PyHyperScattering

Release 0+unknown

Peter Beaucage

Sep 11, 2023

CONTENTS:

1 Introduction **3**
- 1.1 loading . . . 3
- 1.2 integration . . . 3
- 1.3 utility modules . . . 3

2 Loading: getting your data into PyHyperScattering **5**

3 Integration: raw intensity to I(q) **7**

4 Learning to Fly **9**

5 Utility Modules **11**
- 5.1 RSoXS . . . 11
- 5.2 Fitting . . . 11

6 API Documentation **13**
- 6.1 PyHyperScattering package . . . 13
  - 6.1.1 Submodules . . . 13
  - 6.1.2 PyHyperScattering.load module . . . 13
    - 6.1.2.1 PyHyperScattering.load.FileLoader module . . . 13
    - 6.1.2.2 PyHyperScattering.load.ALS11012RSoXSLoader module . . . 14
    - 6.1.2.3 PyHyperScattering.load.SST1RSoXSLoader module . . . 15
    - 6.1.2.4 PyHyperScattering.load.SST1RSoXSDB module . . . 16
    - 6.1.2.5 PyHyperScattering.load.cyrsoxsLoader module . . . 19
  - 6.1.3 PyHyperScattering.integrate module . . . 20
    - 6.1.3.1 PyHyperScattering.integrate.PFEnergySeriesIntegrator module . . . 20
    - 6.1.3.2 PyHyperScattering.integrate.PFGeneralIntegrator module . . . 21
  - 6.1.4 PyHyperScattering.util module . . . 23
    - 6.1.4.1 PyHyperScattering.util.Fitting module . . . 23
    - 6.1.4.2 PyHyperScattering.util.HDR module . . . 23
    - 6.1.4.3 PyHyperScattering.util.IntegrationUtils module . . . 23
    - 6.1.4.4 PyHyperScattering.util.FileIO module . . . 23
    - 6.1.4.5 PyHyperScattering.util.RSoXS module . . . 23
  - 6.1.5 Module contents . . . 23

7 Indices and tables **25**

Python Module Index **27**

Index **29**

To play with PyHyperScattering's tutorial notebooks interactively, use Binder: .. image:: mybinder.org/badge_logo.svg

target
mybinder.org/v2/gh/usnistgov/PyHyperScattering/HEAD

At this stage, the tutorials may be a better overview than this (incomplete, but growing) documentation.

CHAPTER

ONE

INTRODUCTION

PyHyperScattering aims to make working with hyperspectral x-ray and neutron scattering data easy, to make programs that work with such data a combination of simple, logical commands with minimal 'cruft'. In the era of modern computing, there is no reason you should have to think about for loops and how you're storing different intermediate data products - you should be able to go immediately from raw data to an analysis with clear commands, punch down to the data you need for your science quickly. The goal is for these tools to make the mechanics of hyperspectral scattering easier and in so doing, more reproducible, explainable, and robust.

It grew out of the NIST RSoXS program, but aims to be broadly applicable. If it's scattering data, you should be able to load it, reduce it, and slice it.

The tools PyHyper provides are basically divided into three categories:

1.1 loading

get your data from a source - files on disk or a network connection to a DataBroker or whatever - into a standardized structure we call a raw xarray. The metadata should be loaded and converted to a standardized set of terms, intensity corrections applied, and the frames stacked along whatever dimensions you want.

1.2 integration

convert your data from pixel space or qx/qy space to chi-q space - perfect for slicing. generally tools here are built on pyFAI, though some variants also use warp_polar from numpy.

1.3 utility modules

these are pre-canned routines for common analyses, such as RSoXS anisotropic ratio or peak/background fitting. the intent here is that the barrier to building your own code is low, and contributions in module space are especially welcomed and encouraged.

Have fun!

CHAPTER

TWO

LOADING: GETTING YOUR DATA INTO PYHYPERSCATTERING

CHAPTER

THREE

INTEGRATION: RAW INTENSITY TO I(Q)

CHAPTER

FOUR

LEARNING TO FLY

From this point, you're in a place where the next steps largely depend on the analysis you want to do with your data.

The tools and language are that of the xarray package, so the main goal of this section is to teach you just enough xarray to have a vocabulary for converting scientific lines of thought into commands and visualize/dissect the results.

The most important command, by far, is select or .sel

CHAPTER

FIVE

UTILITY MODULES

Despite the power of writing your own analyses, it seems likely that many of the analyses you'll want to do won't actually be that novel.

To that end, we furnish two modules of 'canned' approaches: RSoXS and fitting. User contributions are encouraged in this area.

5.1 RSoXS

5.2 Fitting

CHAPTER

SIX

API DOCUMENTATION

6.1 PyHyperScattering package

6.1.1 Submodules

6.1.2 PyHyperScattering.load module

6.1.2.1 PyHyperScattering.load.FileLoader module

class PyHyperScattering.load.**FileLoader**

Bases: `object`

Abstract class defining a generic scattering file loader. Input is a (or multiple) filename/s and output is a xarray I(pix_x,pix_y,dims,coords) where dims and coords are loaded by user request.

Difference: all coords are dims but not all dims are coords. Dims can also be auto-hinted using the following standard names: energy,exposure,pos_x,pos_y,pos_z,theta.

Individual loaders can try searching metadata for other dim names but this is not guaranteed. Coords can be used to provide a list of values for a dimension when that dimension cannot be hinted, e.g. where vals come from external data.

file_ext = ''

loadFileSeries(*basepath*, *dims*, *coords={}*, *file_filter=None*, *file_filter_regex=None*, *file_skip=None*, *md_filter={}*, *quiet=True*, *output_qxy=False*, *dest_qx=None*, *dest_qy=None*, *output_raw=False*, *image_slice=None*)

Load a series into a single xarray.

Parameters

- **basepath** (*str or Path*) – path to the directory to load
- **dims** (*list*) – dimensions of the resulting xarray, as list of str
- **coords** (*dict*) – dictionary of any dims that are *not* present in metadata
- **file_filter** (*str*) – string that must be in each file name
- **file_filer_regex** (*str*) – regex string that must match in each file name
- **file_skip** (*str*) – string that, if present in file name, means file should be skipped.
- **md_filter** (*dict*) – dict of *required* metadata values; points without these metadata values will be dropped

- **md_filter_regex** (*dict*) – dict of *required* metadata regex; points without these metadata values will be dropped
- **quiet** (*bool*) – skip printing most intermediate output if true.
- **output_qxy** (*bool*) – output a qx/qy stack rather than a pix_x/pix_y stack. This is a lossy operation, the array will be remeshed. Not recommended.
- **output_raw** (*bool*) – Do not apply pixel or q coordinates to the final stack.
- **dest_qx** (*array-like or None*) – set of qx points that you would like the final stack to have. If None, will take the middle image and remesh to that.
- **dest_qy** (*array-like or None*) – set of qy points that you would like the final stack to have. If None, will take the middle image and remesh to that.
- **image_slice** (*tuple of slices*) – If provided, all images will be reduced according to these slice objects

loadSingleImage(*filepath*, *coords=None*, *return_q=None*, ***kwargs*)

md_loading_is_quick = False

peekAtMd(*filepath*)

6.1.2.2 PyHyperScattering.load.ALS11012RSoXSLoader module

class PyHyperScattering.load.**ALS11012RSoXSLoader**(*corr_mode=None*, *user_corr_func=None*, *dark_pedestal=0*, *exposure_offset=0.002*, *dark_subtract=False*, *data_collected_after_mar2021=False*, *constant_md={}*)

Bases: *FileLoader*

Loader for FITS files from the ALS 11.0.1.2 RSoXS instrument

Additional requirement: astropy, for FITS file loader

Usage is mainly via the inherited function integrateImageStack from FileLoader

file_ext = '(.*?).fits'

loadDarks(*basepath*, *dark_base_name*)

Load a series of dark images as a function of exposure time, to be subtracted from subsequently-loaded data.

Parameters

- **basepath** (*str or Path*) – path to load images from
- **dark_base_name** (*str*) – str that must be in file for file to be a dark

loadSampleSpecificDarks(*basepath*, *file_filter=''*, *file_skip='donotskip'*, *md_filter={}*)

load darks matching a specific sample metadata

Used, e.g., to load darks taken at a time of measurement in order to improve the connection between the dark and sample data.

Parameters

- **basepath** (*str*) – path to load darks from
- **file_filter** (*str*) – string that must be in each file name

- **file_skip** (*str*) – string that, if in file name, means file should be skipped.
- **md_filter** (*dict*) – dict of required metadata values. this will be appended with dark images only, no need to put that here.

loadSingleImage(*filepath*, *coords=None*, *return_q=False*, ***kwargs*)

THIS IS A HELPER FUNCTION, mostly - should not be called directly unless you know what you are doing

Load a single image from filepath and return a single-image, raw xarray, performing dark subtraction if so configured.

md_loading_is_quick = True

normalizeMetadata(*headerdict*)

convert the local metadata terms in headerdict to standard nomenclature

Parameters
headerdict (*dict*) – the header returned by the file loader

peekAtMd(*file*)

load the header/metadata without opening the corresponding image

Parameters
file (*str*) – fits file from which to load metadata

6.1.2.3 PyHyperScattering.load.SST1RSoXSLoader module

class PyHyperScattering.load.**SST1RSoXSLoader**(*corr_mode=None*, *user_corr_func=None*, *dark_pedestal=100*, *exposure_offset=0*, *constant_md={}*)

Bases: *FileLoader*

Loader for TIFF files from NSLS-II SST1 RSoXS instrument

file_ext = '(.*?)primary(.*?).tiff'

loadMd(*filepath*)

loadSingleImage(*filepath*, *coords=None*, *return_q=False*, *image_slice=None*, *use_cached_md=False*, ***kwargs*)

HELPER FUNCTION that loads a single image and returns an xarray with either pix_x / pix_y dimensions (if return_q == False) or qx / qy (if return_q == True)

Parameters

- **filepath** (*Pathlib.path*) – path of the file to load
- **coords** (*dict-like*) – coordinate values to inject into the metadata
- **return_q** (*bool*) – return qx / qy coords. If false, returns pixel coords.

md_loading_is_quick = True

peekAtMd(*filepath*)

pix_size_1 = 0.06

pix_size_2 = 0.06

read_baseline(*baseline_csv*)

read_json(*jsonfile*)

read_primary(*primary_csv*, *seq_num*, *cwd*)

read_shutter_toggle(*shutter_csv*)

6.1.2.4 PyHyperScattering.load.SST1RSoXSDB module

class PyHyperScattering.load.**SST1RSoXSDB**(*corr_mode=None*, *user_corr_fun=None*, *dark_subtract=True*, *dark_pedestal=0*, *exposure_offset=0*, *catalog=None*, *catalog_kwargs={}*, *use_precise_positions=False*, *use_chunked_loading=False*)

Bases: object

Loader for bluesky run xarrays form NSLS-II SST1 RSoXS instrument

background()

do_list_append(***kwargs*)

file_ext = ''

loadMd(*run*)

return a dict of metadata entries from the databroker run xarray

loadMonitors(*entry*, *integrate_onto_images: bool = True*, *useShutterThinning: bool = True*, *n_thinning_iters: int = 5*)

Load the monitor streams for entry.

Creates a dataset containing all monitor streams (e.g., Mesh Current, Shutter Timing, etc.) as data variables mapped against time. Optionally, all streams can be indexed against the primary measurement time for the images using integrate_onto_images. Whether or not time integration attempts to account for shutter opening/closing is controlled by useShutterThinning. Warning: for exposure times < 0.5 seconds at SST (as of 9 Feb 2023), useShutterThinning=True may excessively cull data points.

Parameters

- **entry** (*databroker.client.BlueskyRun*) – Bluesky Document containing run information. ex: phs.load.SST1RSoXSDB.c[scanID] or databroker.client.CatalogOfBlueskyRuns[scanID]
- **integrate_onto_images** (*bool, optional*) – whether or not to map timepoints to the image measurement times (as held by the 'primary' stream), by default True Presently bins are averaged between measurements intervals.
- **useShutterThinning** (*bool, optional*) – Whether or not to attempt to thin (filter) the raw time streams to remove data collected during shutter opening/closing, by default False As of 9 Feb 2023 at NSLS2 SST1, using useShutterThinning= True for exposure times of < 0.5s is not recommended because the shutter data is unreliable and too many points will be culled
- **n_thinning_iters** (*int, optional*) – how many iterations of thinning to perform, by default 5 If the data is becoming too sparse, try fewer iterations

Returns

xarray dataset containing all monitor streams as data variables mapped against the dimension "time"

Return type
xr.Dataset

loadRun(*run*, *dims=None*, *coords={}*, *return_dataset=False*, *useMonitorShutterThinning=True*)

Loads a run entry from a catalog result into a raw xarray.

Parameters

- **run** (*DataBroker result, int of a scan id, list of scan ids, list of DataBroker runs*) – a single run from BlueSky
- **dims** (*list*) – list of dimensions you'd like in the resulting xarray. See list of allowed dimensions in documentation. If not set or None, tries to auto-hint the dims from the RSoXS plan_name.
- **CHANGE** – List of dimensions you'd like. If not set, will set all possibilities as dimensions (x, y, theta, energy, polarization)
- **coords** (*dict*) – user-supplied dimensions, see syntax examples in documentation.
- **return_dataset** (*bool,default False*) – return both the data and the monitors as a xr.dataset. If false (default), just returns the data.

Returns
raw xarray containing your scan in PyHyper-compliant format

Return type
raw (xarray)

loadSeries(*run_list*, *meta_dim*, *loadrun_kwargs={}*)

Loads a series of runs into a single xarray object, stacking along meta_dim.

Useful for a set of samples, or a set of polarizations, etc., taken in different scans.

Parameters

- **run_list** (*list*) – list of scan ids to load
- **meta_dim** (*str*) – dimension to stack along. must be a valid attribute/metadata value, such as polarization or sample_name

Returns
xarray.Dataset with all scans stacked

Return type
raw

loadSingleImage(*filepath*, *coords=None*, *return_q=False*, ***kwargs*)

DO NOT USE

This function is preserved as reference for the qx/qy loading conventions.

NOT FOR ACTIVE USE. DOES NOT WORK.

md_loading_is_quick = True

md_lookup = {'energy': 'en_energy_setpoint', 'exposure': 'RSoXS Shutter Opening Time (ms)', 'polarization': 'en_polarization_setpoint', 'sam_th': 'RSoXS Sample Rotation', 'sam_x': 'RSoXS Sample Outboard-Inboard', 'sam_y': 'RSoXS Sample Up-Down', 'sam_z': 'RSoXS Sample Downstream-Upstream'}

md_secondary_lookup = {'energy': 'en_monoen_setpoint'}

peekAtMd(*run*)

pix_size_1 = 0.06

pix_size_2 = 0.06

runSearch(***kwargs*)

Search the catalog using given commands.

Parameters
****kwargs** – passed through to the RawMongo search method of the catalog.

Returns
a catalog result object

Return type
result (obj)

summarize_run(*outputType: str = 'default'*, *cycle: str = None*, *proposal: str = None*, *saf: str = None*, *user: str = None*, *institution: str = None*, *project: str = None*, *sample: str = None*, *sampleID: str = None*, *plan: str = None*, *userOutputs: list = []*, *debugWarnings: bool = False*, ***kwargs*) → DataFrame

Search the Bluesky catalog for scans matching all provided keywords and return metadata as a dataframe.

Matches are made based on the values in the top level of the 'start' dict within the metadata of each entry in the Bluesky Catalog (databroker.client.CatalogOfBlueskyRuns). Based on the search arguments provided, a pandas dataframe is returned where rows correspond to catalog entries (scans) and columns contain metadata. Several presets are provided for choosing which columns are generated, along with an interface for user-provided search arguments and additional metadata. Fails gracefully on bad user input/ changes to underlying metadata scheme.

Ex1: All of the carbon,fluorine,or oxygen scans for a single sample series in the most recent cycle:
bsCatalogReduced4 = db_loader.summarize_run(sample="**bBP_**", institution="NIST", cycle = "2022-2", plan="carbon|fluorine|oxygen")

Ex2: Just all of the scan Ids for a particular sample:
bsCatalogReduced4 = db_loader.summarize_run(sample="BBP_PFP09A", outputType='scans')

Ex3: Complex Search with custom parameters
bsCatalogReduced3 = db_loader.summarize_run(['angle', '-1.6', 'numeric'], outputType='all',sample="**BBP_**", cycle = "2022-2", institution="NIST",plan="carbon", userOutputs = [["Exposure Multiplier", "exptime", r'catalog.start'], ["Stop Time","time",r'catalog.stop']])

Parameters

- **outputType** (*str, optional*) – modulates the content of output columns in the returned dataframe 'default' returns scan_id, start time, cycle, institution, project, sample_name, sample_id, plan name, detector, polarization, exit_status, and num_images 'scans' returns only the scan_ids (1-column dataframe) 'ext_msmt' returns default columns AND bar_spot, sample_rotation 'ext_bio' returns default columns AND uid, saf, user_name 'all' is equivalent to 'default' and all other additive choices
- **cycle** (*str, optional*) – NSLS2 beamtime cycle, regex search e.g., "2022" matches "2022-2", "2022-1"
- **proposal** (*str, optional*) – NSLS2 PASS proposal ID, case-insensitive, exact match, e.g., "GU-310176"
- **saf** (*str, optional*) – Safety Approval Form (SAF) number, exact match, e.g., "309441"

- **user** (*str, optional*) – User name, case-insensitive, regex search e.g., “eliot” matches “Eliot”, “Eliot Gann”
- **institution** (*str, optional*) – Research Institution, case-insensitive, exact match, e.g., “NIST”
- **project** (*str, optional*) – Project code, case-insensitive, regex search, e.g., “liquid” matches “Liquids”, “Liquid-RSoXS”
- **sample** (*str, optional*) – Sample name, case-insensitive, regex search, e.g., “BBP_” matches “BBP_PF902A”
- **sampleID** (*str, optional*) – Sample ID, case-insensitive, regex search, e.g., “BBP_” matches “BBP_PF902A”
- **plan** (*str, optional*) – Measurement Plan, case-insensitive, regex search, e.g., “Full” matches “full_carbon_scan_nd”, “full_fluorine_scan_nd” e.g., “carbon|oxygen|fluorine” matches carbon OR oxygen OR fluorine scans
- ****kwargs** – Additional search terms can be provided as keyword args and will further filter the catalog Valid input follows metadataLabel=’searchTerm’ or metadataLavel = [‘searchTerm’,’matchType’]. Metadata labels must match an entry in the ‘start’ dictionary of the catalog. Supported match types are combinations of ‘case-insensitive’, ‘case-sensitive’, and ‘exact’ OR ‘numeric’. Default behavior is to do a case-sensitive regex match. For metadata labels that are not valid python names, create the kwarg dict before passing into the function (see example 3). Additional search terms will appear in the output data columns. Ex1: passing in cycle=’2022’ would match ‘cycle’=’2022-2’ AND ‘cycle=’2022-1’ Ex2: passing in grazing=[0,’numeric’] would match grazing==0 Ex3: create kwargs first, then pass it into the function.

  kwargs = {‘2weird metadata label’: “Bob”, ‘grazing’: 0, ‘angle’:-1.6}
  db_loader.summarize_run(sample=”BBP_PFP09A”, outputType=’scans’, **kwargs)

- **userOutputs** (*list of lists, optional*) – Additional metadata to be added to output can be specified as a list of lists. Each sub-list specifies a metadata field as a 3 element list of format: [Output column title (str), Metadata label (str), Metadata Source (raw str)], Valid options for the Metadata Source are any of [r’catalog.start’, r’catalog.start[“plan_args”], r’catalog.stop’, r’catalog.stop[“num_events”]’] e.g., userOutputs = [[“Exposure Multiplier”,”exptime”, r’catalog.start’], [“Stop Time”,”time”,r’catalog.stop’]]
- **debugWarnings** (*bool, optional*) – if True, raises a warning with debugging information whenever a key can’t be found.

Returns

Pandas dataframe containing the results of the search, or an empty dataframe if the search fails

6.1.2.5 PyHyperScattering.load.cyrsoxsLoader module

class PyHyperScattering.load.**cyrsoxsLoader**(*eager_load=False*, *profile_time=True*, *use_chunked_loading=False*)

Bases: `object`

Loader for cyrsoxs simulation files

This code is modified from Dean Delongchamp.

file_ext = 'config.txt'

loadDirectory(*directory*, *method=None*, ***kwargs*)

loadDirectoryDask(*directory*, *output_dir='HDF5'*, *morphology_file=None*, *PhysSize=None*)

Loads a CyRSoXS simulation output directory into a Dask-backed qx/qy xarray.

Parameters

- **directory** (*string or Path*) – root simulation directory
- **output_dir** (*string or Path, default /HDF5*) – directory relative to the base to look for hdf5 files in.

loadDirectoryLegacy(*directory*, *output_dir='HDF5'*, *morphology_file=None*, *PhysSize=None*)

Loads a CyRSoXS simulation output directory into a qx/qy xarray.

Parameters

- **directory** (*string or Path*) – root simulation directory
- **output_dir** (*string or Path, default /HDF5*) – directory relative to the base to look for hdf5 files in.

md_loading_is_quick = False

read_config(*fname*)

Reads config.txt from a CyRSoXS simulation and produces a dictionary of values.

Parameters
fname (*string or Path*) – path to file

Returns
dict representation of config file

Return type
config

6.1.3 PyHyperScattering.integrate module

6.1.3.1 PyHyperScattering.integrate.PFEnergySeriesIntegrator module

class PyHyperScattering.PFEnergySeriesIntegrator.**PFEnergySeriesIntegrator**(***kwargs*)

Bases: *PFGeneralIntegrator*

createIntegrator(*en*, *recreate=False*)

integrateImageStack(*img_stack*, *method=None*, *chunksize=None*)

integrateImageStack_dask(*img_stack*, *chunksize=5*)

integrateImageStack_legacy(*img_stack*)

integrateSingleImage(*img*)

recreateIntegrator()

recreate the integrator, after geometry change

setupDestQ(*energies*)

setupIntegrators(*energies*)

Sets up the integrator stack as a function of energy.

The final statement ensures that the integrator for the median of the set is created. This integrator is used to set the output q-binning.

Details: (copied from a message)

The fact that energy is changing during reduction means that if not forced to something, the output q bins of the integrator will move as well (since the pixel to q mappings are moving with energy). Because sparse data in q is a nightmare, we pick a given set of q bins corresponding to the median of the energies in the scan. That is a compromise between a few approaches. This line manually creates that integrator with default q binning settings so we can take those bins and tell all the other integrators to use that output q grid.

It would cosmically be better (for things like resolution calculation) to have the q bins actually move, but sparse arrays are computationally hard. Eventually (2-3 years of high performance Python evolution) I think that will be the right way to do it, this is an intermediate.

PyHyperScattering.PFEnergySeriesIntegrator.**inner_generator**(*df_function='apply'*)

6.1.3.2 PyHyperScattering.integrate.PFGeneralIntegrator module

PyHyperScattering.integrate.WPIntegrator module

class PyHyperScattering.PFGeneralIntegrator.**PFGeneralIntegrator**(*maskmethod='none'*, *maskpath=''*, *maskrotate=True*, *geomethod='none'*, *NIdistance=0*, *NIbcx=0*, *NIbcy=0*, *NItiltx=0*, *NItilty=0*, *NIpixsizex=0*, *NIpixsizey=0*, *template_xr=None*, *energy=2000*, *integration_method='csr_ocl'*, *correctSolidAngle=True*, *maskToNan=True*, *npts=500*, *use_log_ish_binning=False*, *do_1d_integration=False*, *return_sigma=False*, *use_chunked_processing=False*, ***kwargs*)

Bases: object

calibrationFromNikaParams(*distance*, *bcx*, *bcy*, *tiltx*, *tilty*, *pixsizex*, *pixsizey*)

DEPRECATED as of 0.2

Set the local calibrations using Nika parameters.

this will probably only support rotations in the SAXS limit (i.e., where sin(x) ~ x, i.e., a couple degrees) since it assumes the PyFAI and Nika rotations are about the same origin point (which I think isn't true).

Args:

distance: sample-detector distance in mm bcx: beam center x in pixels bcy: beam center y in pixels

tiltx: detector x tilt in deg, see note above tilty: detector y tilt in deg, see note above pixsizex: pixel size in x, microns pixsizey: pixel size in y, microns

calibrationFromTemplateXRParams(*raw_xr*)

Sets calibration from a pyFAI values in a template xarray

Parameters

raw_xr (*raw format xarray*) – a raw_xr bearing the metadata in members

property energy

integrateImageStack(*img_stack*, *method=None*, *chunksize=None*)

integrateImageStack_dask(*data*, *chunksize=5*)

integrateImageStack_legacy(*data*)

integrateSingleImage(*img*)

loadImageMask(***kwargs*)

loads a mask from a generic image

Parameters

- **maskpath** (*(pathlib.Path or String)*) – path to load
- **maskrotate** (*(bool)*) – rotate mask using np.flipud(np.rot90(mask))

loadNikaMask(*filetoload*, *rotate_image=True*, ***kwargs*)

Loads a Nika-generated HDF5 or tiff mask and converts it to an array that matches the local conventions.

Parameters

- **filetoload** (*pathlib.Path or string*) – path to hdf5/tiff format mask from Nika.
- **rotate_image** (*bool, default True*) – rotate image as should work

loadPolyMask(*maskpoints=[]*, ***kwargs*)

loads a polygon mask from a list of polygon points

Args: (list) maskpoints: a list of lists of points, e.g.

[

[#begin polygon 1
[0,0],[0,10],[10,10],[10,0]

], [#later polygons]

]

(tuple) maskshape: (x,y) dimensions of mask to create
if not passed, will assume that the maximum point is included in the mask

loadPyHyperMask(***kwargs*)

Loads a mask json file saved by PyHyper's drawMask routines.

Parameters

maskpath (*(pathlib.Path or string)*) – path to load json file from

property ni_beamcenter_x

property ni_beamcenter_y

property ni_distance

property ni_pixel_x

property ni_pixel_y

property ni_tilt_x

property ni_tilt_y

recreateIntegrator()

recreate the integrator, after geometry change

property wavelength

PyHyperScattering.PFGeneralIntegrator.**inner_generator**(*df_function='apply'*)

Integrator for qx/qy format xarrays using skimage.transform.warp_polar or a custom cuda-accelerated version, warp_polar_gpu

6.1.4 PyHyperScattering.util module

6.1.4.1 PyHyperScattering.util.Fitting module

6.1.4.2 PyHyperScattering.util.HDR module

6.1.4.3 PyHyperScattering.util.IntegrationUtils module

6.1.4.4 PyHyperScattering.util.FileIO module

6.1.4.5 PyHyperScattering.util.RSoXS module

6.1.5 Module contents

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

PyHyperScattering, 23
PyHyperScattering.integrate, 20
PyHyperScattering.integrate.WPIntegrator, 23
PyHyperScattering.load, 13
PyHyperScattering.PFEnergySeriesIntegrator, 20
PyHyperScattering.PFGeneralIntegrator, 21
PyHyperScattering.util, 23
PyHyperScattering.util.FileIO, 23
PyHyperScattering.util.Fitting, 23
PyHyperScattering.util.HDR, 23
PyHyperScattering.util.IntegrationUtils, 23
PyHyperScattering.util.RSoXS, 23

INDEX

A

ALS11012RSoXSLoader (*class in PyHyperScattering.load*), 14

B

background() (*PyHyperScattering.load.SST1RSoXSDB method*), 16

C

calibrationFromNikaParams() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 21
calibrationFromTemplateXRParams() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
createIntegrator() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
cyrsoxsLoader (*class in PyHyperScattering.load*), 19

D

do_list_append() (*PyHyperScattering.load.SST1RSoXSDB method*), 16

E

energy (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 22

F

file_ext (*PyHyperScattering.load.ALS11012RSoXSLoader attribute*), 14
file_ext (*PyHyperScattering.load.cyrsoxsLoader attribute*), 19
file_ext (*PyHyperScattering.load.FileLoader attribute*), 13
file_ext (*PyHyperScattering.load.SST1RSoXSDB attribute*), 16
file_ext (*PyHyperScattering.load.SST1RSoXSLoader attribute*), 15
FileLoader (*class in PyHyperScattering.load*), 13

I

inner_generator() (*in module PyHyperScattering.PFEnergySeriesIntegrator*), 21
inner_generator() (*in module PyHyperScattering.PFGeneralIntegrator*), 23
integrateImageStack() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
integrateImageStack() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
integrateImageStack_dask() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
integrateImageStack_dask() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
integrateImageStack_legacy() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
integrateImageStack_legacy() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
integrateSingleImage() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
integrateSingleImage() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22

L

loadDarks() (*PyHyperScattering.load.ALS11012RSoXSLoader method*), 14
loadDirectory() (*PyHyperScattering.load.cyrsoxsLoader method*), 20
loadDirectoryDask() (*PyHyperScattering.load.cyrsoxsLoader method*), 20
loadDirectoryLegacy() (*PyHyperScattering.load.cyrsoxsLoader method*), 20

loadFileSeries() (*PyHyperScattering.load.FileLoader method*), 13
loadImageMask() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
loadMd() (*PyHyperScattering.load.SST1RSoXSDB method*), 16
loadMd() (*PyHyperScattering.load.SST1RSoXSLoader method*), 15
loadMonitors() (*PyHyperScattering.load.SST1RSoXSDB method*), 16
loadNikaMask() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
loadPolyMask() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
loadPyHyperMask() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 22
loadRun() (*PyHyperScattering.load.SST1RSoXSDB method*), 17
loadSampleSpecificDarks() (*PyHyperScattering.load.ALS11012RSoXSLoader method*), 14
loadSeries() (*PyHyperScattering.load.SST1RSoXSDB method*), 17
loadSingleImage() (*PyHyperScattering.load.ALS11012RSoXSLoader method*), 15
loadSingleImage() (*PyHyperScattering.load.FileLoader method*), 14
loadSingleImage() (*PyHyperScattering.load.SST1RSoXSDB method*), 17
loadSingleImage() (*PyHyperScattering.load.SST1RSoXSLoader method*), 15

M

md_loading_is_quick (*PyHyperScattering.load.ALS11012RSoXSLoader attribute*), 15
md_loading_is_quick (*PyHyperScattering.load.cyrsoxsLoader attribute*), 20
md_loading_is_quick (*PyHyperScattering.load.FileLoader attribute*), 14
md_loading_is_quick (*PyHyperScattering.load.SST1RSoXSDB attribute*), 17
md_loading_is_quick (*PyHyperScattering.load.SST1RSoXSLoader attribute*), 15
md_lookup (*PyHyperScattering.load.SST1RSoXSDB attribute*), 17
md_secondary_lookup (*PyHyperScattering.load.SST1RSoXSDB attribute*), 17
module
PyHyperScattering, 23
PyHyperScattering.integrate, 20
PyHyperScattering.integrate.WPIntegrator, 23
PyHyperScattering.load, 13
PyHyperScattering.PFEnergySeriesIntegrator, 20
PyHyperScattering.PFGeneralIntegrator, 21
PyHyperScattering.util, 23
PyHyperScattering.util.FileIO, 23
PyHyperScattering.util.Fitting, 23
PyHyperScattering.util.HDR, 23
PyHyperScattering.util.IntegrationUtils, 23
PyHyperScattering.util.RSoXS, 23

N

ni_beamcenter_x (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 22
ni_beamcenter_y (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 22
ni_distance (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23
ni_pixel_x (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23
ni_pixel_y (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23
ni_tilt_x (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23
ni_tilt_y (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23
normalizeMetadata() (*PyHyperScattering.load.ALS11012RSoXSLoader method*), 15

P

peekAtMd() (*PyHyperScattering.load.ALS11012RSoXSLoader method*), 15
peekAtMd() (*PyHyperScattering.load.FileLoader method*), 14
peekAtMd() (*PyHyperScattering.load.SST1RSoXSDB method*), 17
peekAtMd() (*PyHyperScattering.load.SST1RSoXSLoader method*), 15
PFEnergySeriesIntegrator (*class in PyHyperScattering.PFEnergySeriesIntegrator*), 20

PFGeneralIntegrator (*class in PyHyperScattering.PFGeneralIntegrator*), 21
pix_size_1 (*PyHyperScattering.load.SST1RSoXSDB attribute*), 18
pix_size_1 (*PyHyperScattering.load.SST1RSoXSLoader attribute*), 15
pix_size_2 (*PyHyperScattering.load.SST1RSoXSDB attribute*), 18
pix_size_2 (*PyHyperScattering.load.SST1RSoXSLoader attribute*), 15
PyHyperScattering
    module, 23
PyHyperScattering.integrate
    module, 20
PyHyperScattering.integrate.WPIntegrator
    module, 23
PyHyperScattering.load
    module, 13
PyHyperScattering.PFEnergySeriesIntegrator
    module, 20
PyHyperScattering.PFGeneralIntegrator
    module, 21
PyHyperScattering.util
    module, 23
PyHyperScattering.util.FileIO
    module, 23
PyHyperScattering.util.Fitting
    module, 23
PyHyperScattering.util.HDR
    module, 23
PyHyperScattering.util.IntegrationUtils
    module, 23
PyHyperScattering.util.RSoXS
    module, 23

R

read_baseline() (*PyHyperScattering.load.SST1RSoXSLoader method*), 15
read_config() (*PyHyperScattering.load.cyrsoxsLoader method*), 20
read_json() (*PyHyperScattering.load.SST1RSoXSLoader method*), 15
read_primary() (*PyHyperScattering.load.SST1RSoXSLoader method*), 16
read_shutter_toggle() (*PyHyperScattering.load.SST1RSoXSLoader method*), 16
recreateIntegrator() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
recreateIntegrator() (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator method*), 23
runSearch() (*PyHyperScattering.load.SST1RSoXSDB method*), 18

S

setupDestQ() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
setupIntegrators() (*PyHyperScattering.PFEnergySeriesIntegrator.PFEnergySeriesIntegrator method*), 20
SST1RSoXSDB (*class in PyHyperScattering.load*), 16
SST1RSoXSLoader (*class in PyHyperScattering.load*), 15
summarize_run() (*PyHyperScattering.load.SST1RSoXSDB method*), 18

W

wavelength (*PyHyperScattering.PFGeneralIntegrator.PFGeneralIntegrator property*), 23