spym

Feb 24, 2023

Contents

1 Installation 3

2 Usage 5

3 API Reference 7

4 Feedback 15

Python Module Index 17

Index 19

`spym` is a Python package for processing Scanning Probe Microscopy (SPM) data, which best integrates with xarray, providing an accessor extension to its *DataArrays* and *Datasets*.

CHAPTER 1

Installation

You can install spym in your system through pip:

```
$ pip install spym
```

1.1 Prerequisites

Spym extends and is best integrated with the xarray package, which is required for the installation. Nevertheless, many spym functions can be used directly on numpy arrays (see the Usage section for more information). For signal processing the scipy package is also required. In the case you need to load and save data from and to NeXus/HDF5 through the *spym.load()* function the nxarray package is needed.

CHAPTER 2

Usage

After installation you can import spym simply with:

```
import spym
```

Now the `spym` accessor will be available to xarray objects. For example to apply the plane() function to an xarray DataArray:

```
dr = xarray.DataArray()
dr.spym.plane()
```

2.1 Examples

The spym package is best integrated with xarray objects. Here is a typical usage for scanning tunneling microscopy data loaded into an xarray Dataset:

```
# Import the package
import spym

# Load the file and show the content
f = spym.load("/path/to/a/supported/file")
f

# Select the channel of interest (e.g. Topography_Forward) and show its content
tf = f.Topography_Forward
tf

# Align the rows
tf.spym.align()

# Make plane on the image
tf.spym.plane()
```

(continues on next page)

(continued from previous page)

```
# Fix the minimum to zero
tf.spym.fixzero()

# Plot the image
tf.spym.plot()
```

Many of the `spym` functions are also applicable directly to numpy arrays:

```
from spym.level import align

aligned, background = align(my_array, baseline='median')
```

The documentation of each `spym` method can be accessed with the `?` syntax:

```
dr.spym.plane?
```

See the API Reference section for a list of all the methods available.

2.2 Supported file formats

The `spym` package provides direct imports through the `spym.load()` function for a few file formats, at present:

- RHK SM4 *.sm4
- Omicron Scala *.par
- NeXus/HDF5 *.nxs (`nxarray` package is needed)

CHAPTER 3

API Reference

This page contains auto-generated API reference documentation[1].

3.1 spym

3.1.1 Subpackages

spym.io

Subpackages

spym.io.omicronscala

Package Contents

Functions

load(parFile)	This method load data and metadata associated to an Omicron SCALA .par file.
to_dataset(parFile, scaling=True)	This method load an Omicron SCALA .par file into an xarray Dataset.
to_nexus(parFile, filename=None, **kwargs)	This method convert an Omicron SCALA .par file into a NeXus file.

spym.io.omicronscala.**load**(*parFile*)

This method load data and metadata associated to an Omicron SCALA .par file.

Args: parFile: the name of the .par file to be loaded

[1] Created with sphinx-autoapi

Returns: a container for the channels in the .par file with their data and metadata

Examples: f = omicronscala.load('/path/to/file.par') # load the file

ch0 = f[0] # assign first channel ch0.label # returns channel name label ch0.data # returns channel data as a numpy array ch0.attrs # returns channel metadata as a dictionary

`spym.io.omicronscala.`**`to_dataset`**(*parFile*, *scaling=True*)

This method load an Omicron SCALA .par file into an xarray Dataset.

The xarray package is required.

Args: parFile: the name of the .par file to be loaded scaling: if True convert data to physical units (default),

if False keep data in ADC units

Returns: an xarray Dataset

Examples: ds = omicronscala.to_dataset('/path/to/file.par')

ds <xarray.Dataset>

ds.Z_Forward <xarray.DataArray>

`spym.io.omicronscala.`**`to_nexus`**(*parFile*, *filename=None*, ***kwargs*)

This method convert an Omicron SCALA .par file into a NeXus file.

The nxarray package is required.

Args: parFile: the name of the .par file to be converted filename: (optional) path of the NeXus file to be saved.

If not provided, a NeXus file is saved in the same folder of the .par file.

**kwargs: any optional argument accepted by nexus NXdata.save() method

Returns: nothing

Examples: omicronscala.to_nexus('/path/to/file.par')

spym.io.rhksm4

Package Contents

Functions

load(sm4file)	This method load data and metadata from an RHK .sm4 file.
to_dataset(sm4file, scaling=True)	This method load an RHK .sm4 file into an xarray Dataset.
to_nexus(sm4file, filename=None, **kwargs)	This method convert an RHK .sm4 file into a NeXus file.

`spym.io.rhksm4.`**`load`**(*sm4file*)

This method load data and metadata from an RHK .sm4 file.

Args: sm4file: the name of the .sm4 file to be loaded

Returns: a container for the pages in the .sm4 file with their data and metadata

Examples: f = rhksm4.load('/path/to/file.sm4') # load the file

p0 = f[0] # assign first page in the file p0.label # returns page label name p0.data # returns page data as a

numpy array p0.attrs # returns page metadata as a dictionary

`spym.io.rhksm4.to_dataset`(*sm4file*, *scaling=True*)

This method load an RHK .sm4 file into an xarray Dataset.

The xarray package is required.

Args: sm4file: the name of the .sm4 file to be loaded scaling: if True convert data to physical units (default),

if False keep data in ADC units

Returns: an xarray Dataset

Examples: ds = rhksm4.to_dataset('/path/to/file.sm4')

ds <xarray.Dataset>

ds.IDxxxxx <xarray.DataArray>

`spym.io.rhksm4.to_nexus`(*sm4file*, *filename=None*, ***kwargs*)

This method convert an RHK .sm4 file into a NeXus file.

The nxarray package is required.

Args: sm4file: the name of the .sm4 file to be converted filename: (optional) path of the NeXus file to be saved.

If not provided, a NeXus file is saved in the same folder of the .sm4 file.

**kwargs: any optional argument accepted by nexus NXdata.save() method

Returns: nothing

Examples: rhksm4.to_nexus('/path/to/file.sm4')

Submodules

spym.io.load

Module Contents

Functions

load(filename, scaling=True)	Import data from common SPM file formats.
convert(filename, folder=None)	Convert data from supported SPM file formats to NeXus/HDF5.

`spym.io.load.load`(*filename*, *scaling=True*)

Import data from common SPM file formats.

Currently supported file formats are:

- NeXus (.nx, .nxs). Package nxarray is needed.
- RHK (.sm4).
- Omicron Scala (.par).

Args: filename: path to the SPM file. scaling: if True convert data to physical units (default), if False keep raw data.

Returns: xarray Dataset with data and metadata.

`spym.io.load.convert`(*filename*, *folder=None*)
Convert data from supported SPM file formats to NeXus/HDF5.

Args: filename: path to the SPM file. folder: (optional) path for converted files. If not provided, converted files are placed in the same folder of the originals.

Returns: Nothing.

Package Contents

Functions

load	
convert(filename, folder=None)	Convert data from supported SPM file formats to NeXus/HDF5.

`spym.io.load`(*filename*, *scaling=True*)
Import data from common SPM file formats.

Currently supported file formats are:

- NeXus (.nx, .nxs). Package nxarray is needed.
- RHK (.sm4).
- Omicron Scala (.par).

Args: filename: path to the SPM file. scaling: if True convert data to physical units (default), if False keep raw data.

Returns: xarray Dataset with data and metadata.

`spym.io.convert`(*filename*, *folder=None*)
Convert data from supported SPM file formats to NeXus/HDF5.

Args: filename: path to the SPM file. folder: (optional) path for converted files. If not provided, converted files are placed in the same folder of the originals.

Returns: Nothing.

spym.plotting

Package Contents

Classes

Plotting	Plotting.

class `spym.plotting.Plotting`(*spym_instance*)
Plotting.

plot(*self*, *title=None*, *waterfall=False*, *waterfall_limit=15*, ***kwargs*)
Plot data with custom parameters using matplotlib.

Args: title: (optional) title of the figure (string). By default gives some basic information on the data

plotted. Pass an empty string to disable it. waterfall: (optional) boolean determining if plot spectrum data as waterfall (default is False). waterfall_limit: (optional) number of spectra above which spectrum data is plotted as image instead of waterfall (default is 15). **kwargs: any argument accepted by xarray.plot() function.

hvplot (*self*, *title=None*, ***kwargs*)
Plot data with custom parameters using hvplot.

Args: title: (optional) title of the figure (string). By default gives some basic information on the data plotted. Pass an empty string to disable it. **kwargs: any argument accepted by hvplot() function.

spym.process

Submodules

spym.process.filters

Module Contents

Classes

Filters	Filters.

Functions

gauss(image, size=3)	Apply Gaussian smoothing filter.
median(image, size=3)	Apply median smoothing filter.
mean(image, size=3)	Apply mean smoothing filter.
sharpen(image, size=3, alpha=30)	Apply a sharpening filter.
destripe(image, min_length=20, hard_threshold=0.4, soft_threshold=0.2, sign='positive', rel_threshold=None)	Find and remove scan stripes by averaging neighbourhood lines.

class spym.process.filters.**Filters** (*spym_instance*)
Filters.

gauss (*self*, ***kwargs*)
Apply Gaussian smoothing filter.

Args: size: size of the filter in pixels.

median (*self*, ***kwargs*)
Apply median smoothing filter.

Args: size: size of the filter in pixels.

mean (*self*, ***kwargs*)
Apply mean smoothing filter.

Args: size: size of the filter in pixels.

sharpen (*self*, ***kwargs*)
Apply a sharpening filter.

Args: size: size of the filter in pixels. alpha: weight.

destripe(*self*, ***kwargs*)

Find and remove scan stripes by averaging neighbourhood lines.

Args: min_length: only scars that are as long or longer than this value (in pixels) will be marked. hard_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect. soft_threshold: values differing at least this much do not form defects themselves, but they are attached to defects obtained from the hard threshold if they touch one. sign: whether mark stripes with positive values, negative values or both. rel_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect (in physical values). Overwrite hard_threshold.

Returns: destriped 2d array.

spym.process.filters.**gauss**(*image*, *size=3*)

Apply Gaussian smoothing filter.

Args: image: numpy array. size: size of the filter in pixels.

Returns: filtered numpy array.

spym.process.filters.**median**(*image*, *size=3*)

Apply median smoothing filter.

Args: image: numpy array. size: size of the filter in pixels.

Returns: filtered numpy array.

spym.process.filters.**mean**(*image*, *size=3*)

Apply mean smoothing filter.

Args: image: numpy array. size: size of the filter in pixels.

Returns: filtered numpy array.

spym.process.filters.**sharpen**(*image*, *size=3*, *alpha=30*)

Apply a sharpening filter.

Args: image: numpy array. size: size of the filter in pixels. alpha: weight.

Returns: filtered numpy array.

spym.process.filters.**destripe**(*image*, *min_length=20*, *hard_threshold=0.4*, *soft_threshold=0.2*, *sign='positive'*, *rel_threshold=None*)

Find and remove scan stripes by averaging neighbourhood lines.

Args: image: 2d numpy array. min_length: only scars that are as long or longer than this value (in pixels) will be marked. hard_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect. soft_threshold: values differing at least this much do not form defects themselves, but they are attached to defects obtained from the hard threshold if they touch one. sign: whether mark stripes with positive values, negative values or both. rel_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect (in physical values). Overwrite hard_threshold.

Returns: destriped 2d array.

spym.process.level

Module Contents

Classes

Level	Level.

Functions

fixzero(image, to_mean=False)	Add a constant to all the data to move the minimum (or the mean value) to zero.
plane(image)	Corrects for image tilting by subtraction of a plane.
align(image, baseline='mean', axis=1, poly_degree=2)	Align rows.

class spym.process.level.**Level**(*spym_instance*)

Level.

fixzero(*self*, ***kwargs*)

Add a constant to all the data to move the minimum (or the mean value) to zero.

Args: to_mean: bool, optional. If true move mean value to zero, if false move mimimum to zero (default).

plane(*self*, ***kwargs*)

Corrects for sample tilting by subtraction of a plane.

align(*self*, ***kwargs*)

Align rows.

Args: baseline: defines how baselines are estimated; 'mean' (default), 'median', 'poly'. axis: axis along wich calculate the baselines. poly_degree: polnomial degree if baseline='poly'.

spym.process.level.**fixzero**(*image*, *to_mean=False*)

Add a constant to all the data to move the minimum (or the mean value) to zero.

Args: image: numpy array. to_mean: bool, optional. If true move mean value to zero, if false move mimimum to zero (default).

Returns: numpy array.

spym.process.level.**plane**(*image*)

Corrects for image tilting by subtraction of a plane.

Args: image: 2d numpy array.

Returns: flattened image as 2d numpy array.

spym.process.level.**align**(*image*, *baseline='mean'*, *axis=1*, *poly_degree=2*)

Align rows.

Args: image: 2d numpy array. baseline: defines how baselines are estimated; 'mean' (default), 'median', 'poly'. axis: axis along wich calculate the baselines. poly_degree: polnomial degree if baseline='poly'.

Returns: corrected 2d numpy array.

Package Contents

Classes

Filters	Filters.
Level	Level.

class spym.process.**Filters**(*spym_instance*)

Filters.

gauss(*self*, ***kwargs*)

Apply Gaussian smoothing filter.

Args: size: size of the filter in pixels.

median(*self*, ***kwargs*)

Apply median smoothing filter.

Args: size: size of the filter in pixels.

mean(*self*, ***kwargs*)

Apply mean smoothing filter.

Args: size: size of the filter in pixels.

sharpen(*self*, ***kwargs*)

Apply a sharpening filter.

Args: size: size of the filter in pixels. alpha: weight.

destripe(*self*, ***kwargs*)

Find and remove scan stripes by averaging neighbourhood lines.

Args: min_length: only scars that are as long or longer than this value (in pixels) will be marked. hard_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect. soft_threshold: values differing at least this much do not form defects themselves, but they are attached to defects obtained from the hard threshold if they touch one. sign: whether mark stripes with positive values, negative values or both. rel_threshold: the minimum difference of the value from the neighbouring upper and lower lines to be considered a defect (in physical values). Overwrite hard_threshold.

Returns: destriped 2d array.

class spym.process.**Level**(*spym_instance*)

Level.

fixzero(*self*, ***kwargs*)

Add a constant to all the data to move the minimum (or the mean value) to zero.

Args: to_mean: bool, optional. If true move mean value to zero, if false move mimimum to zero (default).

plane(*self*, ***kwargs*)

Corrects for sample tilting by subtraction of a plane.

align(*self*, ***kwargs*)

Align rows.

Args: baseline: defines how baselines are estimated; 'mean' (default), 'median', 'poly'. axis: axis along wich calculate the baselines. poly_degree: polnomial degree if baseline='poly'.

spym is part of the reScipy project.

CHAPTER 4

Feedback

Please report any feedback, bugs, or feature requests by opening an issue on the issue tracker of the code repository. You should provide as much information as possible to reproduce the problem, and details of your desiderata.

Python Module Index

S

spym, 7
spym.io, 7
spym.io.load, 9
spym.io.omicronscala, 7
spym.io.rhksm4, 8
spym.plotting, 10
spym.process, 11
spym.process.filters, 11
spym.process.level, 12

Index

A

`align()` (*in module spym.process.level*), 13
`align()` (*spym.process.Level method*), 14
`align()` (*spym.process.level.Level method*), 13

C

`convert()` (*in module spym.io*), 10
`convert()` (*in module spym.io.load*), 9

D

`destripe()` (*in module spym.process.filters*), 12
`destripe()` (*spym.process.Filters method*), 14
`destripe()` (*spym.process.filters.Filters method*), 12

F

`Filters` (*class in spym.process*), 14
`Filters` (*class in spym.process.filters*), 11
`fixzero()` (*in module spym.process.level*), 13
`fixzero()` (*spym.process.Level method*), 14
`fixzero()` (*spym.process.level.Level method*), 13

G

`gauss()` (*in module spym.process.filters*), 12
`gauss()` (*spym.process.Filters method*), 14
`gauss()` (*spym.process.filters.Filters method*), 11

H

`hvplot()` (*spym.plotting.Plotting method*), 11

L

`Level` (*class in spym.process*), 14
`Level` (*class in spym.process.level*), 13
`load()` (*in module spym.io*), 10
`load()` (*in module spym.io.load*), 9
`load()` (*in module spym.io.omicronscala*), 7
`load()` (*in module spym.io.rhksm4*), 8

M

`mean()` (*in module spym.process.filters*), 12
`mean()` (*spym.process.Filters method*), 14
`mean()` (*spym.process.filters.Filters method*), 11
`median()` (*in module spym.process.filters*), 12
`median()` (*spym.process.Filters method*), 14
`median()` (*spym.process.filters.Filters method*), 11

P

`plane()` (*in module spym.process.level*), 13
`plane()` (*spym.process.Level method*), 14
`plane()` (*spym.process.level.Level method*), 13
`plot()` (*spym.plotting.Plotting method*), 10
`Plotting` (*class in spym.plotting*), 10

S

`sharpen()` (*in module spym.process.filters*), 12
`sharpen()` (*spym.process.Filters method*), 14
`sharpen()` (*spym.process.filters.Filters method*), 11
`spym` (*module*), 7
`spym.io` (*module*), 7
`spym.io.load` (*module*), 9
`spym.io.omicronscala` (*module*), 7
`spym.io.rhksm4` (*module*), 8
`spym.plotting` (*module*), 10
`spym.process` (*module*), 11
`spym.process.filters` (*module*), 11
`spym.process.level` (*module*), 12

T

`to_dataset()` (*in module spym.io.omicronscala*), 8
`to_dataset()` (*in module spym.io.rhksm4*), 9
`to_nexus()` (*in module spym.io.omicronscala*), 8
`to_nexus()` (*in module spym.io.rhksm4*), 9