XPCS_library

Fabio Brugnara

Apr 04, 2025

CONTENTS:

XPCS_library is a Python package for X-ray Photon Correlation Spectroscopy (XPCS) data analysis. It contains tools for data processing, analysis, and visualization. The library mainly consist of four modules:

1. XPCS_tools: This module contains functions for data processing and analysis.
2. ID10_tools: This module contains functions to load and work on the data from the ID10 beamline at the European Synchrotron Radiation Facility (ESRF).
3. PETRA3_tools: This module contains functions to load and work on the data from the PETRA III beamline at the Deutsches Elektronen-Synchrotron (DESY).
4. COSMICRAY_tools: This module contains functions to filter out unwanted signals from the E4M detector.

CHAPTER

ONE

XPCS_TOOLS

A python library for XPCS data analysis. Use in combination with the library *ID10_tools* or PETRA3_tools to load the data.

Author: Fabio Brugnara

XPCS_tools.**E2lambda**(*E*)

XPCS_tools.**Q2theta**(*Ei*, *Q*)

XPCS_tools.**decorelation_f**(*t*, *tau*, *beta*, *c*, *y0*)

XPCS_tools.**gen_Qmask**(*Ei*, *theta*, *Q*, *dq*, *Qmap_plot=False*)

Generate the Q masks for the given Q values at the working angle. The function also plot the Qmap for the given energy and angle (if Qmap_plot is True).

Parameters

- **Ei** (*float*) – Energy of the beam in keV
- **theta** (*float*) – Working angle in degrees
- **Q** (*float or list of floats*) – Q value(s) to mask in [1/A]
- **dq** (*float or list of floats*) – Q width(s) to mask in [1/A]
- **Qmap_plot** (*bool*) – If True, plot the Qmap for the given energy and angle

Returns

Qmask (*np.array or dict of np.array*) – Q mask(s) of the e4m detector

XPCS_tools.**gen_mask**(*e4m_data*, *itime*, *OF=None*, *e4m_mask=None*, *Qmask=None*, *mask_geom=None*, *Ith_high=None*, *Ith_low=None*, *Imaxth_high=None*, *Nfi=None*, *Nff=None*, *hist_plots=False*)

Generate a mask for the e4m detector from various options. The function plot the so-obtained mask, and also return some histograms to look at the results (if hist_plots is True).

Parameters

- **e4m_data** (*sparse.csc_matrix*) – Sparse matrix of the e4m detector data
- **itime** (*float*) – Integration time of the e4m detector
- **OF** (*np.array*) – Overflow mask of the e4m detector
- **e4m_mask** (*np.array*) – Mask of the e4m detector lines (slightly wider than the overflow lines, as pixels on the adges are not reliable)
- **Qmask** (*np.array*) – Q mask of the e4m detector

- **mask_geom** (*list of dicts*) – List of geometries to mask (in dictionary form). The supported objects are: - Circle: {'geom': 'Circle', 'Cx': x0, 'Cy': y0, 'r': r, 'inside': True/False} - Rectangle: {'geom': 'Rectangle', 'x0': x0, 'y0': y0, 'xl': xl, 'yl': yl, 'inside': True/False}

  Example: mask_geom = [{'geom': 'Circle', 'Cx': 100, 'Cy': 100, 'r': 10, 'inside': True}, {'geom': 'Rectangle', 'x0': 50, 'y0': 50, 'xl': 20, 'yl': 10, 'inside': False}]

- **Ith_high** (*float*) – Threshold (above) for the mean photon flux of the pixels
- **Ith_low** (*float*) – Threshold (below) for the mean photon flux of the pixels
- **Imaxth_high** (*float*) – Maximum number of counts per pixel treshold
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider
- **hist_plots** (*bool*) – If True, plot the histograms of the mean flux per pixel and maximum counts per pixel.

Returns

mask (*np.array*) – Mask of the e4m detector

XPCS_tools.**gen_plots4mask**(*e4m_data*, *itime*, *Ith_high=None*, *Ith_low=None*, *Imaxth_high=None*, *OF=None*, *e4m_mask=None*, *Qmask=None*, *mask_geom=None*, *Nfi=None*, *Nff=None*, *max_plots=False*, *wide_plots=False*)

Function that generates a number of different plots to create the mask! By default it generates the average flux per pixel map and histogram.

Parameters

- **e4m_data** (*sparse.csr_matrix*) – Sparse matrix of the e4m detector data
- **itime** (*float*) – Integration time of the e4m detector
- **Ith_high** (*float*) – Threshold (above) for the mean photon flux of the pixels [ph/s/px]
- **Ith_low** (*float*) – Threshold (below) for the mean photon flux of the pixels [ph/s/px]
- **Imaxth_high** (*float*) – Maximum number of counts per pixel treshold [ph/px]
- **OF** (*np.array*) – Overflow mask of the e4m detector
- **e4m_mask** (*np.array*) – Mask of the e4m detector lines (slightly wider than the overflow lines, as pixels on the adges are not reliable)
- **Qmask** (*np.array*) – Q mask of the e4m detector
- **mask_geom** (*list of dicts*) – List of geometries to mask.
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider
- **max_plots** (*bool*) – If True, plot the maximum counts per pixel map and histogram.
- **wide_plots** (*bool*) – If True, plot the wide histogram of the mean flux per pixel and maximum counts per pixel (if max_plots is True).

XPCS_tools.**get_G2t**(*e4m_data*, *mask=None*, *Nfi=None*, *Nff=None*, *Lbin=None*, *MKL_library=True*, *NumExpr_library=True*)

Compute the G2t matrix from the e4m, properly masked with the matrix mask.

Parameters

- **e4m_data** (*sparse.csr_matrix*) – Sparse matrix of the e4m detector data

- **mask** (*np.array*) – Mask of the e4m detector
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider
- **Lbin** (*int*) – Binning factor for the frames
- **MKL_library** (*boolean*) – If True, use the MKL library for the matrix multiplication
- **NumExpr_library** (*boolean*) – If True, use the NumExpr library for the normalization

Returns
G2t (*np.array*) – G2t matrix

XPCS_tools.**get_G2t_bybunch**(*e4m_data*, *Nbunch*, *mask=None*, *Nfi=None*, *Nff=None*, *Lbin=None*, *MKL_library=True*, *NumExpr_library=True*)

Compute the G2t matrix from the e4m, bunching the frames in Nbunch bunches, thus averaging the G2t matrix over the bunches.

Parameters

- **e4m_data** (*sparse.csc_matrix*) – Sparse matrix of the e4m detector data
- **Nbunch** (*int*) – Number of bunches to consider
- **mask** (*np.array*) – Mask of the e4m detector
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider
- **Lbin** (*int*) – Binning factor for the frames
- **MKL_library** (*boolean*) – If True, use the MKL library for the matrix multiplication
- **NumExpr_library** (*boolean*) – If True, use the NumExpr library for the normalization

Returns
G2t (*np.array*) – G2t matrix

XPCS_tools.**get_It**(*e4m_data*, *itime*, *mask=None*, *Nfi=None*, *Nff=None*, *Lbin=None*, *Nstep=None*)

Compute the average frame intensity [ph/px/s] vector from the e4m_data, properly masked with the mask.

Parameters

- **e4m_data** (*sparse.csr_matrix*) – Sparse matrix of the e4m detector data
- **itime** (*float*) – Integration time of the e4m detector
- **mask** (*np.array*) – Mask of the e4m detector
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider
- **Lbin** (*int*) – Binning factor for the frames
- **Nstep** (*int*) – Step for the frames

Returns

- **t_Idt** (*np.array*) – Time array for the It vector
- **It** (*np.array*) – It vector

XPCS_tools.**get_g2**(*dt*, *G2t*, *cython=True*)

Compute the g2 from the G2t matrix.

Parameters

- **dt** (*float*) – Time step between frames
- **G2t** (*np.array*) – G2t matrix
- **cython** (*boolean*) – If True, use the cython code to compute the g2

Returns

- **t** (*np.array*) – Time array
- **g2** (*np.array*) – g2 array

XPCS_tools.**get_g2_mt**(*dt*, *g2*)

Compute the multitau g2 from the g2 array.

Parameters

- **dt** (*float*) – Time step between frames
- **g2** (*np.array*) – g2 array

Returns

- **t_multit** (*np.array*) – Time array for the multitau g2
- **g2_multit** (*np.array*) – Multitau g2 array

XPCS_tools.**lambda2E**(*l*)

XPCS_tools.**plot_G2t**(*G2t*, *vmin*, *vmax*, *itime=None*, *t1=None*, *t2=None*, *x1=None*, *x2=None*, *sigma_filter=None*, *full=False*)

‘ Plot the G2t matrix.

Parameters

- **G2t** (*np.array*) – G2t matrix
- **vmin** (*float*) – Minimum value for the color scale
- **vmax** (*float*) – Maximum value for the color scale
- **itime** (*float*) – Integration time of the e4m detector
- **t1** (*float*) – First time to consider (in [s] if itime is provided, otherwise in [frames])
- **t2** (*float*) – Last time to consider (in [s] if itime is provided, otherwise in [frames])
- **x1** (*float*) – If provided, shift the x axis to the given initial value
- **x2** (*float*) – If provided, shift the x axis to the given final value
- **sigma_filter** (*float*) – Sigma for the Gaussian filter (in [frames])
- **full** (*boolean*) – If True, plot the full G2t matrix mirroring the lower part

XPCS_tools.**plot_XYprofile**(*e4m_data*, *itime*, *ax='Y'*, *mask=None*, *Nfi=None*, *Nff=None*)

Plot the X or Y profiles of the e4m detector.

Parameters

- **e4m_data** (*sparse.csc_matrix*) – Sparse matrix of the e4m detector data
- **itime** (*float*) – Integration time of the e4m detector

- **ax** (*str*) – Axis to plot ('X' or 'Y')
- **mask** (*np.array*) – Mask of the e4m detector
- **Nfi** (*int*) – First frame to consider
- **Nff** (*int*) – Last frame to consider

XPCS_tools.**set_beamline**(*beamline_toset: str*)

Set the beamline parameters for the XPCS data analysis. The function load the correct varaibles (Nx, Ny, Npx, lxp, lyp) from the beamline tools.

Parameters

beamline (*str*) – Beamline name ('PETRA3' or 'ID10')

XPCS_tools.**set_expvar**(*X0_toset: int*, *Y0_toset: int*, *L_toset: float*)

Set the experimental variables for the data analysis.

Parameters

- **X0** (*int*) – X0 position of the beam center in pixels
- **Y0** (*int*) – Y0 position of the beam center in pixels
- **L** (*float*) – Distance from the sample to the detector in meters

XPCS_tools.**theta2Q**(*Ei*, *theta*)

CHAPTER

TWO

ID10_TOOLS

A python library for loading and processing data from the ID10 beamline at the European Synchrotron Radiation Facility (ESRF). The library provides functions for loading and processing data from the Eiger 4M detector, including converting dense data to sparse format, loading scan information, and handling overflow values.

Author: Fabio Brugnara

ID10_tools.**convert_dense_e4m_v1**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*, *n_jobs=6*, *of_value=None*, *Nf4overflow=10*)

Convert the e4m data in the master file to a sparse array. The function generate an image (frame) of the overflow values selecting the pixel that are in overfllows in all the first Nf4overflow frames. This image, called OF, can be then used to mask the overflow values in the sparse array.

Parameters

- **raw_folder** (*str*) – the folder where the raw data is stored
- **file_name** (*str*) – the name of the file
- **Nscan** (*int*) – the scan number
- **Nf4overflow** (*int*) – the number of frames to use to generate the overflow image (default=10)

Returns

- **OF** (*np.array*) – the overflow image##################
- **sA** (*scipy.sparse.csr_array*) – the sparse array with the e4m data (shape: Nf x Npx)

ID10_tools.**get_Nbit_v1**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*)

Get the number of bits of the e4m data in the master file. The function loads the first image from the first file and check the maximum value. The maximum value is used to determine the number of bits.

Parameters

- **raw_folder** (*string*) – path to raw data folder
- **sample_name** (*string*) – name of the sample
- **Ndataset** (*int*) – number of the dataset
- **Nscan** (*int*) – number of the scan

Returns

Nbit (*int*) – number of bits of the e4m data

ID10_tools.**load_dense_e4m**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*, *n_jobs=6*, *tosparse=True*, *of_value=None*, *Nf4overflow=10*)

Load all the e4m data present in a scan. If tosparse=True (default) convert the dataframes to a sparse array. In older versions of the line ('v1') many overflow values are present in these frames, as they rapresent the lines on

the detector, and also burned pixels. To save mamory, we want to avoid saving these values within the sparse array, as they are likely to be the same in all frames. The function generate an image (frame) of the overflow values selecting the pixel that are in overfllows in all the first Nf4overflow frames. This image, called OF, can be then used to mask the overflow values in the sparse array.

Work in progress

1) directly look in the scan folder instead of the master file (not relayng on the master hdf5 file). Need for Nframesperfile.

2) add the possibility to load only a part of the data (Nfi, Nff).

Future perspectives

1) Nstep myght be usefull.

Parameters

- **raw_folder** (*string*) – the folder where the raw data is stored
- **file_name** (*string*) – the name of the file
- **Nscan** (*int*) – the scan number
- **n_jobs** (*int*) – number of parallel jobs to use (default=6)
- **tosparse** (*bool*) – if True return a sparse array, otherwise return a numpy array (default=True)
- **Nf4overflow** (*int*) – the number of frames to use to generate the overflow image (default=10)

Returns

- **OF** (*np.array*) – the overflow image (ONLY FOR OLDER ID10 VERSIONS ('v1')!!!)
- **sA** (*scipy.sparse.csr_array*) – the sparse array with the e4m data (shape: Nf x Npx)

ID10_tools.**load_pilatus**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*, *Nfi=None*, *Nff=None*, *Lbin=None*)

Load pilatus images from h5 file.

Work in progress 1) work with multiple files? 2) directly load files from the directory (not relayng on the master hdf5 file)

Parameters

- **raw_folder** (*string*) – path to raw data folder
- **sample_name** (*string*) – name of the sample
- **Ndataset** (*int*) – number of the dataset
- **Nscan** (*int*) – number of the scan
- **Nfi** (*int*) – number of the first image
- **Nff** (*int*) – number of the last image

Returns

pilatus (*dict*) – dictionary with pilatus images

ID10_tools.**load_scan**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*)

Load scan parameters from h5 file. Many try-except are used to avoid errors in the case of missing parameters. The function will try to load the parameters and if they are not present, it will skip them. Fill free to add other parameters to the scan dictionary, with the try-except method.

Parameters

- **raw_folder** (*string*) – path to raw data folder
- **sample_name** (*string*) – name of the sample
- **Ndataset** (*int*) – number of the dataset
- **Nscan** (*int*) – number of the scan

Returns

scan (*dict*) – dictionary with scan parameters

ID10_tools.**load_sparse_e4m**(*raw_folder*, *sample_name*, *Ndataset*, *Nscan*, *Nfi=None*, *Nff=None*, *n_jobs=10*)

Load the sparse array and the overflow image from the correct e4m raw_data folder. This function works differently depending on the version of the ID10 line used. In the older version ('v1') the data should be first converted into the sparse format with the function ID10_tools.convert_dense_e4m. In the new version ('v2') the data is already saved in a sparse format at the line.

Future perspectives 1) implement Nstep to load only a part of the data

Parameters

- **raw_folder** (*string*) – the folder where the raw data is stored
- **sample_name** (*string*) – the name of the sample
- **Ndataset** (*int:*) – the number of the dataset
- **Nscan** (*int*) – the scan number
- **Nfi** (*int*) – the first frame to load (ONLY FOR THE V2 VERSION!) (default=None)
- **Nff** (*int*) – the last frame to load (ONLY FOR THE V2 VERSION!) (default=None)
- **n_jobs** (*int*) – number of parallel jobs to use (default=10)

Returns

- **OF** (*np.array*) – the overflow image (ONLY FOR OLDER ID10 VERSIONS ('v1')!!!)
- **sA** (*scipy.sparse.csr_array*) – the sparse array with the e4m data (shape: Nf x Npx)

ID10_tools.**save_sparse_e4m_v1**(*OF*, *sA*, *raw_folder*, *sample_name*, *Ndataset*, *Nscan*)

Save the sparse array and the overflow image in the correct e4m raw_data folder. This function is usefull only for the older version of the ID10 line ('v1'). In the new version ('v2') the data is already saved in a sparse format.

Future perspectives 1) save sparse array in multiple files to load them faster in parallel

Parameters

- **OF** (*np.array*) – the overflow image
- **sA** (*scipy.sparse.csr_array*) – the sparse array with the e4m data (shape: Nf x Npx)
- **raw_folder** (*str*) – the folder where the raw data is stored
- **sample_name** (*str*) – the name of the sample
- **Ndataset** (*int*) – the number of the dataset
- **Nscan** (*int*) – the scan number

ID10_tools.**set_version**(*v*)

This function set some parameters for using a version of the ID10 line. We can add here as many versions as we want. The version v1 is the one used in the ID10 line before 2023. The v2 version is the one used in the ID10 line after 2023. The function set the parameters for the version selected.

Parameters

v (*string*) – Version of the ID10 line ('v1', 'v2', or new others ...)

CHAPTER

THREE

COSMICRAY_TOOLS

This module contains functions for filtering cosmic rays and gamma rays from E4M data.

Author: Fabio Brugnara

COSMICRAY_tools.**cosmic_filter**(*e4m_data*, *Dpx*, *counts_th*, *mask=None*, *itime=None*, *Nfi=None*, *Nff=None*, *Lbin=None*, *mask_plot=False*, *hist_plot=False*, *Nsigma=10*, *MKL_library=True*)

Cosmic ray filter for E4M data.

Parameters

- **e4m_data** (*sparse.sparray*) – E4M data to be filtered.
- **Dpx** (*int*) – Size of the kernel in pixels.
- **counts_th** (*int*) – Threshold for cosmic ray signal.
- **mask** (*sparse.sparray, optional*) – Mask to be applied to the data. Default is None.
- **itime** (*float, optional*) – Integration time in seconds. Default is None.
- **Nfi** (*int, optional*) – First frame to be loaded. Default is None.
- **Nff** (*int, optional*) – Last frame to be loaded. Default is None.
- **Lbin** (*int, optional*) – Binning factor. Default is None.
- **mask_plot** (*bool, optional*) – If True, plot the mask. Default is False.
- **hist_plot** (*bool, optional*) – If True, plot the histogram. Default is False.
- **Nsigma** (*int, optional*) – Number of standard deviations for the histogram. Default is 10.
- **MKL_library** (*bool, optional*) – If True, use MKL library for matrix multiplication. Default is True.

Returns

- **CR** (*sparse.sparray*) – Cosmic ray mask.
- **Itp** (*sparse.sparray*) – Filtered E4M data.

COSMICRAY_tools.**fast_gamma_filter**(*e4m_data*, *Imaxth_high*, *mask=None*, *info=False*, *itime=None*)

Fast gamma ray filter for E4M data.

Notes

Use mask=None and info=None drammatically improves the speed of the function.

Parameters

- **e4m_data** (*sparse.sparray*) – E4M data to be filtered.
- **Imaxth_high** (*float*) – Threshold for gamma ray signal.
- **mask** (*sparse.sparray, optional*) – Mask to be applied to the data. Default is None.
- **info** (*bool, optional*) – If True, print information about the gamma ray signal. Default is False.
- **itime** (*float, optional*) – Integration time in seconds. Default is None.

Returns

e4m_data (*sparse.sparray*) – Filtered E4M data.

COSMICRAY_tools.**set_beamline**(*beamline_toset*)

Set the beamline parameters for the data analysis.

Parameters

beamline_toset (*str*) – Beamline name

PYTHON MODULE INDEX

INDEX