
- **nmf_kwargs** (*dict, optional*) – Keyword arguments to be passed to the constructor of the estimator.

Returns

- **components, weights** (*numpy.ndarray*) – Extracted components and weights for each pixel-component
- *combination.*

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

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.

Parameters

- **data** (*np.ndarray*) – Data to iterate over.
- **leftover_dims** (*int*, *optional*) – Integer describing which dimensions should not be iterated over. Eg. if data is 3D array and leftover_dims == 1, only first two dimensions will be iterated.
- **desc** (*str*, *optional*) – String to put in the progress bar.
- **quiet** (*bool*, *optional*) – Whether to suppress the progress bar, etc.

`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 dimension 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** (*np.ndarray*) – Array with images at different energies.
- **energies** (*np.ndarray*) – X-ray energies corresponding to images in *frames*. Must have the same shape along the first dimension as *frames*.
- **edge** (`xanespy.edges.Edge`) – An Edge 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.
- **frame_dims** (*int, optional*) – The number of dimensions that each frame has. Eg, 2 means each frame is a two-dimensional image.
- **min_size** (*float, 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 np.ndarray

`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.register_correlations` (*frames, reference, upsample_factor=10, desc='Registering', median_filter_size=None*)

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. None will suppress output.
- **median_filter_size** (*int, optional*) – If provided, a median filter will be applied to each frame. The value of this parameter determines how large the kernel is: 3 creates a (3, 3) kernel; (3, 5) creates a (3, 5) kernel; etc.

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', median_filter_size=None*)

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. None will suppress output.
- **median_filter_size** (*int, optional*) – If provided, a median filter will be applied to each frame. The value of this parameter determines how large the kernel is: 3 creates a (3, 3) kernel; (3, 5) creates a (3, 5) kernel; etc.

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', quiet=False*)

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_matrices()` 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 compatibly 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.
- **quiet** (*bool, optional*) – Whether to suppress the progress bar, etc.

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`

7.1.15 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 microscopy 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 choice:

- **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 40 μ m field-of-view.

7.1.16 Module contents

CHAPTER 8

Indices and tables

- `genindex`
- `modindex`
- `search`

X

- [xanespy](#), 72
- [xanespy.beamlines](#), 23
- [xanespy.edges](#), 26
- [xanespy.exceptions](#), 29
- [xanespy.fitting](#), 30
- [xanespy.importers](#), 32
- [xanespy.plots](#), 37
- [xanespy.qt_frame_view](#), 40
- [xanespy.qt_frameset_presenter](#), 42
- [xanespy.qt_map_view](#), 45
- [xanespy.txmstore](#), 47
- [xanespy.utilities](#), 53
- [xanespy.xanes_frameset](#), 54
- [xanespy.xanes_math](#), 65
- [xanespy.xradia](#), 71

A

- active_energy_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_frame (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_frame_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_frame_component (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_frames() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 42
- active_group (xanespy.xanes_frameset.XanesFrameset attribute), 54
- active_map() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 42
- active_map_component (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_representation (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42
- active_timestep (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
- add_callback() (xanespy.qt_frame_view.FrameChangeSource method), 40
- add_frame_view() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 43
- add_hdf_tree_item() (xanespy.qt_frame_view.QtFrameView method), 40
- add_map_view() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 43
- align_frames() (xanespy.xanes_frameset.XanesFrameset method), 54
- all_energies() (xanespy.edges.Edge method), 26
- animate_frames() (xanespy.qt_frame_view.QtFrameView method), 40
- annotate_spectrum() (xanespy.edges.KEdge method), 27
- annotate_spectrum() (xanespy.edges.LEdge method), 27
- app_ready (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 42, 43
- apply_internal_reference() (in module xanespy.xanes_math), 66
- apply_internal_reference() (xanespy.xanes_frameset.XanesFrameset method), 55
- apply_median_filter() (xanespy.xanes_frameset.XanesFrameset method), 55
- apply_mosaic_reference() (in module xanespy.xanes_math), 66
- apply_references() (in module xanespy.xanes_math), 66
- apply_transformations() (xanespy.xanes_frameset.XanesFrameset method), 56
- aps_old1_regex (xanespy.xradia.XRMFile attribute), 71

B

- back_button_clicked (xanespy.qt_frame_view.QtFrameView attribute), 40
- big_axes() (in module xanespy.plots), 37

`binning()` (*xanespy.xradia.XRMFile* method), 71
`broadcast_reverse()` (in module *xanespy.utilities*), 53
`build_hdf_tree()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`build_hdf_tree()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`busy_status_changed` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 42, 43

C

`calculate_maps()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`calculate_mean_frames()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`calculate_signals()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`calculate_whitelines()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`callbacks` (*xanespy.qt_frame_view.FrameChangeSource* attribute), 40
`change_active_frame()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_frame_cmap()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_frame_component()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_hdf_file()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_hdf_group()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_map_cmap()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_map_component()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`change_vrange()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`clear_axes()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`clear_caches()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`close()` (*xanespy.txmstore.TXMStore* method), 47
`close()` (*xanespy.xradia.XRMFile* method), 71
`cluster_fit` (*xanespy.txmstore.TXMStore* attribute), 47
`cmap` (*xanespy.xanes_frameset.XanesFrameset* attribute), 56
`cmap_changed` (*xanespy.qt_map_view.QtMapView* attribute), 45, 46
`cmap_list_changed` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 43
`columns` (*xanespy.utilities.shape* attribute), 54
`component_changed` (*xanespy.qt_map_view.QtMapView* attribute), 46
`component_list_changed` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 43
`components()` (*xanespy.xanes_frameset.XanesFrameset* method), 56
`connect_presenter()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`connect_presenter()` (*xanespy.qt_map_view.QtMapView* method), 46
`connect_signals()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`create_app()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 43
`create_canvas()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`create_canvas()` (*xanespy.qt_map_view.QtMapView* method), 46
`create_status_bar()` (*xanespy.qt_frame_view.QtFrameView* method), 40
`CreateGroupError`, 29
`crosshairs` (*xanespy.qt_map_view.QtMapView* attribute), 46
`CuKEdge` (class in *xanespy.edges*), 26
`Curve` (class in *xanespy.fitting*), 30

D

`data_group()` (*xanespy.txmstore.TXMStore* method), 47
`data_name` (*xanespy.txmstore.TXMStore* attribute), 48

`data_name` (*xanespy.xanes_frameset.XanesFrameset attribute*), 57
`data_tree()` (*xanespy.txmstore.TXMStore method*), 48
`data_tree()` (*xanespy.xanes_frameset.XanesFrameset method*), 57
`DataFormatError`, 29
`DataNotFoundError`, 29
`DatasetExistsError`, 29
`decode_aps_params()` (*in module xanespy.importers*), 32
`decode_ssrl_params()` (*in module xanespy.importers*), 32
`Detector` (*class in xanespy.beamlines*), 23
`DetectorPoint` (*class in xanespy.beamlines*), 23
`direct_whitelines()` (*in module xanespy.xanes_math*), 66
`downsample_array()` (*in module xanespy.xanes_math*), 67
`draw_colorbar()` (*in module xanespy.plots*), 37
`draw_histogram()` (*xanespy.qt_frame_view.QtFrameView method*), 40
`draw_histogram_colorbar()` (*in module xanespy.plots*), 37
`draw_spectrum()` (*xanespy.qt_frame_view.QtFrameView method*), 41
`dual_axes()` (*in module xanespy.plots*), 37

E

`E_0` (*xanespy.edges.CuKEdge attribute*), 26
`E_0` (*xanespy.edges.Edge attribute*), 26
`E_0` (*xanespy.edges.FeKEdge attribute*), 27
`E_0` (*xanespy.edges.NCACobaltKEdge attribute*), 28
`E_0` (*xanespy.edges.NCACobaltLEdge attribute*), 28
`E_0` (*xanespy.edges.NCANickelKEdge attribute*), 28
`E_0` (*xanespy.edges.NCANickelLEdge attribute*), 29
`Edge` (*class in xanespy.edges*), 26
`edge` (*xanespy.xanes_frameset.XanesFrameset attribute*), 57
`edge_ax` (*xanespy.qt_frame_view.QtFrameView attribute*), 41
`edge_mask` (*xanespy.txmstore.TXMStore attribute*), 48
`edge_mask` (*xanespy.xanes_frameset.XanesFrameset attribute*), 57
`edge_mask_toggled` (*xanespy.qt_map_view.QtMapView attribute*), 46
`edge_range` (*xanespy.edges.CuKEdge attribute*), 26
`edge_range` (*xanespy.edges.Edge attribute*), 26, 27
`edge_range` (*xanespy.edges.FeKEdge attribute*), 27

`edge_range` (*xanespy.edges.NCACobaltKEdge attribute*), 28
`edge_range` (*xanespy.edges.NCACobaltLEdge attribute*), 28
`edge_range` (*xanespy.edges.NCANickelKEdge attribute*), 28
`edge_range` (*xanespy.edges.NCANickelLEdge attribute*), 29
`endtime()` (*xanespy.xanes_frameset.XanesFrameset method*), 57
`endtime()` (*xanespy.xradia.XRMFile method*), 72
`energies` (*xanespy.txmstore.TXMStore attribute*), 48
`energies` (*xanespy.xanes_frameset.XanesFrameset attribute*), 57
`energies()` (*xanespy.xradia.TXRMFile method*), 71
`energies_in_range()` (*xanespy.edges.Edge method*), 27
`energy` (*xanespy.beamlines.DetectorPoint attribute*), 23
`energy` (*xanespy.beamlines.ZoneplatePoint attribute*), 24
`energy()` (*xanespy.xradia.XRMFile method*), 72
`export_beamer_file()` (*xanespy.xanes_frameset.XanesFrameset method*), 57
`Extent` (*in module xanespy.utilities*), 53
`extent` (*xanespy.xanes_frameset.XanesFrameset attribute*), 57
`extract_signals_nmf()` (*in module xanespy.xanes_math*), 67

F

`FeKEdge` (*class in xanespy.edges*), 27
`fig` (*xanespy.qt_frame_view.QtFrameView attribute*), 41
`fig` (*xanespy.qt_map_view.QtMapView attribute*), 46
`figure_hovered` (*xanespy.qt_frame_view.QtFrameView attribute*), 41
`FileExistsError`, 29
`FilenameParseError`, 29
`filenames` (*xanespy.txmstore.TXMStore attribute*), 48
`first_button_clicked` (*xanespy.qt_frame_view.QtFrameView attribute*), 41
`first_frame()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 43
`fit_linear_combinations()` (*xanespy.xanes_frameset.XanesFrameset method*), 58
`fit_parameters` (*xanespy.txmstore.TXMStore attribute*), 48
`fit_spectra()` (*in module xanespy.fitting*), 30
`fit_spectra()` (*xanespy.xanes_frameset.XanesFrameset method*),

58
fitting_param_names() (xanespy.xanes_frameset.XanesFrameset method), 59
foreach() (in module xanespy.utilities), 53
fork_data_group() (xanespy.txmstore.TXMStore method), 48
fork_data_group() (xanespy.xanes_frameset.XanesFrameset method), 59
forward_button_clicked (xanespy.qt_frame_view.QtFrameView attribute), 41
frame_cmap(xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
frame_controls() (xanespy.qt_frame_view.QtFrameView method), 41
frame_data_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
frame_data_cleared (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
frame_hover_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
frame_norm() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 43
frame_pixel (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
frame_shape() (xanespy.xanes_frameset.XanesFrameset method), 59
frame_slider_moved (xanespy.qt_frame_view.QtFrameView attribute), 41
frame_source() (xanespy.txmstore.TXMStore method), 48
frame_vrange_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
FrameAnimation (class in xanespy.qt_frame_view), 40
FrameChangeSource (class in xanespy.qt_frame_view), 40
FrameFileNotFound, 29
frames (xanespy.xanes_frameset.XanesFrameset attribute), 59
frameset_names() (xanespy.txmstore.TXMStore method), 48
FrameSourceError, 29

FramesPool (class in xanespy.xanes_math), 65

G

Gaussian (class in xanespy.fitting), 31
get_component() (in module xanespy.utilities), 53
get_dataset() (xanespy.txmstore.TXMStore method), 48
get_frames() (xanespy.txmstore.TXMStore method), 48
GroupKeyError, 30
guess_params() (xanespy.fitting.KCurve method), 32
gui_viewer() (xanespy.xanes_frameset.XanesFrameset method), 59

H

has_dataset() (xanespy.txmstore.TXMStore method), 49
has_representation() (xanespy.xanes_frameset.XanesFrameset method), 59
hdf_path() (xanespy.xanes_frameset.XanesFrameset method), 59
hdf_path_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 43
hdf_tree_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 44
HDFScopeError, 30
hide() (xanespy.qt_map_view.QtMapView method), 46
hist_ax (xanespy.qt_frame_view.QtFrameView attribute), 41
hist_cb (xanespy.qt_frame_view.QtFrameView attribute), 41
horizontal (xanespy.utilities.Pixel attribute), 53
hover_frame_pixel() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 44

I

image_data() (xanespy.xradia.XRMFile method), 72
image_dtype() (xanespy.xradia.XRMFile method), 72
image_shape() (xanespy.xradia.XRMFile method), 72
image_stack() (xanespy.xradia.TXRMFile method), 71
import_aps32idc_xanes_file() (in module xanespy.importers), 32
import_aps32idc_xanes_files() (in module xanespy.importers), 33
import_aps4idc_sxstm_files() (in module xanespy.importers), 33

import_aps8bm_xanes_dir() (in module *xanespy.importers*), 34
 import_aps8bm_xanes_file() (in module *xanespy.importers*), 34
 import_frameset() (in module *xanespy.importers*), 34
 import_nanosurveyor_frameset() (in module *xanespy.importers*), 35
 import_ssrl_xanes_dir() (in module *xanespy.importers*), 35
 import_stxm_frameset() (in module *xanespy.importers*), 36
 intensities (*xanespy.txmstore.TXMStore* attribute), 49
 intensity_mean (*xanespy.txmstore.TXMStore* attribute), 49
 is_kernel() (in module *xanespy.utilities*), 53
 is_valid() (*xanespy.xradia.XRMFile* method), 72
 iter_indices() (in module *xanespy.xanes_math*), 68

K

k_edge_jump() (in module *xanespy.xanes_math*), 68
 k_edge_mask() (in module *xanespy.xanes_math*), 68
 KCurve (class in *xanespy.fitting*), 32
 KEdge (class in *xanespy.edges*), 27
 keyboard_nav() (*xanespy.qt_map_view.QtMapView* method), 46

L

L3Curve (class in *xanespy.fitting*), 31
 l_edge_mask() (in module *xanespy.xanes_math*), 68
 label_particles() (*xanespy.xanes_frameset.XanesFrameset* method), 60
 last_button_clicked (*xanespy.qt_frame_view.QtFrameView* attribute), 41
 last_frame() (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
 latest_cmap (*xanespy.qt_map_view.QtMapView* attribute), 46
 latest_data_name (*xanespy.txmstore.TXMStore* attribute), 49
 latexify() (in module *xanespy.plots*), 37
 launch() (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
 LEdge (class in *xanespy.edges*), 27
 limits_applied (*xanespy.qt_map_view.QtMapView* attribute), 45, 46
 limits_reset (*xanespy.qt_map_view.QtMapView* attribute), 45, 46
 Line (class in *xanespy.fitting*), 31

line_spectra() (*xanespy.xanes_frameset.XanesFrameset* method), 60
 linear_combination_parameters (*xanespy.txmstore.TXMStore* attribute), 49
 linear_combination_residuals (*xanespy.txmstore.TXMStore* attribute), 49
 linear_combination_sources (*xanespy.txmstore.TXMStore* attribute), 49
 LinearCombination (class in *xanespy.fitting*), 31
 LMOMnKEdge (class in *xanespy.edges*), 27

M

magnification_correction() (in module *xanespy.importers*), 36
 map() (*xanespy.xanes_math.FramesPool* method), 65
 map_clicked (*xanespy.qt_map_view.QtMapView* attribute), 46
 map_cmap (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
 map_cursor_changed (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
 map_data (*xanespy.xanes_frameset.XanesFrameset* attribute), 60
 map_data_changed (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 42, 44
 map_data_cleared (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 42, 44
 map_hovered (*xanespy.qt_map_view.QtMapView* attribute), 46
 map_limits_changed (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
 map_moved (*xanespy.qt_map_view.QtMapView* attribute), 46
 map_names() (*xanespy.txmstore.TXMStore* method), 50
 map_norm() (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
 map_pixel_changed (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
 map_range (*xanespy.edges.CuKEdge* attribute), 26
 map_range (*xanespy.edges.Edge* attribute), 26, 27
 map_range (*xanespy.edges.FeKEdge* attribute), 27
 map_range (*xanespy.edges.NCACobaltKEdge* attribute), 28
 map_range (*xanespy.edges.NCACobaltLEdge* attribute), 28
 map_range (*xanespy.edges.NCANickelKEdge* attribute), 28

`map_range` (*xanespy.edges.NCANickelLEdge* attribute), 29
`map_spectrum_changed` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
`map_vmax_changed` (*xanespy.qt_map_view.QtMapView* attribute), 46
`map_vmin_changed` (*xanespy.qt_map_view.QtMapView* attribute), 46
`mask()` (*xanespy.edges.KEdge* method), 27
`mask()` (*xanespy.edges.LEdge* method), 27
`mean_frame()` (*xanespy.xanes_frameset.XanesFrameset* method), 60
`mean_spectrum_changed` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
`monitor_sector8()` (in module *xanespy.beamlines*), 24
`mouse_clicked_canvas()` (*xanespy.qt_map_view.QtMapView* method), 46
`mouse_in_canvas()` (*xanespy.qt_map_view.QtMapView* method), 46
`move_map_pixel()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
`move_slider()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`new_cmap_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`new_component_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`new_hdf_file_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`new_hdf_group_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`new_image_axes()` (in module *xanespy.plots*), 37
`new_timestep_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`new_vrange_requested` (*xanespy.qt_frame_view.QtFrameView* attribute), 41
`next_frame()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
`NMCNickelKEdge29` (class in *xanespy.edges*), 29
`NoParticleError`, 30
`normalize()` (*xanespy.xanes_frameset.XanesFrameset* method), 60
`num_energies` (*xanespy.xanes_frameset.XanesFrameset* attribute), 60
`num_frames` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 44
`num_images()` (*xanespy.xradia.XRMFile* method), 72
`num_timesteps` (*xanespy.xanes_frameset.XanesFrameset* attribute), 60

N

`name` (*xanespy.edges.CuKEdge* attribute), 26
`name` (*xanespy.edges.Edge* attribute), 26
`name` (*xanespy.edges.FeKEdge* attribute), 27
`name` (*xanespy.edges.LMOMnKEdge* attribute), 28
`name` (*xanespy.edges.NCACobaltKEdge* attribute), 28
`name` (*xanespy.edges.NCACobaltLEdge* attribute), 28
`name` (*xanespy.edges.NCANickelKEdge* attribute), 28
`name` (*xanespy.edges.NCANickelLEdge* attribute), 29
`name` (*xanespy.fitting.Curve* attribute), 31
`name` (*xanespy.fitting.Gaussian* attribute), 31
`name` (*xanespy.fitting.KCurve* attribute), 32
`name` (*xanespy.fitting.L3Curve* attribute), 32
`name` (*xanespy.fitting.LinearCombination* attribute), 31
`NCACobaltKEdge` (class in *xanespy.edges*), 28
`NCACobaltLEdge` (class in *xanespy.edges*), 28
`NCANickelKEdge` (class in *xanespy.edges*), 28
`NCANickelKEdge61` (class in *xanespy.edges*), 28
`NCANickelKEdge62` (class in *xanespy.edges*), 28
`NCANickelLEdge` (class in *xanespy.edges*), 28
`new_axes()` (in module *xanespy.plots*), 37

O

`ole_value()` (*xanespy.xradia.XRMFile* method), 72
`open_file()` (*xanespy.txmstore.TXMStore* method), 50
`open_hdf_file()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`optical_depth_mean` (*xanespy.txmstore.TXMStore* attribute), 50
`optical_depths` (*xanespy.txmstore.TXMStore* attribute), 50
`original_positions` (*xanespy.txmstore.TXMStore* attribute), 50

P

`parallel_map()` (in module *xanespy.utilities*), 53
`param_names` (*xanespy.fitting.Curve* attribute), 31

- param_names (*xanespy.fitting.Gaussian* attribute), 31
 param_names (*xanespy.fitting.KCurve* attribute), 32
 param_names (*xanespy.fitting.L3Curve* attribute), 32
 param_names (*xanespy.fitting.LinearCombination* attribute), 31
 parent_group() (*xanespy.txmstore.TXMStore* method), 50
 parent_name (*xanespy.xanes_frameset.XanesFrameset* attribute), 60
 particle_labels (*xanespy.txmstore.TXMStore* attribute), 50
 particle_labels() (in module *xanespy.xanes_math*), 69
 particle_regions() (*xanespy.xanes_frameset.XanesFrameset* method), 60
 particle_series() (*xanespy.xanes_frameset.XanesFrameset* method), 61
 Pixel (class in *xanespy.utilities*), 53
 pixel_size() (*xanespy.xanes_frameset.XanesFrameset* method), 61
 pixel_sizes (*xanespy.txmstore.TXMStore* attribute), 50
 pixel_to_xy() (in module *xanespy.utilities*), 53
 pixel_unit (*xanespy.txmstore.TXMStore* attribute), 51
 pixel_unit() (*xanespy.xanes_frameset.XanesFrameset* method), 61
 play_button_clicked (*xanespy.qt_frame_view.QtFrameView* attribute), 41
 play_frames() (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
 play_speed_requested (*xanespy.qt_frame_view.QtFrameView* attribute), 41
 plot_beamer_maps() (*xanespy.xanes_frameset.XanesFrameset* method), 61
 plot_beamer_spectra() (*xanespy.xanes_frameset.XanesFrameset* method), 61
 plot_composite_map() (in module *xanespy.plots*), 37
 plot_frame() (*xanespy.xanes_frameset.XanesFrameset* method), 62
 plot_histogram() (*xanespy.xanes_frameset.XanesFrameset* method), 62
 plot_histogram_data() (*xanespy.qt_map_view.QtMapView* method), 46
 plot_kedge_fit() (in module *xanespy.plots*), 37
 plot_line_scans() (*xanespy.xanes_frameset.XanesFrameset* method), 62
 plot_map() (*xanespy.xanes_frameset.XanesFrameset* method), 62
 plot_map_data() (*xanespy.qt_map_view.QtMapView* method), 46
 plot_map_pixel_spectra() (*xanespy.xanes_frameset.XanesFrameset* method), 62
 plot_mean_image() (*xanespy.xanes_frameset.XanesFrameset* method), 63
 plot_pixel_spectra() (in module *xanespy.plots*), 38
 plot_signal_map() (*xanespy.xanes_frameset.XanesFrameset* method), 63
 plot_signals() (*xanespy.xanes_frameset.XanesFrameset* method), 63
 plot_spectra() (in module *xanespy.plots*), 38
 plot_spectra_as_map() (in module *xanespy.plots*), 38
 plot_spectrum() (in module *xanespy.plots*), 38
 plot_spectrum() (*xanespy.qt_map_view.QtMapView* method), 46
 plot_spectrum() (*xanespy.xanes_frameset.XanesFrameset* method), 63
 plot_txm_histogram() (in module *xanespy.plots*), 39
 plot_txm_intermediates() (in module *xanespy.plots*), 39
 plot_txm_map() (in module *xanespy.plots*), 40
 plotting_controls() (*xanespy.qt_frame_view.QtFrameView* method), 41
 position (class in *xanespy.utilities*), 53
 position() (*xanespy.beamlines.Zoneplate* method), 24
 post_edge (*xanespy.edges.CuKEdge* attribute), 26
 post_edge (*xanespy.edges.Edge* attribute), 26, 27
 post_edge (*xanespy.edges.FeKEdge* attribute), 27
 post_edge (*xanespy.edges.NCACobaltKEdge* attribute), 28
 post_edge (*xanespy.edges.NCACobaltLEdge* attribute), 28
 post_edge (*xanespy.edges.NCANickelKEdge* attribute), 28

- post_edge (*xanespy.edges.NCANickelLEdge attribute*), 29
 pre_edge (*xanespy.edges.CuKEdge attribute*), 26
 pre_edge (*xanespy.edges.Edge attribute*), 26, 27
 pre_edge (*xanespy.edges.FeKEdge attribute*), 27
 pre_edge (*xanespy.edges.NCACobaltKEdge attribute*), 28
 pre_edge (*xanespy.edges.NCACobaltLEdge attribute*), 28
 pre_edge (*xanespy.edges.NCANickelKEdge attribute*), 28
 pre_edge (*xanespy.edges.NCANickelLEdge attribute*), 29
 pre_edge_fit (*xanespy.edges.Edge attribute*), 27
 prepare_p0 () (in module *xanespy.fitting*), 30
 prepare_ui () (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 44
 presenter_frame_changed (*xanespy.qt_frame_view.QtFrameView attribute*), 41
 previous_frame () (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 44
 print_ole () (*xanespy.xradia.XRMFile method*), 72
 process_events (*xanespy.qt_frameset_presenter.QtFramesetPresenter attribute*), 44
 prog () (in module *xanespy.utilities*), 53
- ## Q
- QtFramesetPresenter (class in *xanespy.qt_frameset_presenter*), 42
 QtFrameView (class in *xanespy.qt_frame_view*), 40
 QtMapView (class in *xanespy.qt_map_view*), 45
- ## R
- read_metadata () (in module *xanespy.importers*), 37
 redraw_canvas () (*xanespy.qt_map_view.QtMapView method*), 46
 redraw_crosshairs () (*xanespy.qt_map_view.QtMapView method*), 46
 references (*xanespy.txmstore.TXMStore attribute*), 51
 RefinementError, 30
 refresh_frames () (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 44
 regions (*xanespy.edges.CuKEdge attribute*), 26
 regions (*xanespy.edges.Edge attribute*), 26, 27
 regions (*xanespy.edges.FeKEdge attribute*), 27
 regions (*xanespy.edges.LMOMnKEdge attribute*), 28
 regions (*xanespy.edges.NCACobaltLEdge attribute*), 28
 regions (*xanespy.edges.NCANickelKEdge attribute*), 28
 regions (*xanespy.edges.NCANickelKEdge61 attribute*), 28
 regions (*xanespy.edges.NCANickelKEdge62 attribute*), 28
 regions (*xanespy.edges.NCANickelLEdge attribute*), 29
 regions (*xanespy.edges.NMCNickelKEdge29 attribute*), 29
 register_correlations () (in module *xanespy.xanes_math*), 69
 register_template () (in module *xanespy.xanes_math*), 69
 relative_positions (*xanespy.txmstore.TXMStore attribute*), 51
 remove_callback () (*xanespy.qt_frame_view.FrameChangeSource method*), 40
 remove_extra_spines () (in module *xanespy.plots*), 40
 replace_dataset () (*xanespy.txmstore.TXMStore method*), 51
 request_new_vrange () (*xanespy.qt_frame_view.QtFrameView method*), 41
 reset_frame_range () (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 44
 reset_map_range () (*xanespy.qt_frameset_presenter.QtFramesetPresenter method*), 44
 reset_vrange_requested (*xanespy.qt_frame_view.QtFrameView attribute*), 41
 rows (*xanespy.utilities.shape attribute*), 54
- ## S
- sample_position () (*xanespy.xradia.XRMFile method*), 72
 sector8_xanes_script () (in module *xanespy.beamlines*), 24
 segment_materials () (*xanespy.xanes_frameset.XanesFrameset method*), 63
 segments (*xanespy.txmstore.TXMStore attribute*), 51
 select_active_hdf_item () (*xanespy.qt_frame_view.QtFrameView method*), 41
 set_axes_color () (in module *xanespy.plots*), 40
 set_busy_mode () (*xanespy.qt_frame_view.QtFrameView method*), 41

`set_cmap_list()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`set_cmap_list()` (*xanespy.qt_map_view.QtMapView* method), 46
`set_component_list()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`set_component_list()` (*xanespy.qt_map_view.QtMapView* method), 46
`set_frame_vrange()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
`set_map_cursor()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 44
`set_map_limits()` (*xanespy.qt_map_view.QtMapView* method), 46
`set_map_pixel()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 45
`set_map_vmax()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 45
`set_map_vmin()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 45
`set_outside_ticks()` (in module *xanespy.plots*), 40
`set_play_speed()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 45
`set_slider_max()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`set_status_energy()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`set_status_index()` (*xanespy.qt_frame_view.QtFrameView* method), 41
`set_status_metadata()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`set_status_unit()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`set_timestep()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`set_timestep()` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* method), 45
`set_timestep_list()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`set_window_title()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`setup_ui()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`setup_ui()` (*xanespy.qt_map_view.QtMapView* method), 46
`shape` (class in *xanespy.utilities*), 54
`shell` (*xanespy.edges.CuKEdge* attribute), 26
`shell` (*xanespy.edges.FeKEdge* attribute), 27
`shell` (*xanespy.edges.KEdge* attribute), 27
`shell` (*xanespy.edges.LEdge* attribute), 27
`shell` (*xanespy.edges.NCACobaltKEdge* attribute), 28
`shell` (*xanespy.edges.NCANickelKEdge* attribute), 28
`show()` (*xanespy.qt_map_view.QtMapView* method), 46
`show_spectrum_fit` (*xanespy.qt_frameset_presenter.QtFramesetPresenter* attribute), 45
`show_status_message()` (*xanespy.qt_frame_view.QtFrameView* method), 42
`signal_map` (*xanespy.txmstore.TXMStore* attribute), 51
`signal_method` (*xanespy.txmstore.TXMStore* attribute), 51
`signal_weights` (*xanespy.txmstore.TXMStore* attribute), 52
`signals` (*xanespy.txmstore.TXMStore* attribute), 52
`spectra()` (*xanespy.xanes_frameset.XanesFrameset* method), 63
`spectrum()` (*xanespy.xanes_frameset.XanesFrameset* method), 64
`spectrum_ax` (*xanespy.qt_frame_view.QtFrameView* attribute), 42
`spectrum_fit_toggled` (*xanespy.qt_map_view.QtMapView* attribute), 46
`ssl6_xanes_script()` (in module *xanespy.beamlines*), 25
`stage_transformations()` (*xanespy.xanes_frameset.XanesFrameset* method), 64
`start()` (*xanespy.qt_frame_view.FrameChangeSource* method), 40
`starttime()` (*xanespy.xanes_frameset.XanesFrameset* method), 65
`starttime()` (*xanespy.xradia.XRMFile* method), 72
`starttimes()` (*xanespy.xradia.XRMFile* method), 72
`stop()` (*xanespy.qt_frame_view.FrameAnimation* method), 40
`stop()` (*xanespy.qt_frame_view.FrameChangeSource* method), 40

- method), 40
- store() (xanespy.xanes_frameset.XanesFrameset method), 65
- subtract_surroundings() (xanespy.xanes_frameset.XanesFrameset method), 65
- ## T
- timer (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 45
- timestamps (xanespy.txmstore.TXMStore attribute), 52
- timestamps() (xanespy.xanes_frameset.XanesFrameset method), 65
- timestep_list_changed (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 45
- timestep_names (xanespy.txmstore.TXMStore attribute), 52
- timestep_names (xanespy.xanes_frameset.XanesFrameset attribute), 65
- toggle_edge_mask() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- toggle_spectrum_fit() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- transform_images() (in module xanespy.xanes_math), 70
- transformation_matrices() (in module xanespy.xanes_math), 70
- TXMDataset (class in xanespy.txmstore), 47
- TXMStore (class in xanespy.txmstore), 47
- TXRMFile (class in xanespy.xradia), 71
- ## U
- ui (xanespy.qt_frame_view.QtFrameView attribute), 42
- ui (xanespy.qt_map_view.QtMapView attribute), 46
- um_per_pixel() (xanespy.xradia.XRMFile method), 72
- update_crosshair_labels() (xanespy.qt_map_view.QtMapView method), 46
- update_cursor_labels() (xanespy.qt_map_view.QtMapView method), 46
- update_figure_pixel() (xanespy.qt_frame_view.QtFrameView method), 42
- update_map_limits() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- update_maps() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- update_spectra() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- update_spectrum_ui() (xanespy.qt_map_view.QtMapView method), 47
- update_timestep_list() (xanespy.qt_frameset_presenter.QtFramesetPresenter method), 45
- use_edge_mask (xanespy.qt_frameset_presenter.QtFramesetPresenter attribute), 45
- ## V
- validate_parent_group() (xanespy.txmstore.TXMStore method), 52
- VERSION (xanespy.txmstore.TXMStore attribute), 47
- vertical (xanespy.utilities.Pixel attribute), 53
- ## W
- whiteline_fit (xanespy.txmstore.TXMStore attribute), 52
- whiteline_max (xanespy.txmstore.TXMStore attribute), 52
- window (xanespy.qt_frame_view.QtFrameView attribute), 42
- window (xanespy.qt_map_view.QtMapView attribute), 47
- write_scaninfo_header() (in module xanespy.beamlines), 25
- ## X
- x (xanespy.beamlines.DetectorPoint attribute), 23
- x (xanespy.beamlines.ZoneplatePoint attribute), 24
- x (xanespy.utilities.position attribute), 53
- x (xanespy.utilities.xycoord attribute), 54
- XanesFrameset (class in xanespy.xanes_frameset), 54
- XanesMathError, 30
- xanespy (module), 72
- xanespy.beamlines (module), 23
- xanespy.edges (module), 26
- xanespy.exceptions (module), 29
- xanespy.fitting (module), 30
- xanespy.importers (module), 32
- xanespy.plots (module), 37
- xanespy.qt_frame_view (module), 40
- xanespy.qt_frameset_presenter (module), 42
- xanespy.qt_map_view (module), 45
- xanespy.txmstore (module), 47
- xanespy.utilities (module), 53
- xanespy.xanes_frameset (module), 54
- xanespy.xanes_math (module), 65
- xanespy.xradia (module), 71
- XRMFile (class in xanespy.xradia), 71

`xy_to_pixel()` (in module *xanespy.utilities*), 54
`xycoord` (class in *xanespy.utilities*), 54

Y

`y` (*xanespy.beamlines.DetectorPoint* attribute), 23
`y` (*xanespy.beamlines.ZoneplatePoint* attribute), 24
`y` (*xanespy.utilities.position* attribute), 53
`y` (*xanespy.utilities.xycoord* attribute), 54

Z

`z` (*xanespy.beamlines.DetectorPoint* attribute), 23
`z` (*xanespy.beamlines.ZoneplatePoint* attribute), 24
`z` (*xanespy.utilities.position* attribute), 53
`Zoneplate` (class in *xanespy.beamlines*), 23
`ZoneplatePoint` (class in *xanespy.beamlines*), 24