pyFAI

API Documentation

November 27, 2012

Contents

1 Package pyFAI

1.1 Modules

- **_geometry** *(Section 2, p. 5)*
- **azimuthalIntegrator**: This class defines azimuthal integrators (ai here-after).
  (Section 3, p. 6)
- **bilinear** *(Section 4, p. 24)*
- **detectors** *(Section 5, p. 25)*
- **fastcrc**: Simple Cython module for doing CRC32 for checksums, possibly with SSE4 acceleration
  (Section 6, p. 36)
- **geometry** *(Section 7, p. 37)*
- **geometryRefinement** *(Section 8, p. 50)*
- **histogram** *(Section 9, p. 55)*
- **ocl_azim**: C++ less implementation of Dimitris' code based on PyOpenCL
  (Section 10, p. 56)
- **ocl_azim_lut** *(Section 11, p. 62)*
- **opencl** *(Section 12, p. 64)*
- **peakPicker** *(Section 13, p. 68)*
- **reconstruct** *(Section 14, p. 74)*
- **refinment2D** *(Section 15, p. 75)*
- **relabel** *(Section 16, p. 77)*
- **spline**: This is piece of software aims to manipulate spline files for geometric corrections of the 2D detectors using cubic-spline
  (Section 17, p. 78)
- **splitBBox** *(Section 18, p. 83)*
- **splitBBoxLUT** *(Section 19, p. 84)*
- **splitPixel** *(Section 20, p. 85)*
- **utils** *(Section 21, p. 86)*

1.2 Variables

Name	Description
version	**Value:** `'0.8.0'`
logger	**Value:** `logging.getLogger("pyFAI.__init__")`
__package__	**Value:** `'pyFAI'`

2 Module pyFAI._geometry

2.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`
__test__	**Value:** {}

3 Module pyFAI.azimuthalIntegrator

This class defines azimuthal integrators (ai here-after). main methods are:

tth,I = ai.xrpd(data,nbPt) q,I,sigma = ai.saxs(data,nbPt)

Date: 02/07/2012

Author: Jerome Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

3.1 Variables

Name	Description
__status__	**Value:** `'beta'`
logger	**Value:** `logging.getLogger("pyFAI.azimuthalIntegrator")`
ocl	**Value:** `OpenCL devic...`
__package__	**Value:** `'pyFAI'`

3.2 Class AzimuthalIntegrator

object

pyFAI.geometry.Geometry

pyFAI.azimuthalIntegrator.AzimuthalIntegrator

Known Subclasses: pyFAI.geometryRefinement.GeometryRefinement

This class is an azimuthal integrator based on P. Boesecke's geometry and histogram algorithm by Manolo S. del Rio and V.A Sole

All geometry calculation are done in the Geometry class

3.2.1 Methods

__init__(*self*, *dist*=`1`, *poni1*=`0`, *poni2*=`0`, *rot1*=`0`, *rot2*=`0`, *rot3*=`0`, *pixel1*=`None`, *pixel2*=`None`, *splineFile*=`None`, *detector*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

- `dist:` distance sample - detector plan (orthogonal distance, not along the beam), in meter.
- `poni1:` coordinate of the point of normal incidence along the detector's first dimension, in meter
- `poni2:` coordinate of the point of normal incidence along the detector's second dimension, in meter
- `rot1:` first rotation from sample ref to detector's ref, in radians
- `rot2:` second rotation from sample ref to detector's ref, in radians
- `rot3:` third rotation from sample ref to detector's ref, in radians
- `pixel1:` pixel size of the fist dimension of the detector, in meter
- `pixel2:` pixel size of the second dimension of the detector, in meter
- `splineFile:` file containing the geometric distortion of the detector. Overrides the pixel size.
- `detector:` name of the detector or Detector instance.

Overrides: object.__init__

makeMask(*self*, *data*, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *mode*=`'normal'`)

Combines various masks...

Normal mode: False for valid pixels, True for bad pixels Numpy mode: True for valid pixels, false for others

Parameters

- `data:` input array of data
- `mask:` input mask (if none, self.mask is used)
- `dummy:` value of dead pixels
- `delta_dumy:` precision of dummy pixels
- `mode:` can be "normal" or "numpy"

Return Value

array of boolean

xrpd_numpy(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`)

Calculate the powder diffraction pattern from a set of data, an image. Numpy implementation

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in

(type=string)

`correctSolidAngle:` if True, the data are divided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same size as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`polarization_factor:` polarization factor between -1 and +1. 0 for no correction

`dark:` dark noise image

`flat:` flat field image

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd_cython(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`, *pixelSize*=`None`)

Calculate the powder diffraction pattern from a set of data, an image.

Old cython implementation, you should not use it.

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in

(type=string)

`correctSolidAngle:` if True, the data are divided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same size as image) with 1 for masked pixels, and 0 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`polarization_factor:` polarization factor between -1 and +1. 0 for no correction

`dark:` dark noise image

`flat:` flat field image

`pixelSize:` extension of pixels in 2theta (and radians) ... for pixel splittinh

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd_splitBBox(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`)

Calculate the powder diffraction pattern from a set of data, an image. Cython implementation

Add in the cython part a dark and a flat images to be corrected on the fly. This is the default and prefered method.

Parameters

`data`: 2D array from the CCD camera

(type=ndarray)

`nbPt`: number of points in the output pattern

(type=integer)

`filename`: file to save data in ascii format 2 column

(type=string)

`correctSolidAngle`: if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange`: The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange`: The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask`: array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy`: value for dead/masked pixels

`delta_dummy`: precision for dummy value

`polarization_factor`: polarization factor between -1 and +1. 0 for no correction

`dark`: dark noise image

`flat`: flat field image

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd_splitPixel(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`)

Calculate the powder diffraction pattern from a set of data, an image.

Cython implementation

Parameters

`data:`	2D array from the CCD camera *(type=ndarray)*
`nbPt:`	number of points in the output pattern *(type=integer)*
`filename:`	file to save data in *(type=string)*
`mask:`	array (same siza as image) with 0 for masked pixels, and 1 for valid pixels
`dummy:`	value for dead/masked pixels
`delta_dummy:`	precision for dummy value
`polarization_factor:`	polarization factor between -1 and +1. 0 for no correction
`dark:`	dark noise image
`flat:`	flat field image

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`)

Calculate the powder diffraction pattern from a set of data, an image. Cython implementation

Add in the cython part a dark and a flat images to be corrected on the fly. This is the default and prefered method.

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in ascii format 2 column

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`polarization_factor:` polarization factor between -1 and +1. 0 for no correction

`dark:` dark noise image

`flat:` flat field image

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd_OpenCL(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *devicetype*=`'gpu'`, *useFp64*=`True`, *platformid*=`None`, *deviceid*=`None`, *safe*=`True`)

Calculate the powder diffraction pattern from a set of data, an image. Cython implementation

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in ascii format 2 column

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional, disabled for now)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value OpenCL specific parameters:

`devicetype:` "cpu" or "gpu" or "all" or "def"

`useFp64:` shall histogram be done in double precision (adviced)

`platformid:` platform number

`deviceid:` device number

`safe:` set to false if you think your GPU is already set-up correctly (2theta, mask, solid angle...)

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd2_numpy(*self*, *data*, *nbPt2Th*, *nbPtChi*=`360`, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`)

Calculate the 2D powder diffraction pattern (2Theta,Chi) from a set of data, an image

Pure numpy implementation (VERY SLOW !!!)

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt2Th:` number of points in the output pattern in the Radial (horizontal) axis (2 theta)

`nbPtChi:` number of points in the output pattern along the Azimuthal (vertical) axis (chi)

(type=integer)

`filename:` file to save data in

(type=string)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`nbPt:` *(type=integer)*

Return Value

azimuthaly regrouped data, 2theta pos and chipos

(type=3-tuple of ndarrays)

xrpd2_histogram(*self*, *data*, *nbPt2Th*, *nbPtChi*=`360`, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`)

Calculate the 2D powder diffraction pattern (2Theta,Chi) from a set of data, an image

Cython implementation: fast but incaccurate

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt2Th:` number of points in the output pattern in the Radial (horizontal) axis (2 theta)

`nbPtChi:` number of points in the output pattern along the Azimuthal (vertical) axis (chi)

(type=integer)

`filename:` file to save data in

(type=string)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`nbPt:` *(type=integer)*

Return Value

azimuthaly regrouped data, 2theta pos and chipos

(type=3-tuple of ndarrays)

xrpd2_splitBBox(*self*, *data*, *nbPt2Th*, *nbPtChi*=`360`, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`)

Calculate the 2D powder diffraction pattern (2Theta,Chi) from a set of data, an image

Split pixels according to their coordinate and a bounding box

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt2Th:` number of points in the output pattern in the Radial (horizontal) axis (2 theta)

`nbPtChi:` number of points in the output pattern along the Azimuthal (vertical) axis (chi)

(type=integer)

`filename:` file to save data in

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the azimuthal angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`nbPt:` *(type=integer)*

Return Value

azimuthaly regrouped data, 2theta pos. and chi pos.

(type=3-tuple of ndarrays)

xrpd2_splitPixel(*self*, *data*, *nbPt2Th*, *nbPtChi*=360, *filename*=None, *correctSolidAngle*=True, *tthRange*=None, *chiRange*=None, *mask*=None, *dummy*=None, *delta_dummy*=None, *polarization_factor*=0, *dark*=None, *flat*=None)

Calculate the 2D powder diffraction pattern (2Theta,Chi) from a set of data, an image

Split pixels according to their corner positions

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt2Th:` number of points in the output pattern in the Radial (horizontal) axis (2 theta)

`nbPtChi:` number of points in the output pattern along the Azimuthal (vertical) axis (chi)

(type=integer)

`filename:` file to save data in

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the azimuthal angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`polarization_factor:` polarization factor between -1 and +1. 0 for no correction

`dark:` dark noise image

`flat:` flat field image

`nbPt:` *(type=integer)*

Return Value

azimuthaly regrouped data, 2theta pos. and chi pos.

(type=3-tuple of ndarrays)

xrpd2(*self*, *data*, *nbPt2Th*, *nbPtChi*=`360`, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`)

Calculate the 2D powder diffraction pattern (2Theta,Chi) from a set of data, an image

Split pixels according to their coordinate and a bounding box

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt2Th:` number of points in the output pattern in the Radial (horizontal) axis (2 theta)

`nbPtChi:` number of points in the output pattern along the Azimuthal (vertical) axis (chi)

(type=integer)

`filename:` file to save data in

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the azimuthal angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`nbPt:` *(type=integer)*

Return Value

azimuthaly regrouped data, 2theta pos. and chi pos.

(type=3-tuple of ndarrays)

save2D(*self*, *filename*, *I*, *dim1*, *dim2*, *dim1_unit*=`'2th'`)

saxs(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *variance*=`None`, *error_model*=`None`, *qRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *polarization_factor*=`0`, *dark*=`None`, *flat*=`None`, *method*=`'bbox'`)

Calculate the azimuthal integrated Saxs curve

Multi algorithm implementation (tries to be bullet proof)

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data to

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`variance:` array containing the variance of the data, if you know it

(type=ndarray)

`error_model:` When the variance is unknown, an error model can be given: "poisson" (variance = I), "azimuthal" (variance = (I-<I>)ˆ2)

(type=string)

`qRange:` The lower and upper range of the sctter vector q. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`mask:` array (same size as image) with 1 for masked pixels, and 0 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value

`polarization_factor:` polarization factor between -1 and +1. 0 for no correction

`dark:` dark noise image

`flat:` flat field image

`method:` can be "numpy", "cython", "BBox" or "splitpixel"

Return Value

azimuthaly regrouped data, 2theta pos. and chi pos.

(type=3-tuple of ndarrays)

makeHeaders(*self*, *hdr*=`'#'`)

Return Value

a string to be used for headers

get_darkcurrent(*self*)

get_flatfield(*self*)

get_mask(*self*)

get_maskfile(*self*)

reset(*self*)

Reset azimuthal integrator in addition to other arrays.

Overrides: pyFAI.geometry.Geometry.reset

save1D(*self*, *filename*, *dim1*, *I*, *error*=`None`, *dim1_unit*=`'2th_deg'`)

set_darkcurrent(*self*, *dark*)

set_flatfield(*self*, *flat*)

set_mask(*self*, *mask*)

set_maskfile(*self*, *maskfile*)

setup_LUT(*self*, *shape*, *nbPt*, *mask*=`None`, *tthRange*=`None`, *chiRange*=`None`, *mask_checksum*=`None`)

xrpd_LUT(*self*, *data*, *nbPt*, *filename*=`None`, *correctSolidAngle*=`True`, *tthRange*=`None`, *chiRange*=`None`, *mask*=`None`, *dummy*=`None`, *delta_dummy*=`None`, *safe*=`True`)

Calculate the powder diffraction pattern from a set of data, an image. Cython implementation using a Look-Up Table.

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in ascii format 2 column

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional, disabled for now)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value LUT specific parameters:

`safe:` set to false if you think your GPU is already set-up correctly (2theta, mask, solid angle...)

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

xrpd_LUT_OCL(*self*, *data*, *nbPt*, *filename*=None, *correctSolidAngle*=True, *tthRange*=None, *chiRange*=None, *mask*=None, *dummy*=None, *delta_dummy*=None, *safe*=True, *devicetype*='all', *platformid*=None, *deviceid*=None)

Calculate the powder diffraction pattern from a set of data, an image. Cython implementation using a Look-Up Table.

Parameters

`data:` 2D array from the CCD camera

(type=ndarray)

`nbPt:` number of points in the output pattern

(type=integer)

`filename:` file to save data in ascii format 2 column

(type=string)

`correctSolidAngle:` if True, the data are devided by the solid angle of each pixel

(type=boolean)

`tthRange:` The lower and upper range of the 2theta. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional)

`chiRange:` The lower and upper range of the chi angle. If not provided, range is simply (data.min(), data.max()). Values outside the range are ignored.

(type=(float, float), optional, disabled for now)

`mask:` array (same siza as image) with 0 for masked pixels, and 1 for valid pixels

`dummy:` value for dead/masked pixels

`delta_dummy:` precision for dummy value LUT specific parameters:

`safe:` set to false if you think your GPU is already set-up correctly (2theta, mask, solid angle...) OpenCL specific parameters:

`devicetype:` can be "all", "cpu" or "gpu"

Return Value

(2theta, I) in degrees

(type=2-tuple of 1D arrays)

Inherited from pyFAI.geometry.Geometry(Section 7.2)

__repr__(), calcfrom1d(), chi(), chiArray(), chi_corner(), cornerArray(), cornerQArray(), del_chia(), del_dssa(), del_qa(), del_ttha(), delta2Theta(), deltaChi(), deltaQ(), diffSolidAngle(), getFit2D(), getPyFAI(), get_chia(), get_correct_solid_angle_for_spline(), get_dist(), get_dssa(), get_pixel1(), get_pixel2(), get_poni1(), get_poni2(), get_qa(), get_rot1(), get_rot2(), get_rot3(), get_spline(), get_splineFile(), get_ttha(), get_wavelength(), load(), oversampleArray(), polarization(), qArray(), qCornerFunct(), qFunction(), read(), save(), setChiDiscAtPi(), setChiDiscAtZero(), setFit2D(), setOversampling(), setPyFAI(), set_chia(), set_correct_solid_angle_for_spline(), set_dist(), set_dssa(), set_pixel1(), set_pixel2(), set_poni1(), set_poni2(), set_qa(), set_rot1(), set_rot2(), set_rot3(), set_spline(), set_splineFile(), set_ttha(), set_wavelength(), sload(), solidAngleArray(), tth(), tth_corner(), twoThetaArray(), write()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

3.2.2 Properties

Name	Description
darkcurrent	
flatfield	
mask	
maskfile	
Inherited from pyFAI.geometry.Geometry (Section 7.2) chia, correct_SA_spline, dist, dssa, pixel1, pixel2, poni1, poni2, qa, rot1, rot2, rot3, spline, splineFile, ttha, wavelength	
Inherited from object __class__	

4 Module pyFAI.bilinear

Date: 27/10/2012

Author: Jerome Kieffer

Contact: jerome.kieffer@esrf.fr

Copyright: 2011-2012, ESRF

License: GPLv3

4.1 Variables

Name	Description
__package__	**Value:** 'pyFAI'
__test__	**Value:** {}

5 Module pyFAI.detectors

Date: 12/04/2012

Author: J\xc3\xa9r\xc3\xb4me Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

5.1 Functions

detector_factory(*name*)

A kind of factory...

Parameters
    `name`: name of a detector

Return Value
    an instance of the right detector

5.2 Variables

Name	Description
__status__	**Value:** 'beta'
logger	**Value:** logging.getLogger("pyFAI.detectors")
ALL_DETECTORS	**Value:** {'condor': <class 'pyFAI.detectors.Fairchild'>, 'fairchil...
__package__	**Value:** 'pyFAI'

5.3 Class Detector

object ─┐
    pyFAI.detectors.Detector

Known Subclasses: pyFAI.detectors.FReLoN, pyFAI.detectors.Fairchild, pyFAI.detectors.Pilatus, pyFAI.detectors.Xpad_flat

Generic class representing a 2D detector

5.3.1 Methods

__init__(*self*, *pixel1*=None, *pixel2*=None, *splineFile*=None)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

get_splineFile(*self*)

set_splineFile(*self*, *splineFile*)

get_binning(*self*)

set_binning(*self*, *bin_size*=(1, 1))

getPyFAI(*self*)

getFit2D(*self*)

setPyFAI(*self*, ***kwarg*)

setFit2D(*self*)

calc_cartesian_positions(*self*, *d1*=`None`, *d2*=`None`)

Calculate the position of each pixel center in cartesian coordinate and in meter of a couple of coordinates. The half pixel offset is taken into account here !!!

Parameters

`d1`: ndarray of dimension 1 or 2 containing the Y pixel positions

`d2`: ndarray of dimension 1or 2 containing the X pixel positions

Return Value

2-arrays of same shape as d1 & d2 with the position in meter

d1 and d2 must have the same shape, returned array will have the same shape.

calc_mask(*self*)

Detectors with gaps should overwrite this method with something actually calculating the mask!

get_mask(*self*)

set_mask(*self*, *mask*)

set_maskfile(*self*, *maskfile*)

get_maskfile(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.3.2 Properties

Name	Description
splineFile	
binning	
mask	
maskfile	
Inherited from object	
__class__	

5.4 Class Pilatus

object ─┐
pyFAI.detectors.Detector ─┐
pyFAI.detectors.Pilatus

Known Subclasses: pyFAI.detectors.Pilatus1M, pyFAI.detectors.Pilatus2M, pyFAI.detectors.Pilatus6M

Pilatus detector: generic description

5.4.1 Methods

__init__(*self*, *pixel1*=`0.000172`, *pixel2*=`0.000172`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

calc_mask(*self*)

Returns a generic mask for Pilatus detectors...

Overrides: pyFAI.detectors.Detector.calc_mask

Inherited from pyFAI.detectors.Detector(Section 5.3)

calc_cartesian_positions(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.4.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	

continued on next page

Name	Description
binning, mask, maskfile, splineFile	
Inherited from object __class__	

5.4.3 Class Variables

Name	Description
MODULE_SIZE	**Value:** `(195, 487)`
MODULE_GAP	**Value:** `(17, 7)`

5.5 Class Pilatus1M

object
pyFAI.detectors.Detector
pyFAI.detectors.Pilatus
pyFAI.detectors.Pilatus1M

Pilatus 1M detector

5.5.1 Methods

__init__(*self*, *pixel1*=`0.000172`, *pixel2*=`0.000172`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from pyFAI.detectors.Pilatus(Section 5.4)

__repr__(), calc_mask()

Inherited from pyFAI.detectors.Detector(Section 5.3)

calc_cartesian_positions(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.5.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	
binning, mask, maskfile, splineFile	
Inherited from object	
__class__	

5.5.3 Class Variables

Name	Description
Inherited from pyFAI.detectors.Pilatus (Section 5.4)	
MODULE_GAP, MODULE_SIZE	

5.6 Class Pilatus2M

object
pyFAI.detectors.Detector
pyFAI.detectors.Pilatus
pyFAI.detectors.Pilatus2M

Pilatus 2M detector

5.6.1 Methods

__init__(*self*, *pixel1*=`0.000172`, *pixel2*=`0.000172`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from pyFAI.detectors.Pilatus(Section 5.4)

__repr__(), calc_mask()

Inherited from pyFAI.detectors.Detector(Section 5.3)

calc_cartesian_positions(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.6.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	
binning, mask, maskfile, splineFile	
Inherited from object	
__class__	

5.6.3 Class Variables

Name	Description
Inherited from pyFAI.detectors.Pilatus (Section 5.4)	
MODULE_GAP, MODULE_SIZE	

5.7 Class Pilatus6M

object

pyFAI.detectors.Detector

pyFAI.detectors.Pilatus

pyFAI.detectors.Pilatus6M

Pilatus 6M detector

5.7.1 Methods

__init__(*self*, *pixel1*=`0.000172`, *pixel2*=`0.000172`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from pyFAI.detectors.Pilatus(Section 5.4)

__repr__(), calc_mask()

Inherited from pyFAI.detectors.Detector(Section 5.3)

calc_cartesian_positions(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.7.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	
binning, mask, maskfile, splineFile	
Inherited from object	
__class__	

5.7.3 Class Variables

Name	Description
Inherited from pyFAI.detectors.Pilatus (Section 5.4)	
MODULE_GAP, MODULE_SIZE	

5.8 Class Fairchild

object

pyFAI.detectors.Detector

pyFAI.detectors.Fairchild

Fairchild Condor 486:90 detector

5.8.1 Methods

__init__(*self*, *pixel1*=`1.5e-05`, *pixel2*=`1.5e-05`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from pyFAI.detectors.Detector(Section 5.3)

__repr__(), calc_cartesian_positions(), calc_mask(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.8.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	
binning, mask, maskfile, splineFile	
Inherited from object	
__class__	

5.9 Class FReLoN

object
└ pyFAI.detectors.Detector
  └ **pyFAI.detectors.FReLoN**

FReLoN detector (spline mandatory to correct for geometric distortion)

5.9.1 Methods

__init__(*self*, *splineFile*)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

Inherited from pyFAI.detectors.Detector(Section 5.3)

__repr__(), calc_cartesian_positions(), calc_mask(), getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.9.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3)	
binning, mask, maskfile, splineFile	
Inherited from object	
__class__	

5.10 Class Xpad_flat

object
pyFAI.detectors.Detector
pyFAI.detectors.Xpad_flat

Xpad detector: generic description for image with

5.10.1 Methods

__init__(*self*, *pixel1*=`0.00013`, *pixel2*=`0.00013`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

calc_mask(*self*)

Returns a generic mask for Xpad detectors... discards the first line and raw form all modules: those are 2.5x bigger and often mis - behaving

Overrides: pyFAI.detectors.Detector.calc_mask

calc_cartesian_positions(*self*, *d1*=`None`, *d2*=`None`)

Calculate the position of each pixel center in cartesian coordinate and in meter of a couple of coordinates. The half pixel offset is taken into account here !!!

Parameters

`d1`: ndarray of dimension 1 or 2 containing the Y pixel positions

`d2`: ndarray of dimension 1or 2 containing the X pixel positions

Return Value

2-arrays of same shape as d1 & d2 with the position in meter

d1 and d2 must have the same shape, returned array will have the same shape.

Overrides: pyFAI.detectors.Detector.calc_cartesian_positions

Inherited from pyFAI.detectors.Detector(Section 5.3)

getFit2D(), getPyFAI(), get_binning(), get_mask(), get_maskfile(), get_splineFile(), setFit2D(), setPyFAI(), set_binning(), set_mask(), set_maskfile(), set_splineFile()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

5.10.2 Properties

Name	Description
Inherited from pyFAI.detectors.Detector (Section 5.3) binning, mask, maskfile, splineFile	
Inherited from object __class__	

5.10.3 Class Variables

Name	Description
MODULE_SIZE	**Value:** `(120, 80)`
MODULE_GAP	**Value:** `(30.4615384615, 3)`

6 Module pyFAI.fastcrc

Simple Cython module for doing CRC32 for checksums, possibly with SSE4 acceleration

Date: 19-11-2012

Author: Jerome Kieffer

Contact: Jerome.kieffer@esrf.fr

License: GPL v3+

6.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`
__test__	**Value:** {}

7 Module pyFAI.geometry

Date: 09/06/2012

Author: Jerome Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

7.1 Variables

Name	Description
__status__	**Value:** `'beta'`
logger	**Value:** `logging.getLogger("pyFAI.geometry")`
__package__	**Value:** `'pyFAI'`

7.2 Class Geometry

object ─┐
pyFAI.geometry.Geometry

Known Subclasses: pyFAI.azimuthalIntegrator.AzimuthalIntegrator

```
This class is an azimuthal integrator based on P. Boesecke's geometry and
histogram algorithm by Manolo S. del Rio and V.A Sole

Detector is assumed to be corrected from "raster orientation" effect.
It is not addressed here but rather in the Detector object or at read time.
Considering there is no tilt:
Detector fast dimension (dim2) is supposed to be horizontal (dimension X of the image)
Detector slow dimension (dim1) is supposed to be vertical, upwards (dimension Y of the i
The third dimension is chose such as the referential is orthonormal, so dim3 is along in

Demonstration of the equation done using Mathematica.
=====================================================



Axis 1 is along first dimension of detector (when not tilted), this is the slow dimensio
```

```
 x1={1,0,0}
Axis 2 is along second dimension of detector (when not tilted), this is the fast dimensi
 x2={0,1,0}
Axis 3 is along the incident X-Ray beam
 x3={0,0,1}
We define the 3 rotation around axis 1, 2 and 3:
 rotM1 = RotationMatrix[rot1,x1] =  {{1,0,0},{0,cos[rot1],-sin[rot1]},{0,sin[rot1],cos[r
 rotM2 =  RotationMatrix[rot2,x2] = {{cos[rot2],0,sin[rot2]},{0,1,0},{-sin[rot2],0,cos[r
 rotM3 =  RotationMatrix[rot3,x3] = {{cos[rot3],-sin[rot3],0},{sin[rot3],cos[rot3],0},{0


Rotations of the detector are applied first Rot around axis 1, then axis 2 and finally a
 R = rotM3.rotM2.rotM1
   = {{cos[rot2] cos[rot3],cos[rot3] sin[rot1] sin[rot2]-cos[rot1] sin[rot3],cos[rot1] c
      {cos[rot2] sin[rot3],cos[rot1] cos[rot3]+sin[rot1] sin[rot2] sin[rot3],-cos[rot3]
      {-sin[rot2],cos[rot2] sin[rot1],cos[rot1] cos[rot2]}}
In Python notation:
PForm[R.x1] = [cos(rot2)*cos(rot3),cos(rot2)*sin(rot3),-sin(rot2)]
PForm[R.x2] = [cos(rot3)*sin(rot1)*sin(rot2) - cos(rot1)*sin(rot3),cos(rot1)*cos(rot3) +
PForm[R.x3] = [cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3),-(cos(rot3)*sin(rot1)

* Coordinates of the Point of Normal Incidence:
 PONI = R.{0,0,L}
 PForm[PONI] = [L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3)),
               L*(-(cos(rot3)*sin(rot1)) + cos(rot1)*sin(rot2)*sin(rot3)),L*cos(rot1)*co


* Any pixel on detector plan at coordinate (d1, d2) in meters. Detector is at z=L
 P={d1,d2,L}
 PForm[R.P] =  [t1, t2, t3] =
            = [d1*cos(rot2)*cos(rot3) + d2*(cos(rot3)*sin(rot1)*sin(rot2) - cos(rot1)*si
               d1*cos(rot2)*sin(rot3)  + d2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*s
               d2*cos(rot2)*sin(rot1) - d1*sin(rot2) + L*cos(rot1)*cos(rot2)]

* Distance sample (origin) to detector point (d1,d2)
 FForm[Norm[R.P]] = sqrt(pow(Abs(L*cos(rot1)*cos(rot2) + d2*cos(rot2)*sin(rot1) - d1*sin
                    pow(Abs(d1*cos(rot2)*cos(rot3) + d2*(cos(rot3)*sin(rot1)*sin(rot2) -
                    L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3))),2) +
                    pow(Abs(d1*cos(rot2)*sin(rot3) + L*(-(cos(rot3)*sin(rot1)) + cos(rot
                    d2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*sin(rot3))),2))

* cos(2theta) is defined as (R.P component along x3) over the distance from origin to da
 tth = ArcCos [-(R.P).x3/Norm[R.P]]
 FForm[tth] = Arccos((-(L*cos(rot1)*cos(rot2)) - d2*cos(rot2)*sin(rot1) + d1*sin(rot2))/
```

```
                    sqrt(pow(Abs(L*cos(rot1)*cos(rot2) + d2*cos(rot2)*sin(rot1) - d1*sin
                      pow(Abs(d1*cos(rot2)*cos(rot3) + d2*(cos(rot3)*sin(rot1)*sin(rot2)
                     L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3))),2) +
                      pow(Abs(d1*cos(rot2)*sin(rot3) + L*(-(cos(rot3)*sin(rot1)) + cos(r
                     d2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*sin(rot3))),2)))

* tan(2theta) is defined as sqrt(t1**2 + t2**2) / t3
 tth = ArcTan2 [sqrt(t1**2 + t2**2) , t3 ]

Getting 2theta from it's tangeant seems both more precise (around beam stop very far fro
Currently there is a swich in the method to follow one path or the other.

* Tangeant of angle chi is defined as (R.P component along x1) over (R.P component along
 chi = ArcTan[((R.P).x1) / ((R.P).x2)]
 FForm[chi] = ArcTan2(d1*cos(rot2)*cos(rot3) + d2*(cos(rot3)*sin(rot1)*sin(rot2) - cos(r
                       L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3)),
                      d1*cos(rot2)*sin(rot3) + L*(-(cos(rot3)*sin(rot1)) + cos(rot1)*sin
                        d2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*sin(rot3)))
```

7.2.1 Methods

__init__(*self*, *dist*=`1`, *poni1*=`0`, *poni2*=`0`, *rot1*=`0`, *rot2*=`0`, *rot3*=`0`, *pixel1*=`None`, *pixel2*=`None`, *splineFile*=`None`, *detector*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

- `dist:` distance sample - detector plan (orthogonal distance, not along the beam), in meter.
- `poni1:` coordinate of the point of normal incidence along the detector's first dimension, in meter
- `poni2:` coordinate of the point of normal incidence along the detector's second dimension, in meter
- `rot1:` first rotation from sample ref to detector's ref, in radians
- `rot2:` second rotation from sample ref to detector's ref, in radians
- `rot3:` third rotation from sample ref to detector's ref, in radians
- `pixel1:` pixel size of the fist dimension of the detector, in meter
- `pixel2:` pixel size of the second dimension of the detector, in meter
- `splineFile:` file containing the geometric distortion of the detector. Overrides the pixel size.

Overrides: object.__init__

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

tth(*self*, *d1*, *d2*, *param*=`None`, *path*=`'cython'`)

Calculates the 2theta value for the center of a given pixel (or set of pixels)

Parameters

`d1`: position(s) in pixel in first dimension (c order)

(type=scalar or array of scalar)

`d2`: position(s) in pixel in second dimension (c order)

(type=scalar or array of scalar)

`path`: can be ”cos”, ”tan” or ”cython” @return 2theta in radians

Return Value

floar or array of floats.

qFunction(*self*, *d1*, *d2*, *param*=`None`, *path*=`'cython'`)

Calculates the q value for the center of a given pixel (or set of pixels) in nm-1

q = 4pi/lambda sin(2theta / 2)

Parameters

`d1`: position(s) in pixel in first dimension (c order)

(type=scalar or array of scalar)

`d2`: position(s) in pixel in second dimension (c order)

(type=scalar or array of scalar @return q in in nm^(-1))

Return Value

float or array of floats.

qArray(*self*, *shape*)

Generate an array of the given shape with q(i,j) for all elements.

qCornerFunct(*self*, *d1*, *d2*)

calculate the q_vector for any pixel corner

tth_corner(*self*, *d1*, *d2*)

Calculates the 2theta value for the corner of a given pixel (or set of pixels)

Parameters

`d1`: position(s) in pixel in first dimension (c order)

(type=scalar or array of scalar)

`d2`: position(s) in pixel in second dimension (c order)

(type=scalar or array of scalar @return 2theta in radians)

Return Value

floar or array of floats.

twoThetaArray(*self*, *shape*)

Generate an array of the given shape with two-theta(i,j) for all elements.

chi(*self*, *d1*, *d2*, *path=*`'cython'`)

Calculate the chi (azimuthal angle) for the centre of a pixel at coordinate d1,d2 which in the lab ref has coordinate: X1 = p1*cos(rot2)*cos(rot3) + p2*(cos(rot3)*sin(rot1)*sin(rot2) - cos(rot1)*sin(rot3)) - L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3)) X2 = p1*cos(rot2)*sin(rot3) - L*(-(cos(rot3)*sin(rot1)) + cos(rot1)*sin(rot2)*sin(rot3)) + p2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*sin(rot3)) X3 = -(L*cos(rot1)*cos(rot2)) + p2*cos(rot2)*sin(rot1) - p1*sin(rot2) hence tan(Chi) = X2 / X1

Parameters

`d1`: pixel coordinate along the 1st dimention (C convention)

(type=float or array of them)

`d2`: pixel coordinate along the 2nd dimention (C convention)

(type=float or array of them)

`path`: can be "tan" (i.e via numpy) or "cython"

Return Value

chi, the azimuthal angle in rad

chi_corner(*self*, *d1*, *d2*)

Calculate the chi (azimuthal angle) for the corner of a pixel at coordinate d1,d2 which in the lab ref has coordinate: X1 = p1*cos(rot2)*cos(rot3) + p2*(cos(rot3)*sin(rot1)*sin(rot2) - cos(rot1)*sin(rot3)) - L*(cos(rot1)*cos(rot3)*sin(rot2) + sin(rot1)*sin(rot3)) X2 = p1*cos(rot2)*sin(rot3) - L*(-(cos(rot3)*sin(rot1)) + cos(rot1)*sin(rot2)*sin(rot3)) + p2*(cos(rot1)*cos(rot3) + sin(rot1)*sin(rot2)*sin(rot3)) X3 = -(L*cos(rot1)*cos(rot2)) + p2*cos(rot2)*sin(rot1) - p1*sin(rot2) hence tan(Chi) = X2 / X1

Parameters

`d1`: pixel coordinate along the 1st dimention (C convention)

(type=float or array of them)

`d2`: pixel coordinate along the 2nd dimention (C convention)

(type=float or array of them)

Return Value

chi, the azimuthal angle in rad

chiArray(*self*, *shape*)

Generate an array of the given shape with chi(i,j) (azimuthal angle) for all elements.

cornerArray(*self*, *shape*)

Generate a 3D array of the given shape with (i,j) (azimuthal angle) for all elements.

cornerQArray(*self*, *shape*)

Generate a 3D array of the given shape with (i,j) (azimuthal angle) for all elements.

delta2Theta(*self*, *shape*)

Generate a 3D array of the given shape with (i,j) with the max distance between the center and any corner in 2 theta

deltaChi(*self*, *shape*)

Generate a 3D array of the given shape with (i,j) with the max distance between the center and any corner in chi-angle

deltaQ(*self*, *shape*)

Generate a 3D array of the given shape with (i,j) with the max distance between the center and any corner in q_vector

diffSolidAngle(*self*, *d1*, *d2*)

calulate the solid angle of the current pixels

solidAngleArray(*self*, *shape*)

Generate an array of the given shape with the solid angle of the current element two-theta(i,j) for all elements.

save(*self*, *filename*)

Save the refined parameters.

Parameters

`filename`: name of the file where to save the parameters

(type=string)

write(*self*, *filename*)

Save the refined parameters.

Parameters

`filename`: name of the file where to save the parameters

(type=string)

sload(*cls*, *filename*)

A static method combining the constructor and the loader from a

Parameters

`filename`: name of the file to load

(type=string)

Return Value

instance of Gerometry of AzimuthalIntegrator set-up with the parameter from the file.

load(*self*, *filename*)

Load the refined parameters from a file.

Parameters

`filename`: name of the file to load

(type=string)

read(*self*, *filename*)

Load the refined parameters from a file.

Parameters

`filename`: name of the file to load

(type=string)

getPyFAI(*self*)

return the parameter set from the PyFAI geometry as a dictionary

setPyFAI(*self*, ***kwargs*)

set the geometry from a pyFAI-like dict

getFit2D(*self*)

return a dict with parameters compatible with fit2D geometry

setFit2D(*self*, *directDist*, *centerX*, *centerY*, *tilt*=`0.0`, *tiltPlanRotation*=`0.0`, *pixelX*=`None`, *pixelY*=`None`, *splineFile*=`None`)

Set the Fit2D-like parameter set: For geometry description see HPR 1996 (14) pp-240

Parameters

`direct:` direct distance from sample to detector along the incident beam (in millimeter as in fit2d)

`tilt:` tilt in degrees

`tiltPlanRotation:` Rotation (in degrees) of the tilt plan arround the Z-detector axis * 0deg -> Y does not move, +X goes to Z<0 * 90deg -> X does not move, +Y goes to Z<0 * 180deg -> Y does not move, +X goes to Z>0 * 270deg -> X does not move, +Y goes to Z>0

`pixelX, pixelY:` as in fit2d they ar given in micron, not in meter

`centerX, centerY:` pixel position of the beam center

`splineFile:` name of the file containing the spline

setChiDiscAtZero(*self*)

Set the position of the discontinuity of the chi axis between 0 and 2pi. By default it is between pi and -pi

setChiDiscAtPi(*self*)

Set the position of the discontinuity of the chi axis between -pi and +pi. This is the default behavour

setOversampling(*self*, *iOversampling*)

set the oversampling factor

oversampleArray(*self*, *myarray*)

polarization(*self*, *shape*=`None`, *factor*=`0.98`)

Calculate the polarization correction accoding to the polarization factor:

Parameters

`factor:` (Ih-Iv)/(Ih+Iv): varies between 0 (no polarization) and 1 (where division by 0 could occure) @return 2D array with polarization correction array (intensity/polarisation)

reset(*self*)

reset most arrays that are cached: used when a parameter changes.

calcfrom1d(*self*, *tth*, *I*, *shape*=`None`, *mask*=`None`, *dim1_unit*=`'2th_deg'`, *correctSolidAngle*=`True`)

Computes a 2D image from a 1D integrated profile

Parameters

`tth`: 1D array with 2theta in degrees

`I`: scattering intensity @return 2D image reconstructed

set_dist(*self*, *value*)

get_dist(*self*)

set_poni1(*self*, *value*)

get_poni1(*self*)

set_poni2(*self*, *value*)

get_poni2(*self*)

set_rot1(*self*, *value*)

get_rot1(*self*)

set_rot2(*self*, *value*)

get_rot2(*self*)

set_rot3(*self*, *value*)

get_rot3(*self*)

set_wavelength(*self*, *value*)

get_wavelength(*self*)

get_ttha(*self*)

set_ttha(*self*, *value*)

del_ttha(*self*)

get_chia(*self*)

set_chia(*self*, *value*)

del_chia(*self*)

get_dssa(*self*)

set_dssa(*self*, *value*)

del_dssa(*self*)

get_qa(*self*)

set_qa(*self*, *value*)

del_qa(*self*)

get_pixel1(*self*)

set_pixel1(*self*, *pixel1*)

get_pixel2(*self*)

set_pixel2(*self*, *pixel2*)

get_splineFile(*self*)

set_splineFile(*self*, *splineFile*)

get_spline(*self*)

set_spline(*self*, *spline*)

get_correct_solid_angle_for_spline(*self*)

set_correct_solid_angle_for_spline(*self*, *value*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

7.2.2 Properties

Name	Description
dist	
poni1	
poni2	
rot1	
rot2	
rot3	
wavelength	
ttha	2theta array in cache
chia	chi array in cache
dssa	solid angle array in cache
qa	Q array in cache
pixel1	
pixel2	
splineFile	
spline	
correct_SA_spline	
Inherited from object	
__class__	

8 Module pyFAI.geometryRefinement

Date: 23/12/2011

Author: Jerome Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

8.1 Variables

Name	Description
__status__	**Value:** `'development'`
logger	**Value:** `logging.getLogger("pyFAI.geometryRefinement")`
ROCA	**Value:** `'/opt/saxs/roca'`
__package__	**Value:** `'pyFAI'`

8.2 Class GeometryRefinement

object

pyFAI.geometry.Geometry

pyFAI.azimuthalIntegrator.AzimuthalIntegrator

pyFAI.geometryRefinement.GeometryRefineme

8.2.1 Methods

__init__(*self*, *data*, *dist*=`1`, *poni1*=`None`, *poni2*=`None`, *rot1*=`0`, *rot2*=`0`, *rot3*=`0`, *pixel1*=`None`, *pixel2*=`None`, *splineFile*=`None`, *detector*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

`data:` ndarray float64 shape = n, 3 col0: pos in dim0 (in pixels) col1: pos in dim1 (in pixels) col2: associated tth value (in rad)

`detector:` name of the detector or Detector instance.

Overrides: object.__init__

guess_poni(*self*)

Poni can be guessed by the centroid of the ring with lowest 2Theta

set_tolerance(*self*, *value*=`10`)

Parameters

`value:` Tolerance as a percentage

residu1(*self*, *param*, *d1*, *d2*, *tthRef*)

residu2(*self*, *param*, *d1*, *d2*, *tthRef*)

refine1(*self*)

refine2(*self*, *maxiter*=`1000000`)

simplex(*self*, *maxiter*=`1000000`)

anneal(*self*, *maxiter*=`1000000`)

chi2(*self*, *param*=`None`)

roca(*self*)

run roca to optimise the parameter set

set_dist_max(*self*, *value*)

get_dist_max(*self*)

set_dist_min(*self, value*)

get_dist_min(*self*)

set_poni1_min(*self, value*)

get_poni1_min(*self*)

set_poni1_max(*self, value*)

get_poni1_max(*self*)

set_poni2_min(*self, value*)

get_poni2_min(*self*)

set_poni2_max(*self, value*)

get_poni2_max(*self*)

set_rot1_min(*self, value*)

get_rot1_min(*self*)

set_rot1_max(*self, value*)

get_rot1_max(*self*)

set_rot2_min(*self, value*)

get_rot2_min(*self*)

set_rot2_max(*self, value*)

get_rot2_max(*self*)

set_rot3_min(*self, value*)

get_rot3_min(*self*)

set_rot3_max(*self*, *value*)

get_rot3_max(*self*)

Inherited from pyFAI.azimuthalIntegrator.AzimuthalIntegrator(Section 3.2)

get_darkcurrent(), get_flatfield(), get_mask(), get_maskfile(), makeHeaders(), makeMask(), reset(), save1D(), save2D(), saxs(), set_darkcurrent(), set_flatfield(), set_mask(), set_maskfile(), setup_LUT(), xrpd(), xrpd2(), xrpd2_histogram(), xrpd2_numpy(), xrpd2_splitBBox(), xrpd2_splitPixel(), xrpd_LUT(), xrpd_LUT_OCL(), xrpd_OpenCL(), xrpd_cython(), xrpd_numpy(), xrpd_splitBBox(), xrpd_splitPixel()

Inherited from pyFAI.geometry.Geometry(Section 7.2)

__repr__(), calcfrom1d(), chi(), chiArray(), chi_corner(), cornerArray(), cornerQArray(), del_chia(), del_dssa(), del_qa(), del_ttha(), delta2Theta(), deltaChi(), deltaQ(), diffSolidAngle(), getFit2D(), getPyFAI(), get_chia(), get_correct_solid_angle_for_spline(), get_dist(), get_dssa(), get_pixel1(), get_pixel2(), get_poni1(), get_poni2(), get_qa(), get_rot1(), get_rot2(), get_rot3(), get_spline(), get_splineFile(), get_ttha(), get_wavelength(), load(), oversampleArray(), polarization(), qArray(), qCornerFunct(), qFunction(), read(), save(), setChiDiscAtPi(), setChiDiscAtZero(), setFit2D(), setOversampling(), setPyFAI(), set_chia(), set_correct_solid_angle_for_spline(), set_dist(), set_dssa(), set_pixel1(), set_pixel2(), set_poni1(), set_poni2(), set_qa(), set_rot1(), set_rot2(), set_rot3(), set_spline(), set_splineFile(), set_ttha(), set_wavelength(), sload(), solidAngleArray(), tth(), tth_corner(), twoThetaArray(), write()

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

8.2.2 Properties

Name	Description
dist_max	
dist_min	
poni1_min	
poni1_max	
poni2_min	
poni2_max	
rot1_min	
rot1_max	
rot2_min	

continued on next page

Name	Description
rot2_max	
rot3_min	
rot3_max	
Inherited from pyFAI.azimuthalIntegrator.AzimuthalIntegrator (Section 3.2) darkcurrent, flatfield, mask, maskfile	
Inherited from pyFAI.geometry.Geometry (Section 7.2) chia, correct_SA_spline, dist, dssa, pixel1, pixel2, poni1, poni2, qa, rot1, rot2, rot3, spline, splineFile, ttha, wavelength	
Inherited from object __class__	

9 Module pyFAI.histogram

Date: 20120916

Author: Jerome Kieffer

9.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`
__test__	**Value:** {}

10 Module pyFAI.ocl_azim

```
C++ less implementation of Dimitris' code based on PyOpenCL

TODO and trick from dimitris still missing:
  * dark-current subtraction is still missing
  * In fact you might want to consider doing the conversion on the GPU when
    possible. Think about it, you have a uint16 to float which for large arrays
    was slow.. You load on the graphic card a uint16 (2x transfer speed) and
    you convert to float inside so it should be blazing fast.
```

Date: 07/11/2012

Author: Jerome Kieffer

Contact: jerome.kieffer@esrf.fr

Copyright: 2012, ESRF, Grenoble

License: GPLv3

10.1 Variables

Name	Description
logger	**Value:** `logging.getLogger("pyFAI.ocl_azim_pyocl")`
__package__	**Value:** `'pyFAI'`

10.2 Class Integrator1d

object ─┐
　　pyFAI.ocl_azim.Integrator1d

Attempt to implements ocl_azim using pyopencl

10.2.1 Methods

__init__(*self*, *filename*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

`filename:` file in which profiling information are saved

Overrides: object.__init__

__dealloc__(*self*)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

log(*self*, ***kwarg*)

log in a file all opencl events

getConfiguration(*self*, *Nimage*, *Nbins*, *useFp64*=`None`)

getConfiguration gets the description of the integrations to be performed and keeps an internal copy

Parameters

`Nimage:` number of pixel in image

`Nbins:` number of bins in regrouped histogram

`useFp64:` use double precision. By default the same as init!

configure(*self*, *kernel*=`None`)

The method configure() allocates the OpenCL resources required and compiled the OpenCL kernels. An active context must exist before a call to configure() and getConfiguration() must have been called at least once. Since the compiled OpenCL kernels carry some information on the integration parameters, a change to any of the parameters of getConfiguration() requires a subsequent call to configure() for them to take effect.

If a configuration exists and configure() is called, the configuration is cleaned up first to avoid OpenCL memory leaks

Parameters

`kernel_path:` is the path to the actual kernel

loadTth(*self*, *tth*, *dtth*, *tth_min*=None, *tth_max*=None)

Load the 2th arrays along with the min and max value.

loadTth maybe be recalled at any time of the execution in order to update the 2th arrays.

loadTth is required and must be called at least once after a configure()

setSolidAngle(*self*, *solidAngle*)

Enables SolidAngle correction and uploads the suitable array to the OpenCL device.

By default the program will assume no solidangle correction unless setSolidAngle() is called. From then on, all integrations will be corrected via the SolidAngle array.

If the SolidAngle array needs to be changes, one may just call setSolidAngle() again with that array

Parameters

`solidAngle:` numpy array representing the solid angle of the given pixel

unsetSolidAngle(*self*)

Instructs the program to not perform solidangle correction from now on.

SolidAngle correction may be turned back on at any point

setMask(*self*, *mask*)

Enables the use of a Mask during integration. The Mask can be updated by recalling setMask at any point.

The Mask must be a PyFAI Mask. Pixels with 0 are masked out. TODO: check and invert!

Parameters

`mask:` numpy.ndarray of integer.

unsetMask(*self*)

Disables the use of a Mask from that point. It may be re-enabled at any point via setMask

setDummyValue(*self*, *dummy*, *delta_dummy*)

Enables dummy value functionality and uploads the value to the OpenCL device.

Image values that are similar to the dummy value are set to 0.

Parameters

`dummy`: value in image of missing values (masked pixels?)

`delta_dummy`: precision for dummy values

unsetDummyValue(*self*)

Disable a dummy value. May be re-enabled at any time by setDummyValue

setRange(*self*, *lowerBound*, *upperBound*)

Instructs the program to use a user - defined range for 2th values

setRange is optional. By default the integration will use the tth_min and tth_max given by loadTth() as integration range. When setRange is called it sets a new integration range without affecting the 2th array. All values outside that range will then be discarded when interpolating. Currently, if the interval of 2th (2th + -d2th) is not all inside the range specified, it is discarded. The bins of the histogram are RESCALED to the defined range and not the original tth_max - tth_min range.

setRange can be called at any point and as many times required after a valid configuration is created.

Parameters

`lowerBound`: A float value for the lower bound of the integration range

`upperBound`: A float value for the upper bound of the integration range

unsetRange(*self*)

Disable the use of a user-defined 2th range and revert to tth_min,tth_max range

unsetRange instructs the program to revert to its default integration range. If the method is called when no user-defined range had been previously specified, no action will be performed

execute(*self*, *image*)

Perform a 1D azimuthal integration

execute() may be called only after an OpenCL device is configured and a Tth array has been loaded (at least once) It takes the input image and based on the configuration provided earlier it performs the 1D integration. Notice that if the provided image is bigger than N then only N points will be taked into account, while if the image is smaller than N the result may be catastrophic. set/unset and loadTth methods have a direct impact on the execute() method. All the rest of the methods will require at least a new configuration via configure().

Takes an image, integrate and return the histogram and weights

Parameters

`image`: image to be processed as a numpy array

Return Value

tth_out, histogram, bins

TODO: to improve performances, the image should be casted to float32 in an optimal way: currently using numpy machinery but would be better if done in OpenCL

init(*self*, *devicetype*=`'GPU'`, *useFp64*=`True`, *platformid*=`None`, *deviceid*=`None`)

Initial configuration: Choose a device and initiate a context. Devicetypes can be GPU,gpu,CPU,cpu,DEF,ACC,ALL. Suggested are GPU,CPU. For each setting to work there must be such an OpenCL device and properly installed. E.g.: If Nvidia driver is installed, GPU will succeed but CPU will fail. The AMD SDK kit (AMD APP) is required for CPU via OpenCL.

Parameters

`devicetype`: string in [”cpu”,”gpu”, ”all”, ”acc”]

`useFp64`: boolean specifying if double precision will be used

`platformid`: integer

`devid`: integer

clean(*self*, *preserve_context*=`False`)

Free OpenCL related resources allocated by the library.

clean() is used to reinitiate the library back in a vanilla state. It may be asked to preserve the context created by init or completely clean up OpenCL. Guard/Status flags that are set will be reset.

@param preserve_context Flag that preserves the context (True) or destroys all OpenCL resources (False)

get_status(*self*)

return a dictionnary with the status of the integrator: for compatibilty with former implementation

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

10.2.2 Properties

Name	Description
Inherited from object	
__class__	

11 Module pyFAI.ocl_azim_lut

Date: 18/10/2012

Author: Jerome Kieffer

Contact: jerome.kieffer@esrf.fr

Copyright: 2012, ESRF, Grenoble

License: GPLv3

11.1 Variables

Name	Description
logger	**Value:** `logging.getLogger("pyFAI.ocl_azim_lut")`
__package__	**Value:** `'pyFAI'`

11.2 Class OCL_LUT_Integrator

object ─┐
pyFAI.ocl_azim_lut.OCL_LUT_Integrator

11.2.1 Methods

__init__(*self*, *lut*, *image_size*, *devicetype*=`'all'`, *platformid*=`None`, *deviceid*=`None`, *checksum*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

- `lut`: array of uint32 - float32 with shape (nbins, lut_size) with indexes and coefficients
- `checksum`: pre - calculated checksum to prevent re - calculating it :)

Overrides: object.__init__

__del__(*self*)

Destructor: release all buffers

integrate(*self*, *data*, *dummy*=None, *delta_dummy*=None, *dark*=None, *flat*=None, *solidAngle*=None, *polarization*=None, *dark_checksum*=None, *flat_checksum*=None, *solidAngle_checksum*=None, *polarization_checksum*=None)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

11.2.2 Properties

Name	Description
Inherited from object	
__class__	

12 Module pyFAI.opencl

Date: 06/11/2012

Author: Jerome Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

12.1 Variables

Name	Description
__status__	**Value:** `'beta'`
logger	**Value:** `logging.getLogger("pyFAI.opencl")`
ocl	**Value:** `OpenCL devic...`
__package__	**Value:** `'pyFAI'`

12.2 Class Device

object
└ **pyFAI.opencl.Device**

Simple class that contains the structure of an OpenCL device

12.2.1 Methods

__init__(*self*, *name*=`None`, *type*=`None`, *version*=`None`, *driver_version*=`None`, *extensions*=`''`, *memory*=`None`, *available*=`None`, *cores*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.2.2 Properties

Name	Description
Inherited from object	
__class__	

12.3 Class Platform

object ┐
pyFAI.opencl.Platform

Simple class that contains the structure of an OpenCL platform

12.3.1 Methods

__init__(*self*, *name*=`None`, *vendor*=`None`, *version*=`None`, *extensions*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

add_device(*self*, *device*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.3.2 Properties

Name	Description
Inherited from object	

continued on next page

Name	Description
__class__	

12.4 Class OpenCL

object
pyFAI.opencl.OpenCL

Simple class that wraps the structure ocl_tools_extended.h

12.4.1 Methods

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

select_device(*self*, *type*=`'ALL'`, *memory*=`None`, *extensions*=`[]`, *best*=`True`)

Select a device based on few parameters (at the end, keep the one with most memory)

Parameters

- `type:` "gpu" or "cpu" or "all" ....
- `memory:` minimum amount of memory (int)
- `extensions:` list of extensions to be present
- `best:` shall we look for the

create_context(*self*, *devicetype*='ALL', *useFp64*=False, *platformid*=None, *deviceid*=None)

Choose a device and initiate a context.

Devicetypes can be GPU,gpu,CPU,cpu,DEF,ACC,ALL. Suggested are GPU,CPU. For each setting to work there must be such an OpenCL device and properly installed. E.g.: If Nvidia driver is installed, GPU will succeed but CPU will fail. The AMD SDK kit is required for CPU via OpenCL.

Parameters

`devicetype`: string in ["cpu","gpu", "all", "acc"]

`useFp64`: boolean specifying if double precision will be used

`platformid`: integer

`devid`: integer

Return Value

OpenCL context on the selected device

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __init__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

12.4.2 Properties

Name	Description
Inherited from object	
__class__	

12.4.3 Class Variables

Name	Description
platforms	**Value:** `[NVIDIA CUDA, AMD Accelerated Parallel Processing, Intel(...`

13 Module pyFAI.peakPicker

Date: 23/12/2011

Author: J\xc3\xa9r\xc3\xb4me Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

13.1 Variables

Name	Description
__status__	**Value:** `'development'`
logger	**Value:** `logging.getLogger("pyFAI.peakPicker")`
TARGET_SIZE	**Value:** `1024`
__package__	**Value:** `'pyFAI'`

13.2 Class PeakPicker

object
└ **pyFAI.peakPicker.PeakPicker**

13.2.1 Methods

__init__(*self*, *strFilename*, *reconst*=`False`, *mask*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Parameters

`reconst`: shall negative values be reconstucted (wipe out problems with pilatus gaps)

Overrides: object.__init__

gui(*self*, *log*=`False`)

Parameters

`log`: show z in log scale

load(*self*, *filename*)

load a filename and plot data on the screen (if GUI)

display_points(*self*)

onclick(*self*, *event*)

readFloatFromKeyboard(*self*, *text*, *dictVar*)

Read float from the keyboard ....

Parameters

`text`: string to be displayed

`dictVar`: dict of this type: {1: [set_dist_min],3: [set_dist_min, set_dist_guess, set_dist_max]}

finish(*self*, *filename*=`None`)

Ask the 2theta values for the given points

contour(*self*, *data*)

Parameters

`data`:

massif_contour(*self*, *data*)

Parameters

`data`:

closeGUI(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.2.2 Properties

Name	Description
Inherited from object	
__class__	

13.3 Class ControlPoints

object
└ **pyFAI.peakPicker.ControlPoints**

This class contains a set of control points with (optionaly) their diffrection 2Theta angle

13.3.1 Methods

__init__(*self*, *filename*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__ extit(inherited documentation)

__repr__(*self*)

repr(x)

Overrides: object.__repr__ extit(inherited documentation)

__len__(*self*)

check(*self*)

check internal consistency of the class

reset(*self*)

remove all stored values and resets them to default

append(*self*, *points*, *angle*=`None`)

Parameters

`point`: list of points

`angle`: 2-theta angle in radians

append_2theta_deg(*self*, *points*, *angle*=`None`)

Parameters

`point`: list of points

`angle`: 2-theta angle in degrees

pop(*self*, *idx*=`None`)

Remove the set of points at given index (by default the last)

Parameters

`idx:` poistion of the point to remove

save(*self*, *filename*)

Save a set of control points to a file

Parameters

`filename:` name of the file

Return Value

None

load(*self*, *filename*)

load all control points from a file

getList(*self*)

Retrieve the list of control points suitable for geometry refinement

readAngleFromKeyboard(*self*)

Ask the 2theta values for the given points

setWavelength(*self*, *value*=`None`)

getWavelength(*self*)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.3.2 Properties

Name	Description
wavelength	
Inherited from object __class__	

13.4 Class Massif

object ─┐
pyFAI.peakPicker.Massif

A massif is defined as an area around a peak, it is used to find neighbouring peaks

13.4.1 Methods

__init__(*self*, *data*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__

nearest_peak(*self*, *x*)

@returns the coordinates of the nearest peak

calculate_massif(*self*, *x*)

defines a map of the massif around x and returns the mask

find_peaks(*self*, *x*, *nmax*=`200`, *annotate*=`None`, *massif_contour*=`None`, *stdout*=`sys.stdout`)

All in one function that finds a maximum from the given seed (x) then calculates the region extension and extract position of the neighboring peaks.

Parameters

- `x`: seed for the calculation, input coordinates
- `nmax`: maximum number of peak per region
- `annotate`: call back method taking number of points + coordinate as input.
- `massif_contour`: callback to show the contour of a massif with the given index.
- `stdout`: this is the file where output is written by default.

Return Value

list of peaks

initValleySize(*self*)

getValleySize(*self*)

setValleySize(*self*, *size*)

delValleySize(*self*)

getBinnedData(*self*)

@return binned data

getMedianData(*self*)

getBluredData(*self*)

getLabeledMassif(*self*, *pattern*=None)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

13.4.2 Properties

Name	Description
valley_size	Defines the minimum distance between two massifs
Inherited from object __class__	

14 Module pyFAI.reconstruct

14.1 Variables

Name	Description
__package__	**Value:** 'pyFAI'
__test__	**Value:** {}

15 Module pyFAI.refinment2D

Date: 23/08/2012

Author: J\xc3\xa9r\xc3\xb4me Kieffer

Contact: Jerome.Kieffer@ESRF.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

15.1 Variables

Name	Description
__status__	**Value:** `'beta'`
logger	**Value:** `logging.getLogger("pyFAI.refinment2D")`
__package__	**Value:** `'pyFAI'`

15.2 Class Refinment2D

object ─┐
pyFAI.refinment2D.Refinment2D

refine the parameters from image itself ...

15.2.1 Methods

__init__(*self*, *img*, *ai*=`None`)

x.__init__(...) initializes x; see x.__class__.__doc__ for signature

Overrides: object.__init__

get_shape(*self*)

reconstruct(*self*, *tth*, *I*)

Reconstruct a perfect image according to 2th / I given in input

Parameters

`tth`: 2 theta array

`I`: intensity array

diff_tth_X(*self*, *dx*=`0.1`)

diff_tth_tilt(*self*, *dx*=`0.1`)

diff_Fit2D(*self*, *axis*=`'all'`, *dx*=`0.1`)

scan_centerX(*self*, *width*=`1.0`, *points*=`10`)

scan_tilt(*self*, *width*=`1.0`, *points*=`10`)

scan_Fit2D(*self*, *width*=`1.0`, *points*=`10`, *axis*=`'tilt'`, *dx*=`0.1`)

Inherited from object

__delattr__(), __format__(), __getattribute__(), __hash__(), __new__(), __reduce__(), __reduce_ex__(), __repr__(), __setattr__(), __sizeof__(), __str__(), __subclasshook__()

15.2.2 Properties

Name	Description
shape	
Inherited from object __class__	

16 Module pyFAI.relabel

Date: 20120916

Author: Jerome Kieffer

Contact: Jerome.kieffer@esrf.fr

License: GPLv3+

16.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`
__status__	**Value:** `'stable'`
__test__	**Value:** {}

17 Module pyFAI.spline

This is piece of software aims to manipulate spline files for geometric corrections of the 2D detectors using cubic-spline

Author: J\xc3\xa9r\xc3\xb4me Kieffer

Contact: Jerome.Kieffer@esrf.eu

Copyright: European Synchrotron Radiation Facility, Grenoble, France

License: GPLv3+

17.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`

17.2 Class Spline

This class is a python representation of the spline file Those file represent cubic splines for 2D detector distortions and makes heavy use of fitpack (dierckx in netlib) — A Python-C wrapper to FITPACK (by P. Dierckx). FITPACK is a collection of FORTRAN programs for curve and surface fitting with splines and tensor product splines. See http://www.cs.kuleuven.ac.be/cwis/research or http://www.netlib.org/dierckx/index.html

17.2.1 Methods

__init__(*self*, *filename*=`None`)

this is the constructor of the Spline class, for

__repr__(*self*)

zeros(*self*, *xmin*=`0.0`, *ymin*=`0.0`, *xmax*=`2048.0`, *ymax*=`2048.0`, *pixSize*=`None`)

defines a spline file with no (zero) displacement.

Parameters

`xmin`: minimum coordinate in x, usually zero

(type=float)

`xmax`: maximum coordinate in x (+1) usually 2048

(type=float)

`ymin`: minimum coordinate in y, usually zero

(type=float)

`ymax`: maximum coordinate y (+1) usually 2048

(type=float)

zeros_like(*self*, *other*)

defines a spline file with no (zero) displacement with the same shape as the other one given.

Parameters

`other`: another Spline

(type=Spline)

read(*self*, *filename*)

read an ascii spline file from file

Parameters

`filename`: name of the file containing the cubic spline distortion file

(type=string)

comparison(*self*, *ref*, *verbose*=`False`)

Compares the current spline distortion with a reference

Parameters

`ref`: another spline file

Return Value

True or False depending if the splines are the same or not

spline2array(*self*, *timing*=`False`)

calculates the displacement matrix using fitpack bisplev(x, y, tck, dx = 0, dy = 0)

Evaluate a bivariate B-spline and its derivatives. Return a rank-2 array of spline function values (or spline derivative values) at points given by the cross-product of the rank-1 arrays x and y. In special cases, return an array or just a float if either x or y or both are floats.

splineFuncX(*self*, *x*, *y*)

calculates the displacement matrix using fitpack for the X direction

Parameters

`x`: numpy array repesenting the points in the x direction

`y`: numpy array repesenting the points in the y direction

Return Value

displacement matrix for the X direction

(type=numpy arrays)

splineFuncY(*self*, *x*, *y*)

calculates the displacement matrix using fitpack for the Y direction

Parameters

`x`: numpy array repesenting the points in the x direction

`y`: numpy array repesenting the points in the y direction

Return Value

displacement matrix for the Y direction

(type=numpy array)

array2spline(*self*, *smoothing*=`1000`, *timing*=`False`)

calculates the spline coefficents from the displacements matrix using fitpack

writeEDF(*self*, *basename*)

save the distortion matrices into a couple of files called basename-x.edf and basename-y.edf

write(*self*, *filename*)

save the cubic spline in an ascii file usable with Fit2D or SPD

Parameters

`filename`: name of the file containing the cubic spline distortion file

(type=string)

tilt(*self*, *center*=`(0.0, 0.0)`, *tiltAngle*=`0.0`, *tiltPlanRot*=`0.0`, *distanceSampleDetector*=`1.0`, *timing*=`False`)

The tilt method apply a virtual tilt on the detector, the point of tilt is given by the center

Parameters

`center`: position of the point of tilt, this point will not be moved.

(type=2tuple of floats)

`tiltAngle`: the value of the tilt in degrees

(type=float in the range [-90:+90] degrees)

`tiltPlanRot`: the rotation of the tilt plan with the Ox axis (0 deg for y axis invariant, 90 deg for x axis invariant)

(type=Float in the range [-180:180])

`distanceSampleDetector`: the distance from sample to detector in meter (along the beam, so distance from sample to center)

(type=float)

Return Value

tilted Spline instance

(type=Spline)

setPixelSize(*self*, *pixelSize*)

sets the size of the pixel from a 2-tuple of floats expressed in meters.

Parameters

`pixelSize`: *(type=2-tuple of float)*

getPixelSize(*self*)

Return Value

the size of the pixel from a 2D detector

(type=2-tuple of floats expressed in meter.)

bin(*self*, *binning*=`None`)

18 Module pyFAI.splitBBox

18.1 Variables

Name	Description
__package__	**Value:** `'pyFAI'`
__test__	**Value:** {}

19 Module pyFAI.splitBBoxLUT

19.1 Variables

Name	Description
__package__	**Value:** 'pyFAI'
__test__	**Value:** {}

20 Module pyFAI.splitPixel

20.1 Variables

Name	Description
__package__	**Value:** 'pyFAI'
__test__	**Value:** {}

21 Module pyFAI.utils

21.1 Functions

timeit(*func*)

---

gaussian_filter(*input*, *sigma*, *mode*=`'reflect'`, *cval*=`0.0`)

2-dimensional Gaussian filter implemented with FFTw

Parameters

`input`: input array to filter

(type=array-like)

`sigma`: standard deviation for Gaussian kernel. The standard deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in which case it is equal for all axes.

(type=scalar or sequence of scalars)

`mode`: {'reflect','constant','nearest','mirror', 'wrap'}, optional The "mode" parameter determines how the array borders are handled, where "cval" is the value when mode is equal to 'constant'. Default is 'reflect'

`cval`: scalar, optional Value to fill past edges of input if "mode" is 'constant'. Default is 0.0

---

expand(*input*, *sigma*, *mode*=`'constant'`, *cval*=`0.0`)

Expand array a with its reflection on boundaries

Parameters

`a`: 2D array

`sigma`: float or 2-tuple of floats

`mode`: "constant","nearest" or "reflect"

`cval`: filling value used for constant, 0.0 by default

relabel(*label*, *data*, *blured*, *max_size*=`None`)

Relabel limits the number of region in the label array. They are ranked relatively to their max(I0)-max(blur(I0)

Parameters

- `label:` a label array coming out of scipy.ndimage.measurement.label
- `data:` an array containing the raw data
- `blured:` an array containing the blured data
- `max_size:` the max number of label wanted @return array like label

averageImages(*listImages*, *output*=`None`, *threshold*=`0.1`, *minimum*=`None`, *maximum*=`None`, *darks*=`None`, *flats*=`None`)

Takes a list of filenames and create an average frame discarding all saturated pixels.

Parameters

- `listImages:` list of string representing the filenames
- `output:` name of the optional output file
- `threshold:` what is the upper limit? all pixel > max*(1-threshold) are discareded.
- `minimum:` minimum valid value or True
- `maximum:` maximum valid value
- `darks:` list of dark current images for subtraction
- `flats:` list of flat field images for division

boundingBox(*img*)

Tries to guess the bounding box around a valid massif

Parameters

- `img:` 2D array like

Return Value

4-typle (d0_min, d1_min, d0_max, d1_max)

removeSaturatedPixel(*ds*, *threshold*=`0.1`, *minimum*=`None`, *maximum*=`None`)

Parameters

`ds`: a dataset as ndarray

`threshold`: what is the upper limit? all pixel > max*(1-threshold) are discareded.

`minimum`: minumum valid value (or True for auto-guess)

`maximum`: maximum valid value

Return Value

another dataset

binning(*inputArray*, *binsize*)

Parameters

`inputArray`: input ndarray

`binsize`: int or 2-tuple representing the size of the binning

Return Value

binned input ndarray

unBinning(*binnedArray*, *binsize*)

Parameters

`binnedArray`: input ndarray

`binsize`: 2-tuple representing the size of the binning

Return Value

unBinned input ndarray

21.2 Variables

Name	Description
logger	**Value:** `logging.getLogger("pyFAI.utils")`
timelog	**Value:** `logging.getLogger("pyFAI.timeit")`
__package__	**Value:** `'pyFAI'`

Index