xrayutilities Documentation

Release 0.99

Dominik Kriegner **Eugen Wintersberger**

March 20, 2013

CONTENTS

If you look for downloading the package go here. Installation instructions you find further down Installation.

CHAPTER

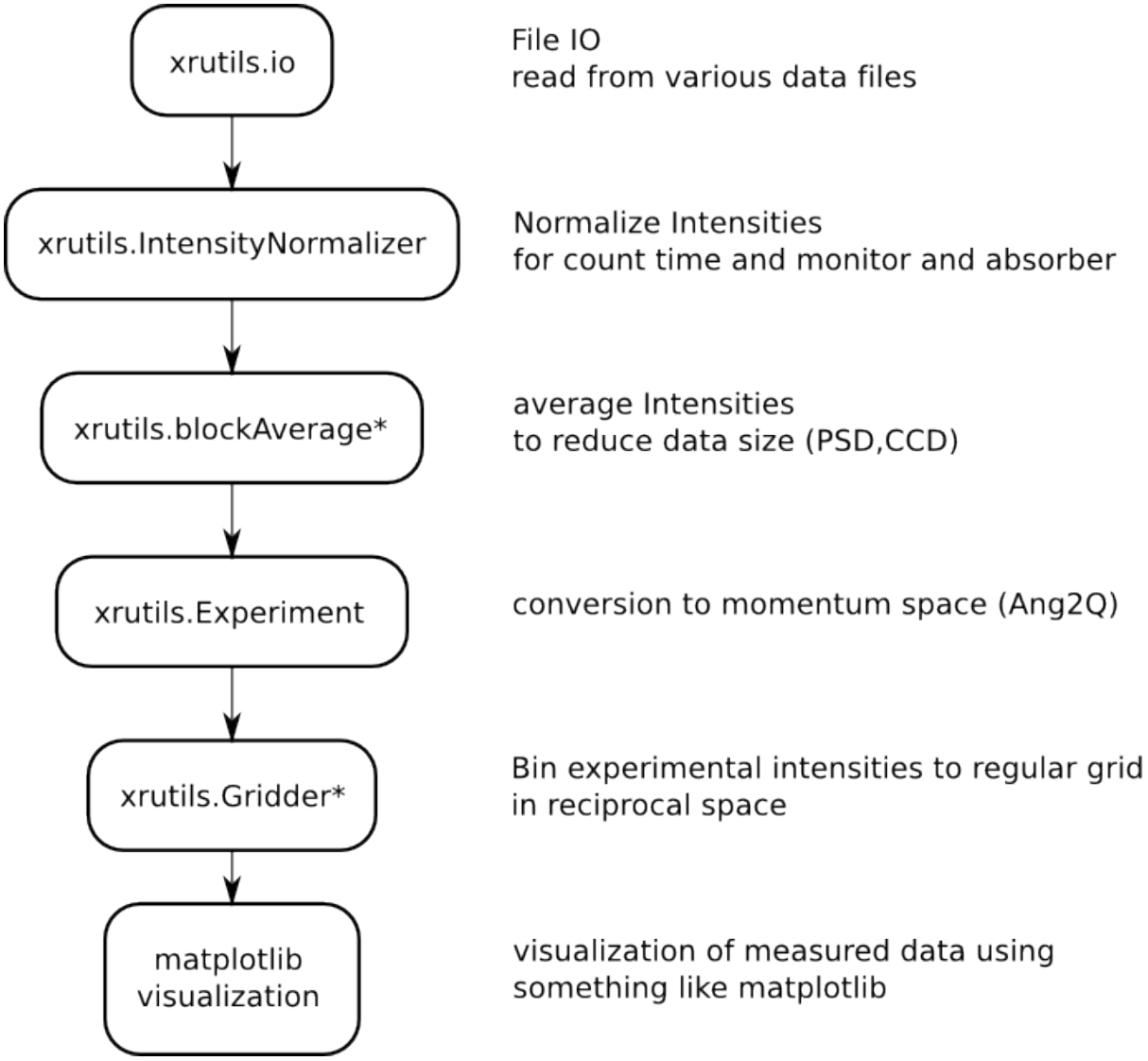
ONE

INTRODUCTION

xrayutilities is a collection of scripts used to analyze x-ray diffraction data. It consists of a python package and several routines coded in C. It especially useful for the reciprocal space conversion of diffraction data taken with linear and area detectors.

In the following two concepts of usage for the *xrayutilities* package will be described. First one should get a brief idea of how to analyze x-ray diffraction data with *xrayutilities*. After that the concept of how angular coordinates of Bragg reflections are calculated is presented.

1.1 Concept of usage



xrayutilities provides a set of functions to read experimental data from various data file formats. All of them are gathered in the *io Package*. After reading data with a function from the io-submodule the data might be need to be corrected for monitor counts and or absorber corrected. A special set of functions is provided to perform this for point and linear detectors.

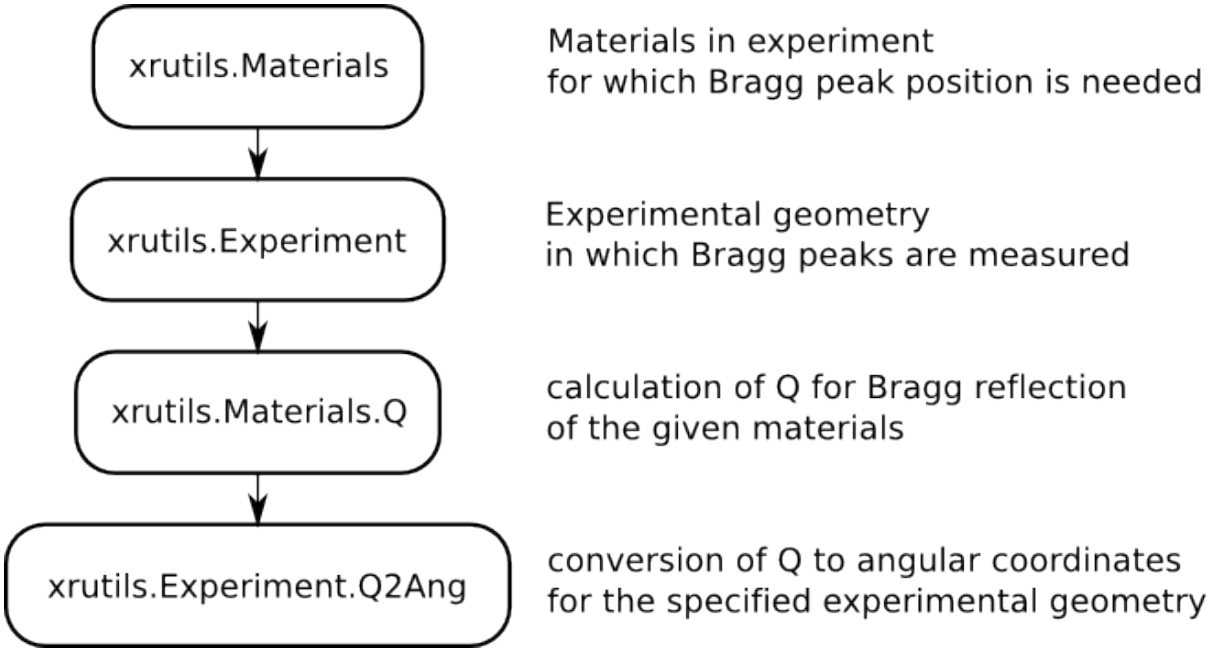
Since the amount of data taken with modern detectors often is too large to be able to work with them properly a function for reducing the data from linear and are detectors are provided. They use block-averaging to reduce the amount of data. Use those carefully not to loose the featured you are interested in your measurements.

After the pre-treatment of the data the core part of the package is the transformation of the angular data to reciprocal space. This is done as described in more detail below using the *experiment Module*. The classes provided within the experiment module provide routines to help performing X-ray diffraction experiments. This includes methods to calculate the diffraction angles (described below) needed to align samples and to convert data between angular and recip- rocal space. The conversion from angular to reciprocal space is implemented very general for various goniometer geometries. It is especially useful in combination with linear and area detectors as described in (arxiv link) Users should in normal cases only need the initialized routines, which predefine a certain goniometer geometry like the popular four-cirlce and six-circle geometry.

After the conversion to reciprocal space in order to visualize the data it is convenient to transform them to a regular grid in reciprocal space. For this purpose in *xrayutilities* the *gridder Module* is included. For the visualization of the data in reciprocal space the usage of matplotlib is recommended.

A practical example showing the usage is given below.

1.2 Angle calculation using the material classes



Calculation of angles needed to align Bragg reflections in various diffraction geometries is done using the Materials defined in the *materials Package*. This package provides a set of classes to describe crystal lattices and materials. Once such a material is properly defined one can calculate its properties, which includes the reciprocal lattice points, optical properties like the refractive index, the structure factor (including the atomic scattering factor) and the complex polarizability. These atomic properties are extracted from a database included in *xrayutilities*.

Using such a material and an experimental class from the *experiment Module* describing the experimental setup the needed diffraction angles can be calculated for certain coplanar diffraction (high, low incidence), grazing incidence diffraction and also special non-coplanar diffraction geometries.

1.3 hello world

A first example with step by step explanation is shown in the following. It showcases the use of *xrayutilities* to calculate angles and read a scan recorded with a linear detector from spec-file and plots the result as reciprocal space map using matplotlib.

```
"""
Example script to show how to use xrayutilities to read and plot
reciprocal space map scans from a spec file created at the ESRF/ID10B

for details about the measurement see:
    D Kriegner et al. Nanotechnology 22 425704 (2011)
    http://dx.doi.org/10.1088/0957-4484/22/42/425704
"""

import numpy
import matplotlib.pyplot as plt
import xrutils as xu
import os

# global setting for the experiment
sample = "test" # sample name used also as file name for the data file
```

```
energy = 8042.5 # x-ray energy in eV
center_ch = 715.9 # center channel of the linear detector used in the experiment
chpdeg = 345.28 # channels per degree of the linear detector (mounted along z direction, which co
roi=[100,1340] # region of interest of the detector
nchannel = 1500 # number of channels of the detector

# intensity normalizer function responsible for count time and absorber correction
normalizer_detcorr = xu.IntensityNormalizer("MCA",mon="Monitor",time="Seconds",absfun=lambda d: d

# substrate material used for Bragg peak calculation to correct for experimental offsets
InP = xu.materials.InP

# initialize experimental class to specify the reference directions of your crystal
# 11-2: inplane reference
# 111: surface normal
hxrd = xu.HXRD(InP.Q(1,1,-2),InP.Q(1,1,1),en=energy)

# configure linear detector
# detector direction + parameters need to be given
hxrd.Ang2Q.init_linear('z-',center_ch,nchannel,chpdeg=chpdeg,roi=roi)

# read spec file and save to HDF5-file
# since reading is much faster from HDF5 once the data are transformed
h5file = os.path.join("data",sample+".h5")
try: s  # try if spec file object already exist from a previous run of the script ("run -i" in ip
except NameError: s = xu.io.SPECFile(sample+".spec",path="data")
else: s.Update()
s.Save2HDF5(h5file)

################################
# InP (333) reciprocal space map
omalign = 43.0529 # experimental aligned values
ttalign = 86.0733
[omnominal,dummy,dummy,ttnominal] = hxrd.Q2Ang(InP.Q(3,3,3)) # nominal values of the substrate pe

# read the data from the HDF5 file (scan number:36, names of motors in spec file: omega= sample r
[om,tt],MAP = xu.io.geth5_scan(h5file,36,'omega','gamma')
# normalize the intensity values (absorber and count time corrections)
psdraw = normalizer_detcorr(MAP)
# remove unusable detector channels/regions (no averaging of detector channels)
psd = xu.blockAveragePSD(psdraw, 1, roi=roi)

# convert angular coordinates to reciprocal space + correct for offsets
[qx,qy,qz] = hxrd.Ang2Q.linear(om,tt,delta=[omalign-omnominal, ttalign-ttnominal])

# calculate data on a regular grid of 200x201 points
gridder = xu.Gridder2D(200,201)
gridder(qy,qz,psd)
# maplog function limits the shown dynamic range to 8 orders of magnitude from the maxium
INT = xu.maplog(gridder.gdata.transpose(),8.,0)

# plot the intensity as contour plot using matplotlib
plt.figure()
cf = plt.contourf(gridder.xaxis, gridder.yaxis,INT,100,extend='min')
plt.xlabel(r'$Q_{[11\bar2]}$ ($\AA^{-1}$)')
plt.ylabel(r'$Q_{[\bar1\bar1\bar1]}$ ($\AA^{-1}$)')
cb = plt.colorbar(cf)
cb.set_label(r"$\log($Int$)$ (cps)")
```

More such examples can be found on the *Examples* page.

CHAPTER

TWO

XRUTILS PYTHON PACKAGE

xrutils is a package for assisting with x-ray diffraction experiments

It helps with planning experiments as well as analyzing the data.

Authors Dominik Kriegner and Eugen Wintersberger

for more details see the full *API-documentation*

CHAPTER

THREE

INSTALLATION

3.1 Express instructions

- install the dependencies (Windows: pythonxy,‘SCons <http://www.scons.org>‘_; Linux/Unix: see below for dependencies).
- download *xrayutilities* from here or use git to check out the latest version.
- open a command line and navigate to the downloaded sources and execute:

```
> scons install
```

which will install *xrayutilities* to the default directory. It should be possible to use it (*import xrutils*) from now on in python scripts.

Note: The python package of *xrayutilities* is called ‘’xrutils’‘

3.2 Detailed instructions

Installing *xrayutilities* is a two steps process

- install required C libraries and Python modules
- build and install the *xrayutilities* C library and Python module

All steps are described in detail below and are performed by the SCons installer. The package can be installed on Linux, Mac OS X and Microsoft Windows, however it is mostly tested on Linux/Unix platforms. Please inform one of the authors in case the installation fails!

3.3 Required third party software

To keep the coding effort as small as possible *xrayutilities* depends on a large number of third party libraries and Python modules.

The needed dependencies are:

- **GCC** Gnu Compiler Collection or any compatible C compiler. On windows you most probably should use MinGW or CygWin. Others might work but are untested.
- **HDF5** a versatile binary data format (library is implemented in C). Although the library is not called directly, it is needed by the pytables Python module (see below).
- **Python** the scripting language in which most of *xrayutilities* code is written in.
- **Scons** a pythonic autotools/make replacement used for building the C library.

- **git** a version control system used to keep track on the *xrayutilities* development. (only needed for development)

Additionally, the following Python modules are needed in order to make *xrayutilities* work as intended:

- **Numpy** a Python module providing numerical array objects
- **Scipy** a Python module providing standard numerical routines, which is heavily using numpy arrays
- **Python-Tables** a powerful Python interface to HDF5.
- **Matplotlib** a Python module for high quality 1D and 2D plotting (optionally)
- **IPython** although not a dependency of *xrayutilities* the IPython shell is perfectly suited for the interactive use of the *xrayutilities* python package.

After installing all required packages you can continue with installing and building the C library.

3.4 Building and installing the library and python package

xrayutilities uses the SCons build system to compile the C components of the system. You can build the library simply by typing

```
>scons
```

in the root directory of the source distribution. To build using debug flags ({tt -g -O0}) type

```
>scons debug=1
```

instead. After building, the library and python package are installed by

```
>scons install --prefix=<install path>
```

The library is installed in *<install path>/lib*. Installation of the Python module is done via the *distutils* package (called by SCons automatically). The *–prefix* option sets the root directory for the installation. If it is omitted the libary is installed under /usr/lib/ on Unix systems or in the Python installation directory on Windows.

3.5 Setup of the Python package

You need to make your Python installation aware of where to look for the module. This is usually only needed when installing in non-standard *<install path>* locations. For this case append the installation directory to your *PYTHONPATH* environment variable by

```
>export PYTHONPATH=$PYTHONPATH:<local install path>/lib64/python2.7/site-packages
```

on a Unix/Linux terminal. Or, to make this configuration persistent append this line to your local *.bashrc* file in your home directory. On MS Windows you would like to create a environment variable in the system preferences under system in the advanced tab (Using pythonxy this is done automatically). Be sure to use the correct directory which might be similar to

```
<local install path>/Lib/site-packages
```

on Windows systems.

3.6 Notes for installing on Windows

Since there is no packages manager on Windows the packages need to be installed manual (including all the dependecies) or a pre-packed solution needs to be used. We strongly suggest to use the pyhton(x,y) python distribution, which includes already most of the needed dependencies for installing *xrayutilities*.

When using python(x,y) you only have to install SCons in addition (download the latest version from www.scons.org). All other dependencies are available as plugins to python(x,y) and are installed by default anyhow. The setup of the environment variables is also done by the python(x,y) installation. One can proceed with the installation of *xrayutilities* directly!

In case you want to do it the hard way install all of the following (versions in brackets indicate the tested set of versions by t

- MinGW (0.4alpha)
- Python (2.7.2)
- scons (2.1.0)
- numpy (1.6.1)
- scipy (0.10.1)
- numexpr (1.4.2) needed for pytables
- pytables (2.3.1)
- matplotlib (1.1.0)
- ipython (0.12)

It is suggested to add the MinGW binary directory, as well as the Python and Python-scripts directory to the Path environment variable as described above! Installation is done as described above.

CHAPTER

FOUR

EXAMPLES AND API-DOCUMENTATION

4.1 Examples

In the following a few code-snippets are shown which should help you getting started with *xrayutilities*. Not all of the codes shown in the following will be run-able as stand-alone script. For fully running scripts look in the `examples` directory in the download found here.

4.1.1 Reading data from data files

The `io` submodule provides classes for reading x-ray diffraction data in various formats. In the following few examples are given.

Reading SPEC files

Working with spec files in *xrayutilities* can be done in two distinct ways.

1. parsing the spec file for scan headers; and parsing the data only when needed
2. parsing the spec file for scan headers; parsing all data and dump them to an HDF5 file; reading the data from the HDF5 file.

Both methods have their pros and cons. For example when you parse the spec-files over a network connection you need to re-read the data again over the network if using method 1) whereas you can dump them to a local file with method 2). But you will parse data of the complete file while dumping it to the HDF5 file.

Both methods work incremental, so they do not start at the beginning of the file when you reread it, but start from the last position they were reading and work with files including data from linear detectors.

An working example for both methods is given in the following.:

```
import tables
import xrutils as xu
import os

# open spec file or use open SPECfile instance
try: s
except NameError:
    s = xu.io.SPECFile("sample_name.spec",path="./specdir")

# method (1)
scan10 = s[9] # Returns a SPECScan class, note 9 because the list starts at 0
scan10.ReadData()
scan10data = scan10.data

```

```
# method (2)
h5file = os.path.join("h5dir","h5file.h5")
s.Save2HDF5(h5file) # save content of SPEC file to HDF5 file
# read data from HDF5 file
[angle1,angle2],scan10data = xu.io.geth5_scan(h5file,[10], "motorname1", "motorname2")
```

See Also:

the fully working example *hello world*

In the following it is shown how to re-parsing the SPEC file for new scans and reread the scans (1) or update the HDF5 file(2)

```
s.Update() # reparse for new scans in open SPECFile instance

# reread data method (1)
scan10 = s[9] # Returns a SPECScan class
scan10.ReadData()
scan10data = scan10.data

# reread data method (2)
s.Save2HDF5(h5) # save content of SPEC file to HDF5 file
# read data from HDF5 file
[angle1,angle2],scan10data = xu.io.geth5_scan(h5file,[10], "motorname1", "motorname2")
```

Reading EDF files

EDF files are mostly used to store CCD frames at ESRF recorded from various different detectors. This format is therefore used in combination with SPEC files. In an example the EDFFile class is used to parse the data from EDF files and store them to an HDF5 file. HDF5 if perfectly suited because it can handle large amount of data and compression.:

```
import tables
import xrutils as xu
import numpy

specfile = "specfile.spec"
h5file = "h5file.h5"
h5 = tables.openFile(h5file,mode='a')

s = xu.io.SPECFile(specfile,path=specdir)
s.Save2HDF5(h5) # save to hdf5 file

# read ccd frames from EDF files
for i in range(1,1000,1):
    efile = "edfdir/sample_%04d.edf" %i
    e = xu.io.edf.EDFFile(efile,path=specdir)
    e.ReadData()
    g5 = h5.createGroup(h5.root,"frelon_%04d" %i)
    e.Save2HDF5(h5,group=g5)

h5.close()
```

See Also:

the fully working example provided in the `examples` directory perfectly suited for reading data from beamline ID01

Other formats

Other formats which can be read include

- files recorded from Panalytical diffractometers in the `.xrdml` format.
- files produces by the experimental control software at Hasylab/Desy (spectra).
- ccd images in the tiff file format produced by RoperScientific CCD cameras and Perkin Elmer detectors.
- files from recorded by Seifert diffractometer control software (`.nja`)
- basic support is also provided for reading of `cif` files from structure database to extract unit cell parameters

See the `examples` directory for more information and working example scripts.

4.1.2 Angle calculation using `experiment` and `material` classes

Methods for high angle x-ray diffraction experiments. Mostly for experiments performed in coplanar scattering geometry. An example will be given for the calculation of the position of Bragg reflections.

```
1  import xrutils as xu
2  Si = xu.materials.Si  # load material from materials submodule
3
4  # initialize experimental class with directions from experiment
5  hxrd = xu.HXRD(Si.Q(1,1,-2),Si.Q(1,1,1))
6  # calculate angles of Bragg reflections and print them to the screen
7  om,chi,phi,tt = hxrd.Q2Ang(Si.Q(1,1,1))
8  print("Si (111)")
9  print("om,tt: %8.3f %8.3f" %(om,tt))
10 om,chi,phi,tt = hxrd.Q2Ang(Si.Q(2,2,4))
11 print("Si (224)")
12 print("om,tt: %8.3f %8.3f" %(om,tt))
```

Note that on line 5 the `HXRD` class is initialized without specifying the energy used in the experiment. It will use the default energy stored in the configuration file, which defaults to CuK α_1.

One could also call:

```
hxrd = xu.HXRD(Si.Q(1,1,-2),Si.Q(1,1,1),en=10000) # energy in eV
```

to specify the energy explicitly. The `HXRD` class by default describes a four-circle goniometer as described in more detail here.

Similar functions exist for other experimental geometries. For grazing incidence diffraction one might use:

```
gid = xu.GID(Si.Q(1,-1,0),Si.Q(0,0,1))
# calculate angles and print them to the screen
(alphai,azimuth,tt,beta) = gid.Q2Ang(Si.Q(2,-2,0))
print("azimuth,tt: %8.3f %8.3f" %(azimuth,tt))
```

There are two implementations for GID experiments. Both describe 2S+2D diffractometers. They differ by the order of the detector circles. One describes a setup as available at ID10B/ESRF.

There exists also a powder diffraction class, which is able to convert powder scans from angular to reciprocal space and furthermore powder scans of materials can be simulated in a very primitive way, which should only be used to get an idea of the peak positions expected from a certain material.

```
1  import xrutils as xu
2  import matplotlib.pyplot as plt
3
4  energy = (2*8048 + 8028)/3. # copper k alpha 1,2
5
6  # creating Indium powder
7  In_powder = xu.Powder(xu.materials.In,en=energy)
8  # calculating the reflection strength for the powder
9  In_powder.PowderIntensity()
10
11 # convoluting the peaks with a gaussian in q-space
```

```
peak_width = 0.01 # in q-space
resolution = 0.0005 # resolution in q-space
In_th,In_int = In_powder.Convolute(resolution,peak_width)

plt.figure()
plt.xlabel(r"2Theta (deg)"); plt.ylabel(r"Intensity")
# plot the convoluted signal
plt.plot(In_th*2,In_int/In_int.max(),'k-',label="Indium powder convolution")
# plot each peak in a bar plot
plt.bar(In_powder.ang*2, In_powder.data/In_powder.data.max(), width=0.3, bottom=0, linewidth=0, co

plt.legend(); plt.set_xlim(15,100); plt.grid()
```

One can also print the peak positions and other informations of a powder by

```
>>> print In_powder
   Powder diffraction object
   -------------------------
   Material: In
   Lattice:
   a1 = (3.252300 0.000000 0.000000), 3.252300
   a2 = (0.000000 3.252300 0.000000), 3.252300
   a3 = (0.000000 0.000000 4.946100), 4.946100
   alpha = 90.000000, beta = 90.000000, gamma = 90.000000
   Lattice base:
   Base point 0: In (49) (0.000000 0.000000 0.000000) occ=1.00 b=0.00
   Base point 1: In (49) (0.500000 0.500000 0.500000) occ=1.00 b=0.00
   Reflections:
   --------------
         h k l     |    tth    |    |Q|    |    Int     |   Int (%)
      ---------------------------------------------------------------
       [-1, 0, -1]    32.9611      2.312       217.75      100.00
        [0, 0, -2]    36.3267      2.541        41.80       19.20
       [-1, -1, 0]    39.1721      2.732        67.72       31.10
      [-1, -1, -2]    54.4859      3.731        50.75       23.31
      ....
```

4.1.3 Using the `material` class

xrayutilities provides a set of python classes to describe crystal lattices and materials.

Examples show how to define a new material by defining its lattice and deriving a new material, furthermore materials can be used to calculate the structure factor of a Bragg reflection for an specific energy or the energy dependency of its structure factor for anomalous scattering. Data for this are taken from a database which is included in the download.

First defining a new material from scratch is shown. This consists of an lattice with base and the type of atoms with elastic constants of the material:

```
import xrutils as xu

# defining a ZincBlendeLattice with two types of atoms and lattice constant a
def ZincBlendeLattice(aa,ab,a):
    #create lattice base
    lb = xu.materials.LatticeBase()
    lb.append(aa,[0,0,0])
    lb.append(aa,[0.5,0.5,0])
    lb.append(aa,[0.5,0,0.5])
    lb.append(aa,[0,0.5,0.5])
    lb.append(ab,[0.25,0.25,0.25])
    lb.append(ab,[0.75,0.75,0.25])
    lb.append(ab,[0.75,0.25,0.75])
```

```
    lb.append(ab,[0.25,0.75,0.75])

    #create lattice vectors
    a1 = [a,0,0]
    a2 = [0,a,0]
    a3 = [0,0,a]

    l = xu.materials.Lattice(a1,a2,a3,base=lb)
    return l

# defining InP, no elastic properties are given,
# helper functions exist to create the (6,6) elastic tensor for cubic materials
atom_In = xu.materials.elements.In
atom_P = xu.materials.elements.P
elastictensor = xu.materials.CubicElasticTensor(10.11e+10,5.61e+10,4.56e+10)
InP  = xu.materials.Material("InP",ZincBlendeLattice(atom_In, atom_P ,5.8687), elastictensor)
```

InP is of course already included in the xu.materials module and can be loaded by:

```
InP = xu.materials.InP
```

like many other materials.

Using the material properties the calculation of the reflection strength of a Bragg reflection can be done as follows:

```
import xrutils as xu
import numpy

# defining material and experimental setup
InAs = xu.materials.InAs
energy= 8048 # eV

# calculate the structure factor for InAs (111) (222) (333)
hkllist = [[1,1,1],[2,2,2],[3,3,3]]
for hkl in hkllist:
    qvec = InAs.Q(hkl)
    F = InAs.StructureFactor(qvec,energy)
    print(" |F| = %8.3f" %numpy.abs(F))
```

Similar also the energy dependence of the structure factor can be determined:

```
import matplotlib.pyplot as plt

energy= numpy.linspace(500,20000,5000) # 500 - 20000 eV
F = InAs.StructureFactorForEnergy(InAs.Q(1,1,1),energy)

plt.figure(); plt.clf()
plt.plot(energy,F.real,'k-',label='Re(F)')
plt.plot(energy,F.imag,'r-',label='Imag(F)')
plt.xlabel("Energy (eV)"); plt.ylabel("F"); plt.legend()
```

It is also possible to calculate the components of the structure factor of atoms, which may be needed for input into XRD simulations.:

```
# f = f0(|Q|) + f1(en) + j * f2(en)
import xrutils as xu
import numpy

Fe = xu.materials.elements.Fe # iron atom
Q = numpy.array([0,0,1.9],dtype=numpy.double)
en = 10000 # energy in eV

print "Iron (Fe): E: %9.1f eV" % en
print "f0: %8.4g" % Fe.f0(numpy.linalg.norm(Q))
```

```
print "f1: %8.4g" % Fe.f1(en)
print "f2: %8.4g" % Fe.f2(en)
```

4.2 API-documentation

4.2.1 xrutils Package

xrutils is a package for assisting with x-ray diffraction experiments

It helps with planning experiments as well as analyzing the data.

Authors Dominik Kriegner and Eugen Wintersberger

config Module

module to parse xrutils user-specific config file the parsed values are provide as global constants for the use in other parts of xrutils. The config file with the default constants is found in the python installation path of xrutils. It is however not recommended to change things there, instead the user-specific config file ~/.xrutils.conf or the local xrutils.conf file should be used.

exception Module

xrutils derives its own exceptions which are raised upon wrong input when calling one of xrutils functions. none of the pre-defined exceptions is made for that purpose.

exception xrutils.exception.**InputError**(*msg*)

Bases: exceptions.Exception

Exception raised for errors in the input. Either wrong datatype not handled by TypeError or missing mandatory keyword argument (Note that the obligation to give keyword arguments might depend on the value of the arguments itself)

Attibutesexpr – input expression in which the error occurred :msg: – explanation of the error

experiment Module

module helping with planning and analyzing experiments

various classes are provided for

* describing experiments * calculating angular coordinates of Bragg reflections * converting angular coordinates to Q-space and vice versa * simulating powder diffraction patterns for materials

class xrutils.experiment.**Experiment**(*ipdir*, *ndir*, ***keyargs*)

Bases: object

base class for describing experiments users should use the derived classes: HXRD, GID, Powder

Ang2HKL(**args*, ***kwargs*)

angular to (h,k,l) space conversion. It will set the UB argument to Ang2Q and pass all other parameters unchanged. See Ang2Q for description of the rest of the arguments.

Parameters

****kwargs: optional keyword arguments**

Breciprocal space conversion matrix of a Material. you can specify the matrix B (default identiy matrix) shape needs to be (3,3)

matMaterial object to use to obtain a B matrix (e.g. xu.materials.Si) can be used as alternative to the B keyword argument B is favored in case both are given

Uorientation matrix U can be given if none is given the orientation defined in the Experiment class is used.

dettypedetector type: one of ('point', 'linear', 'area') decides which routine of Ang2Q to call default 'point'

Returns

H K L coordinates as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input (*args)

Q2Ang(*qvec*)

TiltAngle(*q*, *deg=True*)

TiltAngle(q,deg=True): Return the angle between a q-space position and the surface normal.

Parameters

qlist or numpy array with the reciprocal space position

optional keyword arguments:

degTrue/False whether the return value should be in degree or radians :(default: True)

Transform(*v*, ***kwargs*)

transforms a vector, matrix or tensor of rank 4 (e.g. elasticity tensor) to the coordinate frame of the Experiment class.

Parameters

vobject to transform, list or numpy array of shape (n,) (n,n), (n,n,n,n) where n is the rank of the transformation matrix

Returns

transformed object of the same shape as v

energy

wavelength

class xrutils.experiment.**GID**(*idir*, *ndir*, ***keyargs*)

Bases: xrutils.experiment.Experiment

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a four circle (alpha_i,azimuth,twotheta,beta) goniometer to help with GID experiments at the ROTATING ANODE. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*ai*, *phi*, *tt*, *beta*, ***kwargs*)

angular to momentum space conversion for a point detector. Also see help GID.Ang2Q for procedures which treat line and area detectors

Parameters

ai,phi,tt,betasample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than ai,phi,tt,beta - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang (*Q*, *trans=True*, *deg=True*, ***kwargs*)
calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

Qa list or numpy array of shape (3) with q-space vector components

optional keyword arguments:

transTrue/False apply coordinate transformation on Q

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

Returns

a numpy array of shape (4) with the four GID scattering angles which are [alpha_i,azimuth,twotheta,beta]

alpha_iincidence angle to surface (at the moment always 0)

azimuthsample rotation with respect to the inplane reference direction

twothetascattering angle

betaexit angle from surface (at the moment always 0)

class xrutils.experiment.**GID_ID10B** (*idir*, *ndir*, ***keyargs*)
Bases: xrutils.experiment.GID

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a four circle (theta,omega,delta,gamma) goniometer to help with GID experiments at ID10B / ESRF. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q (*th*, *om*, *delta*, *gamma*, ***kwargs*)
angular to momentum space conversion for a point detector. Also see help GID_ID10B.Ang2Q for procedures which treat line and area detectors

Parameters

th,om,delta,gammasample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than th,om,delta,gamma - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(*Q*, *trans=True*, *deg=True*, ***kwargs*)
calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

Qa list or numpy array of shape (3) with q-space vector components

optional keyword arguments:

transTrue/False apply coordinate transformation on Q

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

Returns

a numpy array of shape (4) with the four GID scattering angles which are (theta,omega,delta,gamma)

thetaincidence angle to surface (at the moment always 0)

omegasample rotation with respect to the inplane reference direction

deltaexit angle from surface (at the moment always 0)

gammascattering angle

class xrutils.experiment.**GISAXS**(*idir*, *ndir*, ***keyargs*)
Bases: xrutils.experiment.Experiment

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a three circle (alpha_i,twotheta,beta) goniometer to help with GISAXS experiments at the ROTATING ANODE. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*ai*, *tt*, *beta*, ***kwargs*)
angular to momentum space conversion for a point detector. Also see help GISAXS.Ang2Q for procedures which treat line and area detectors

Parameters

ai,tt,betasample and detector angles as numpy array, lists or Scalars must be given.
all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 3. used angles are than ai,tt,beta - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(*Q*, *trans=True*, *deg=True*, ***kwargs*)

class xrutils.experiment.**HXRD**(*idir*, *ndir*, ***keyargs*)

Bases: xrutils.experiment.Experiment

class describing high angle x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as helps with analyzing measured data

the class describes a two circle (omega,twotheta) goniometer to help with coplanar x-ray diffraction experiments. Nevertheless 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*om*, *tt*, ***kwargs*)

angular to momentum space conversion for a point detector. Also see help HXRD.Ang2Q for procedures which treat line and area detectors

Parameters

om,ttsample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 2. used angles are than om,tt - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(**Q*, ***keyargs*)

Convert a reciprocal space vector Q to COPLANAR scattering angles. The keyword argument trans determines whether Q should be transformed to the experimental coordinate frame or not.

Parameters

Qa list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz

optional keyword arguments:

transTrue/False apply coordinate transformation on Q (default True)

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

geometrydetermines the scattering geometry:

- "hi_lo" high incidence and low exit
- "lo_hi" low incidence and high exit
- **"real" general geometry with angles determined by q-coordinates (azimuth); this and upper geomet**
  - "realTilt" general geometry with angles determined by q-coordinates (tilt); returns [omega,chi,phi,twotheta]

defaultself.geometry

refracboolean to determine if refraction is taken into account :default: False if True then also a material must be given

matMaterial object; needed to obtain its optical properties for refraction correction, otherwise not used

full_outputboolean to determine if additional output is given to determine scattering angles more accurately in case refraction is set to True :default: False

fi,fdif refraction correction is applied one can optionally specify the facet through which the beam enters (fi) and exits (fd) fi, fd must be the surface normal vectors (not transformed & not necessarily normalized). If omitted the normal direction of the experiment is used.

Returns

a numpy array of shape (4) with four scattering angles which are [omega,chi,phi,twotheta]

omegaincidence angle with respect to surface

chisample tilt for the case of non-coplanar geometry

phisample azimuth with respect to inplane reference direction

twothetascattering angle

if full_output: a numpy array of shape (6) with five angles which are [omega,chi,phi,twotheta,psi_i,psi_d]

psi_ioffset of the incidence beam from the scattering plane due to refraction

pdi_doffset ot the diffracted beam from the scattering plane due to refraction

class xrutils.experiment.**NonCOP**(*idir*, *ndir*, ***keyargs*)

Bases: xrutils.experiment.Experiment

class describing high angle x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as helps with analyzing measured data for NON-COPLANAR measurements, where the tilt is used to align asymmetric peaks, like in the case of a polefigure measurement.

the class describes a four circle (omega,twotheta) goniometer to help with x-ray diffraction experiments. Linear and area detectors can be treated as described in "help self.Ang2Q"

Ang2Q(*om*, *chi*, *phi*, *tt*, ***kwargs*)

angular to momentum space conversion for a point detector. Also see help NonCOP.Ang2Q for procedures which treat line and area detectors

Parameters

om,chi,phi,ttsample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than om,chi,phi,tt - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(**Q*, ***keyargs*)

Convert a reciprocal space vector Q to NON-COPLANAR scattering angles. The keyword argument trans determines whether Q should be transformed to the experimental coordinate frame or not.

Parameters

Qa list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz

optional keyword arguments:

transTrue/False apply coordinate transformation on Q (default True)

degTrue/Flase (default True) determines if the angles are returned in radians or degree

Returns

a numpy array of shape (4) with four scattering angles which are [omega,chi,phi,twotheta]

omegasample rocking angle

chisample tilt

phisample azimuth

twothetascattering angle (detector)

class xrutils.experiment.**Powder**(*mat*, ***keyargs*)
Bases: xrutils.experiment.Experiment

Experimental class for powder diffraction This class is able to simulate a powder spectrum for the given material

Convolute(*stepwidth*, *width*, *min=0*, *max=None*)
Convolutes the intensity positions with Gaussians with width in momentum space of "width". returns array of angular positions with corresponding intensity

thetaarray with angular positions

intintensity at the positions ttheta

PowderIntensity(*tt_cutoff=180*)
Calculates the powder intensity and positions up to an angle of tt_cutoff (deg) and stores the result in:

dataarray with intensities

angangular position of intensities

qposreciprocal space position of intensities

Q2Ang(*qpos*, *deg=True*)
Converts reciprocal space values to theta angles

class xrutils.experiment.**QConversion**(*sampleAxis*, *detectorAxis*, *r_i*, ***kwargs*)
Bases: object

Class for the conversion of angular coordinates to momentum space for arbitrary goniometer geometries

the class is configured with the initialization and does provide three distinct routines for conversion to momentum space for

* point detector: point(...) or __call__() * linear detector: linear(...) * area detector: area(...)

linear() and area() can only be used after the init_linear() or init_area() routines were called

UB

area(**args*, ***kwargs*)
angular to momentum space conversion for a area detector the center pixel defined by the init_area routine must be in direction of self.r_i when detector angles are zero

the detector geometry must be initialized by the init_area(...) routine

Parameters

***args: sample and detector angles as numpy array, lists or Scalars**

in total len(self.sampleAxis)+len(detectorAxis) must be given always starting with the outer most circle all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: possible keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

roiregion of interest for the detector pixels; e.g. [100,900,200,800] :(default: self._area_roi)

Navnumber of channels to average to reduce data size e.g. [2,2] :(default: self._area_nav)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape ((*)*(self._area_roi[1]-self._area_roi[0]+1)*(self._area_roi[3]-self._area_roi[2]+1) , 3) were detectorDir1 is the fastest varing

detectorAxis

property handler for _detectorAxis

Returns

list of detector axis following the syntax /[xyz][+-]/

energy

init_area(*detectorDir1*, *detectorDir2*, *cch1*, *cch2*, *Nch1*, *Nch2*, *distance=None*, *pwidth1=None*, *pwidth2=None*, *chpdeg1=None*, *chpdeg2=None*, *detrot=0*, *tiltazimuth=0*, *tilt=0*, ***kwargs*)

initialization routine for area detectors detector direction as well as distance and pixel size or channels per degree must be given. Two separate pixel sizes and channels per degree for the two orthogonal directions can be given

Parameters

detectorDir1direction of the detector (along the pixel direction 1); e.g. 'z+' means higher pixel numbers at larger z positions

detectorDir2direction of the detector (along the pixel direction 2); e.g. 'x+'

cch1,2center pixel, in direction of self.r_i at zero detectorAngles

Nch1number of detector pixels along direction 1

Nch2number of detector pixels along direction 2

distancedistance of center pixel from center of rotation

pwidth1,2width of one pixel (same unit as distance)

chpdeg1,2channels per degree (only absolute value is relevant) sign determined through detectorDir1,2

detrotdetector rotation around primary beam direction

tiltazimuthdirection of the tilt vector in the detector plane (in degree)

tilttilt of the detector plane around an axis normal to the direction given by the tiltazimuth

---

Note: Note: Either distance and pwidth1,2 or chpdeg1,2 must be given !!

****kwargs: optional keyword arguments**

Navnumber of channels to average to reduce data size (default: [1,1])

roiregion of interest for the detector pixels; e.g. [100,900,200,800]

init_linear(*detectorDir*, *cch*, *Nchannel*, *distance=None*, *pixelwidth=None*, *chpdeg=None*, *tilt=0*, ***kwargs*)

initialization routine for linear detectors detector direction as well as distance and pixel size or channels per degree must be given.

Parameters

detectorDirdirection of the detector (along the pixel array); e.g. 'z+'

cchcenter channel, in direction of self.r_i at zero detectorAngles

Nchanneltotal number of detector channels

distancedistance of center channel from center of rotation

pixelwidthwidth of one pixel (same unit as distance)

chpdegchannels per degree (only absolute value is relevant) sign determined through detectorDir

!! Either distance and pixelwidth or chpdeg must be given !!

tilttilt of the detector axis from the detectorDir (in degree)

****kwargs: optional keyword arguments**

Navnumber of channels to average to reduce data size (default: 1)

roiregion of interest for the detector pixels; e.g. [100,900]

linear(**args*, ***kwargs*)

angular to momentum space conversion for a linear detector the cch of the detector must be in direction of self.r_i when detector angles are zero

the detector geometry must be initialized by the init_linear(...) routine

Parameters

***args: sample and detector angles as numpy array, lists or Scalars**

in total len(self.sampleAxis)+len(detectorAxis) must be given always starting with the outer most circle all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: possible keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

Navnumber of channels to average to reduce data size (default: self._linear_nav)

roiregion of interest for the detector pixels; e.g. [100,900] (default: self._linear_roi)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape ((*)*(self._linear_roi[1]-self._linear_roi[0]+1) , 3)

point (**args*, ***kwargs*)

angular to momentum space conversion for a point detector located in direction of self.r_i when detector angles are zero

Parameters

***args: sample and detector angles as numpy array, lists**

or Scalars in total len(self.sampleAxis)+len(detectorAxis) must be given, always starting with the outer most circle. all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree :(default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

sampleAxis

property handler for _sampleAxis

Returns

list of sample axis following the syntax /[xyzk][+-]/

wavelength

gridder Module

class xrutils.gridder.**Gridder**(***keyargs*)

Bases: object

KeepData(*bool*)

Normalize(*bool*)

SetChunkSize(*n*)

SetChunkUnit(*u*)

SetThreads(*n*)

class xrutils.gridder.**Gridder1D**(*nx*, ***keyargs*)
Bases: xrutils.gridder.Gridder

Clear()

data

xaxis

class xrutils.gridder.**Gridder2D**(*nx*, *ny*, ***keyargs*)
Bases: xrutils.gridder.Gridder

Clear()

SetResolution(*nx*, *ny*)

data

xaxis

xmatrix

yaxis

ymatrix

class xrutils.gridder.**Gridder3D**(*nx*, *ny*, *nz*, ***keyargs*)
Bases: xrutils.gridder.Gridder2D

SetResolution(*nx*, *ny*, *nz*)

zaxis

zmatrix

libxrayutils Module

this module uses the ctypes package to provide access to the functions implemented in the libxrayutils C library.

the functions provided by this module are low level. Users should use the derived functions in the corresponding submodules

normalize Module

module to provide functions that perform block averaging of intensity arrays to reduce the amount of data (mainly for PSD and CCD measurements

and

provide functions for normalizing intensities for

* count time * absorber (user-defined function) * monitor * flatfield correction

class xrutils.normalize.**IntensityNormalizer**(*det*, ***keyargs*)
Bases: object

generic class for correction of intensity (point detector,or MCA, single CCD frames) for count time and absorber factors the class must be supplied with a absorber correction function and works with data structures provided by xrutils.io classes or the corresponding objects from hdf5 files read by pytables

absfun
absfun property handler returns the costum correction function or None

avmon
av_mon property handler returns the value of the average monitor or None if average is calculated from the monitor field

darkfield
flatfield property handler returns the current set darkfield of the detector or None if not set

det
det property handler returns the detector field name

flatfield
flatfield property handler returns the current set flatfield of the detector or None if not set

mon
mon property handler returns the monitor field name or None if not set

time
time property handler returns the count time or the field name of the count time or None if time is not set

xrutils.normalize.**blockAverage1D**(*data*, *Nav*)
perform block average for 1D array/list of Scalar values all data are used. at the end of the array a smaller cell may be used by the averaging algorithm

Parameter

datadata which should be contracted (length N)

Navnumber of values which should be averaged

Returns

block averaged numpy array of data type numpy.double (length ceil(N/Nav))

xrutils.normalize.**blockAverage2D**(*data2d*, *Nav1*, *Nav2*, ***kwargs*)
perform a block average for 2D array of Scalar values all data are used therefore the margin cells may differ in size

Parameter

data2darray of 2D data shape (N,M)

Nav1,2a field of (Nav1 x Nav2) values is contracted

****kwargs: optional keyword argument**

roiregion of interest for the 2D array. e.g. [20,980,40,960] N = 980-20; M = 960-40

Returns

block averaged numpy array with type numpy.double with shape (ceil(N/Nav1), ceil(M/Nav2))

xrutils.normalize.**blockAveragePSD**(*psddata*, *Nav*, ***kwargs*)
perform a block average for serveral PSD spectra all data are used therefore the last cell used for averaging may differ in size

Parameter

psddataarray of 2D data shape (Nspectra,Nchannels)

Navnumber of channels which should be averaged

****kwargs: optional keyword argument**

roiregion of interest for the 2D array. e.g. [20,980] Nchannels = 980-20

Returns

block averaged psd spectra as numpy array with type numpy.double of shape (Nspectra , ceil(Nchannels/Nav))

utilities Module

xrutils utilities contains a conglomeration of useful functions which do not fit into one of the other files

xrutils.utilities.**maplog**(*inte*, *dynlow='config'*, *dynhigh='config'*, ***keyargs*)

clips values smaller and larger as the given bounds and returns the log10 of the input array. The bounds are given as exponent with base 10 with respect to the maximum in the input array. The function is implemented in analogy to J. Stangl's matlab implementation.

Parameters

inte numpy.array, values to be cut in range

dynlow 10^(-dynlow) will be the minimum cut off

dynhigh 10^(-dynhigh) will be the maximum cut off

optional keyword arguments (NOT IMPLEMENTED):

abslow 10^(abslow) will be taken as lower boundary

abshigh 10^(abshigh) will be taken as higher boundary

Returns

numpy.array of the same shape as inte, where values smaller/larger then 10^(-dynlow,dynhigh) were replaced by 10^(-dynlow,dynhigh)

Example

```
>>> lint = maplog(int,5,2)
```

utilities_noconf Module

xrutils utilities contains a conglomeration of useful functions this part of utilities does not need the config class

xrutils.utilities_noconf.**energy**(*en*)

convert common energy names to energies in eV

so far this works with CuKa1, CuKa2, CuKa12, CuKb, MoKa1

Parameter

en energy (scalar (energy in eV will be returned unchanged) or string with name of emission line)

Returns

energy in eV as float

xrutils.utilities_noconf.**lam2en**(*inp*)

converts the input energy in eV to a wavelength in Angstrom or the input wavelength in Angstrom to an energy in eV

Parameter

inp either an energy in eV or an wavelength in Angstrom

Returns

float, energy in eV or wavlength in Angstrom

Examples

```
>>> lambda = lam2en(8048)
>>> energy = lam2en(1.5406)
```

xrutils.utilities_noconf.**wavelength**(*wl*)
convert common energy names to energies in eV

so far this works with CuKa1, CuKa2, CuKa12, CuKb, MoKa1

Parameter

wlwavelength (scalar (wavelength in Angstrom will be returned unchanged) or string with name of emission line)

Returns

wavelength in Angstrom as float

Subpackages

analysis Package

xrutils.analysis is a package for assisting with the analysis of x-ray diffraction data, mainly reciprocal space maps

Routines for obtaining line cuts from gridded reciprocal space maps are offered, with the ability to integrate the intensity perpendicular to the line cut direction.

line_cuts Module

xrutils.analysis.line_cuts.**fwhm_exp**(*pos*, *data*)
function to determine the full width at half maximum value of experimental data. Please check the obtained value visually (noise influences the result)

Parameter

posposition of the data points

datadata values

Returns

fwhm value (single float)

xrutils.analysis.line_cuts.**get_omega_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)
extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_omega_scan_bounds_ang**(*omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of omega scan

Parameters

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qx,qzreciprocal space coordinates of the omega scan boundaries

Example

```
>>> qxb,qzb = get_omega_scan_bounds_ang(1.0,4.0,2.4,240,qrange=0.1)
```

xrutils.analysis.line_cuts.**get_omega_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *omrange*, *npoints*, ***kwargs*)

extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the omega scan should be extracted

qzcenterqz-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_qx_scan**(*qx*, *qz*, *intensity*, *qzpos*, ***kwargs*)

extract qx line scan at position qzpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qz

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

Returns

qx,qxintqx scan coordinates and intensities (bounds=False)

qx,qxint,(qxb,qyb)qx scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qxcut,qxcut_int = get_qx_scan(qx,qz,inten,5.0,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extract qz line scan at position qxpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qx

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan(qx,qz,inten,1.5,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan_int**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extracts a qz scan from a gridded reciprocal space map with integration along omega (sample rocking angle) or 2theta direction

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

angrangeintegration range in angular direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

intdirintegration direction 'omega': sample rocking angle (default) '2theta': scattering angle

Returns

qz,qzintqz scan coordinates and intensities (bounds=False)

qz,qzint,(qzb,qzb)qz scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qzcut,qzcut_int = get_qz_scan_int(qx,qz,inten,5.0,omrange=0.3)
```

xrutils.analysis.line_cuts.**get_radial_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_ang(qx,qz,intensity,32.0,64.0,30.0,800,omrange=0.2)
```

xrutils.analysis.line_cuts.**get_radial_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of radial scan

Parameters

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxrad,qzradreciprocal space boundaries of radial scan

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_radial_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the radial scan should be extracted

qzcenterqz-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_q(qx,qz,intensity,0.0,5.0,1.0,100,omrange=0.01)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_ang(qx,qz,intensity,32.0,64.0,4.0,400)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of 2theta scan

Parameters

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the 2theta scan to extract

npointsnumber of points of the 2theta scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxtt,qzttreciprocal space boundaries of 2theta scan (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_ttheta_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the 2theta scan should be extracted

qzcenterqz-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_q(qx,qz,intensity,0.0,4.0,4.4,440)
```

xrutils.analysis.line_cuts.**getindex**(*x*, *y*, *xgrid*, *ygrid*)

gives the indices of the point x,y in the grid given by xgrid ygrid xgrid,ygrid must be arrays containing equidistant points

Parameters

x,ycoordinates of the point of interest (float)

xgrid,ygridgrid coordinates in x and y direction (array)

Returns

ix,iyindex of the closest gridpoint (lower left) of the point (x,y)

line_cuts3d Module

xrutils.analysis.line_cuts3d.**get_qx_scan3d**(*gridder*, *qypos*, *qzpos*, ***kwargs*)

extract qx line scan at position y,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qypos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qx,qxintqx scan coordinates and intensities

Example

```
>>> qxcut,qxcut_int = get_qx_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qy_scan3d**(*gridder*, *qxpos*, *qzpos*, ***kwargs*)

extract qy line scan at position x,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qy,qyintqy scan coordinates and intensities

Example

```
>>> qycut,qycut_int = get_qy_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qz_scan3d**(*gridder*, *qxpos*, *qypos*, ***kwargs*)

extract qz line scan at position x,y from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qyposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**getindex3d**(*x*, *y*, *z*, *xgrid*, *ygrid*, *zgrid*)
gives the indices of the point x,y,z in the grid given by xgrid ygrid zgrid xgrid,ygrid,zgrid must be arrays containing equidistant points

Parameters

x,y,zcoordinates of the point of interest (float)

xgrid,ygrid,zgridgrid coordinates in x,y,z direction (array)

Returns

ix,iy,izindex of the closest gridpoint (lower left) of the point (x,y,z)

misc Module miscellaneous functions helpful in the analysis and experiment

xrutils.analysis.misc.**getangles**(*peak*, *sur*, *inp*)
calculates the chi and phi angles for a given peak

Parameter

peakarray which gives hkl for the peak of interest

surhkl of the surface

inpinplane reference peak or direction

Returns

[chi,phi] for the given peak on surface sur with inplane direction inp as reference

Example

To get the angles for the -224 peak on a 111 surface type[chi,phi] = getangles([-2,2,4],[1,1,1],[2,2,4])

sample_align Module functions to help with experimental alignment during experiments, especially for experiments with linear detectors

xrutils.analysis.sample_align.**area_detector_calib**(*angle1*, *angle2*, *ccdimages*, *detaxis*, *r_i*, *plot=True*, *cut_off=0.7*, *start=(0, 0, 0, 0)*, *fix=(False, False, False, False)*, *fig=None*, *wl=None*)
function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

parameters

angle1outer detector arm angle

angle2inner detector arm angle

ccdimagesimages of the ccd taken at the angles given above

detaxisdetector arm rotation axis :default: ['z+','y-']

r_iprimary beam direction [xyz][+-] default 'x+'

Keyword_arguments

plotflag to determine if results and intermediate results should be plotted :default: True

cut_offcut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%

startsequence of start values of the fit for parameters, which can not be estimated automatically these are: tiltazimuth,tilt,detector_rotation,outerangle_offset. By default (0,0,0,0) is used.

fixfix parameters of start (default: (False,False,False,False))

figmatplotlib figure used for plotting the error :default: None (creates own figure)

wlwavelength of the experiment in Angstrom (default: config.WAVELENGTH) value does not matter here and does only affect the scaling of the error

xrutils.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03*, *0.03)*, *plot=True*)

helper function to determine the Bragg peak position in a reciprocal space map used to obtain the position needed for correction of the data. the determination is done by fitting a two dimensional Gaussian (xrutils.math.Gauss2d)

PLEASE ALWAYS CHECK THE RESULT CAREFULLY!

Parameter

om,ttangular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]

psdintensity values needed for fitting

omalignaligned omega value, used as first guess in the fit

ttalignaligned two theta values used as first guess in the fit these values are also used to set the range for the fit: the peak should be within +/-frangeAA^{-1} of those values

exphxrdexperiment class used for the conversion between angular and reciprocal space.

frangedata range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})

plotif True (default) function will plot the result of the fit in comparison with the measurement.

Returns

Omfit,ttfit,params,covariance fitted angular values, and the fit parameters (of the Gaussian) as well as their errors

xrutils.analysis.sample_align.**linear_detector_calib**(*angle*, *mca_spectra*, ***keyargs*)

function to calibrate the detector distance/channel per degrees for a straigt linear detector mounted on a detector arm

parameters

anglearray of angles in degree of measured detector spectra

mca_spectracorresponding detector spectra :(shape: (len(angle),Nchannels)

****keyargs passed to psd_chdeg function used for the modelling additional options:**

r_iprimary beam direction as vector [xyz][+-]; default: 'y+'

detaxisdetector rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

returns

L/pixelwidth*pi/180 ~= channel/degree, center_channel[, detector_tilt]

The function also prints out how a linear detector can be initialized using the results obtained from this calibration.

Note: Note: distance of the detector is given by: channel_width*channelperdegree/tan(radians(1))

---

xrutils.analysis.sample_align.**miscut_calc**(*phi*, *aomega*, *zeros=None*, *plot=True*, *omega0=None*)

function to calculate the miscut direction and miscut angle of a sample by fitting a sinusoidal function to the variation of the aligned omega values of more than two reflections. The function can also be used to fit reflectivity alignment values in various azimuths.

Parameters

phiazimuths in which the reflection was aligned (deg)

aomegaaligned omega values (deg)

zeros(optional) angles at which surface is parallel to the beam (deg). For the analysis the angles (aomega-zeros) are used.

plotflag to specify if a visualization of the fit is wanted. :default: True

omega0if specified the nominal value of the reflection is not included as fit parameter, but is fixed to the specified value. This value is MANDATORY if ONLY TWO AZIMUTHs are given.

Returns

[omega0,phi0,miscut]

list with fitted values for

omega0the omega value of the reflection should be close to the nominal one

phi0the azimuth in which the primary beam looks upstairs

miscutamplitude of the sinusoidal variation == miscut angle

xrutils.analysis.sample_align.**psd_chdeg**(*angles*, *channels*, *stdev=None*, *usetilt=False*, *plot=True*)

function to determine the channels per degree using a linear fit of the function nchannel = center_ch+chdeg*tan(angles) or the equivalent including a detector tilt

Parameters

anglesdetector angles for which the position of the beam was measured

channelsdetector channels where the beam was found

keyword arguments:

stdevstandard deviation of the beam position

plotflag to specify if a visualization of the fit should be done

usetiltwhether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: False)

Returns (L/pixelwidth*pi/180 ,centerch[,tilt]):

L/pixelwidth*pi/180 = channel/degree for large detector distance with L sample detector disctance, and pixelwidth the width of one detector channel

Centerch center channel of the detector

Tilt tilt of the detector from perpendicular to the beam

Note: Note: distance of the detector is given by: channelwidth*channelperdegree/tan(radians(1))

xrutils.analysis.sample_align.**psd_refl_align**(*primarybeam*, *angles*, *channels*, *plot=True*)

function which calculates the angle at which the sample is parallel to the beam from various angles and detector channels from the reflected beam. The function can be used during the half beam alignment with a linear detector.

Parameters

primarybeamprimary beam channel number

angleslist or numpy.array with angles

channelslist or numpy.array with corresponding detector channels

plotflag to specify if a visualization of the fit is wanted :default: True

Returns

omegaangle at which the sample is parallel to the beam

Example

```
>>> psd_refl_align(500,[0,0.1,0.2,0.3],[550,600,640,700])
```

io Package

cif Module

class xrutils.io.cif.**CIFFile**(*filename*)

Bases: object

class for parsing CIF (Crystallographic Information File) files. The class aims to provide an additional way of creating material classes instead of manual entering of the information the lattice constants and unit cell structure are parsed from the CIF file

Lattice()
returns a lattice object with the structure from the CIF file

Parse()
function to parse a CIF file. The function reads the space group symmetry operations and the basic atom positions as well as the lattice constants and unit cell angles

SymStruct()
function to obtain the list of different atom positions in the unit cell for the different types of atoms. The data are obtained from the data parsed from the CIF file.

edf Module

class xrutils.io.edf.**EDFDirectory**(*datapath*, ***keyargs*)

Bases: object

Parses a directory for EDF files, which can be stored to a HDF5 file for further usage

Save2HDF5(*h5*, ***keyargs*)
Save2HDF5(h5,**keyargs): Saves the data stored in the EDF files in the specified directory in a HDF5 file as a HDF5 arrays in a subgroup. By default the data is stored in a group given by the foldername - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data (default: pathname) :comp: activate compression - true by default

class xrutils.io.edf.**EDFFile**(*fname*, ***keyargs*)
Bases: object

ReadData()
Read the CCD data into the .data object this function is called by the initialization

Save2HDF5(*h5*, ***keyargs*)
Save2HDF5(h5,**keyargs): Saves the data stored in the EDF file in a HDF5 file as a HDF5 array. By default the data is stored in the root group of the HDF5 file - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data :comp: activate compression - true by default

imagereader Module

class xrutils.io.imagereader.**ImageReader**(*nop1*, *nop2*, *hdrlen=0*, *flatfield=None*, *darkfield=None*, *dtype=<type 'numpy.int16'>*, *byte_swap=False*)
Bases: object

parse CCD frames in the form of tiffs or binary data (*.bin) to numpy arrays. ignore the header since it seems to contain no useful data

The routine was tested so far with RoperScientific files with 4096x4096 pixels created at Hasylab Hamburg, which save an 16bit integer per point. Perkin Elmer images created at Hasylab Hamburg with 2048x2048 pixels.

readImage(*filename*)
read image file and correct for dark- and flatfield in case the necessary data are available.

returned data = ((image data)-(darkfield))/flatfield*average(flatfield)

Parameter

filenamefilename of the image to be read. so far only single filenames are supported. The data might be compressed. supported extensions: .tiff, .bin and .bin.xz

class xrutils.io.imagereader.**PerkinElmer**(*keyargs)
Bases: xrutils.io.imagereader.ImageReader

parse PerkinElmer CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 2048x2048 pixel images created at Hasylab Hamburg which save an 32bit float per point.

class xrutils.io.imagereader.**RoperCCD**(*keyargs)
Bases: xrutils.io.imagereader.ImageReader

parse RoperScientific CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 4096x4096 pixel images created at Hasylab Hamburg which save an 16bit integer per point.

panalytical_xml Module Panalytical XML (www.XRDML.com) data file parser

based on the native python xml.dom.minidom module. want to keep the number of dependancies as small as possible

class xrutils.io.panalytical_xml.**XRDMLFile**(*fname*)
Bases: object

class to handle XRDML data files. The class is supplied with a file name and uses the XRDMLScan class to parse the xrdMeasurement in the file

class xrutils.io.panalytical_xml.**XRDMLMeasurement**(*measurement*)
Bases: object

class to handle scans in a XRDML datafile

xrutils.io.panalytical_xml.**getOmPixcel**(*omraw*, *ttraw*)
function to reshape the Omega values into a form needed for further treatment with xrutils

xrutils.io.panalytical_xml.**getxrdml_map**(*filetemplate*, *scannrs=None*, *path='.'*, *roi=None*)
parses multiple XRDML file and concatenates the results for parsing the xrutils.io.XRDMLFile class is used. The function can be used for parsing maps measured with the PIXCel and point detector.

Parameter

filetemplatetemplate string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames

scannrsint or list of scan numbers

pathcommon path to the filenames

roiregion of interest for the PIXCel detector, for other measurements this i not usefull!

Returns

om,tt,psdas flattened numpy arrays

Example

```
>>> om,tt,psd = xrutils.io.getxrdml_map("samplename_%d.xrdml",[1,2],path="./data")
```

radicon Module python module for converting radicon data to HDF5

xrutils.io.radicon.**hst2hdf5**(*h5*, *hstfile*, *nofchannels*, ***keyargs*)
Converts a HST file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

hstfilename of the HST file

nofchannelsnumber of channels

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

hstpathpath where the HST file is located (default is the current working directory)

xrutils.io.radicon.**rad2hdf5**(*h5*, *rdcfile*, ***keyargs*)
Converts a RDC file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

rdcfilename of the RDC file

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

rdcpathpath where the RDC file is located (default is the current working directory)

xrutils.io.radicon.**selecthst**(*et_limit*, *mca_info*, *mca_array*)
Select historgrams form the complete set of recorded MCA data and stores it into a new numpy array. The selection is done due to a exposure time limit. Spectra below this limit are ignored.

required input arguments:

et_limitexposure time limit

mca_infopytables table with the exposure data

mca_arrayarray with all the MCA spectra

return value:a numpy array with the selected mca spectra of shape (hstnr,channels).

rotanode_alignment Module parser for the alignment log file of the rotating anode

class xrutils.io.rotanode_alignment.**RA_Alignment**(*filename*)
Bases: object

class to parse the data file created by the alignment routine (tpalign) at the rotating anode spec installation

this routine does an iterative alignment procedure and saves the center of mass values were it moves after each scan. It iterates between two different peaks and iteratively aligns at each peak between two different motors (om/chi at symmetric peaks, om/phi at asymmetric peaks)

Parse()
parser to read the alignment log and obtain the aligned values at every iteration.

get(*key*)

keys()
returns a list of keys for which aligned values were parsed

plot(*pname*)
function to plot the alignment history for a given peak

Parameters

pnamepeakname for which the alignment should be plotted

seifert Module a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two posibilities how the data is stored (depending on the use detector):

1. as a simple line scan (using the point detector)
2. as a map using the PSD

In the first case the data ist stored

class xrutils.io.seifert.**SeifertHeader**
Bases: object

save_h5_attribs(*obj*)

class xrutils.io.seifert.**SeifertMultiScan**(*filename*, *m_scan*, *m2*)
Bases: object

dump2hdf5(*h5*, **args*, ***keyargs*)
Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument "group" must be used.

required arguments:

h5a HDF5 file object

optional positional arguments:name for the intensity matrix name for the scan motor name for the second motor more then three parameters are ignored.

optional keyword arguments:

grouppath to the HDF5 group where to store the data

dump2mlab(*fname*, **args*)
Store the data in a matlab file.

parse()

class xrutils.io.seifert.**SeifertScan**(*filename*)
Bases: object

dump2h5(*h5*, **args*, ***keyargs*)
Save the data stored in the Seifert ASCII file to a HDF5 file.

required input arguments:

h5HDF5 file object

optional arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

optional keyword arguments:

groupHDF5 group object where to store the data.

dump2mlab(*fname*, **args*)
Save the data from a Seifert scan to a matlab file.

required input arugments:

fnamename of the matlab file

optional position arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

parse()

xrutils.io.seifert.**repair_key**(*key*)
Repair a key string in the sense that the string is changed in a way that it can be used as a valid Python identifier. For that purpose all blanks within the string will be replaced by _ and leading numbers get an preceeding _.

spec Module a threaded class for observing a SPEC data file

Motivation

SPEC files can become quite large. Therefore, subsequently reading the entire file to extract a single scan is a quite cumbersome procedure. This module is a proof of concept code to write a file observer starting a reread of the file starting from a stored offset (last known scan position)

class xrutils.io.spec.**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)
Bases: object

Save2HDF5(*h5*, ***keyargs*)

class xrutils.io.spec.**SPECFile**(*filename*, ***keyargs*)
Bases: object

This class represents a single SPEC file. The class provides methodes for updateing an already opened file which makes it particular interesting for interactive use.

Parse()
Parses the file from the starting at last_offset and adding found scans to the scan list.

Save2HDF5(*h5f*, ***keyargs*)
Save the entire file in an HDF5 file. For that purpose a group is set up in the root group of the file with the name of the file without extension and leading path. If the method is called after an previous update only the scans not written to the file meanwhile are saved.

required arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

compactivate compression - true by default

nameoptional name for the file group

Update()
reread the file and add newly added files. The parsing starts at the data offset of the last scan gathered during the last parsing run.

class xrutils.io.spec.**SPECLog**(*filename*, *prompt*, ***keyargs*)
Bases: object

Parse()

Update()

class xrutils.io.spec.**SPECMCA**(*nchan*, *roistart*, *roistop*)
Bases: object

SPECMCA - represents an MCA object in a SPEC file. This class is an abstract class not itended for being used directly. Instead use one of the derived classes SPECMCAFile or SPECMCAInline.

class xrutils.io.spec.**SPECMCAFile**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECMCAInline**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECScan**(*name*, *scannr*, *command*, *date*, *time*, *itime*, *colnames*, *hoffset*, *doffset*, *fid*, *imopnames*, *imopvalues*, *scan_status*)
Bases: object

Represents a single SPEC scan.

ClearData()
Delete the data stored in a scan after it is no longer used.

ReadData()
Set the data attribute of the scan class.

Save2HDF5(*h5f*, ***keyargs*)
Save a SPEC scan to an HDF5 file. The method creates a group with the name of the scan and stores the data there as a table object with name "data". By default the scan group is created under the root group of the HDF5 file. The title of the scan group is ususally the scan command. Metadata of the scan are stored as attributes to the scan group. Additional custom attributes to the scan group can be passed as a dictionary via the optattrs keyword argument.

input arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

groupname or group object of the HDF5 group where to store the data

titlea string with the title for the data

desca string with the description of the data

optattrsa dictionary with optional attributes to store for the data

compactivate compression - true by default

SetMCAParams(*mca_column_format*, *mca_channels*, *mca_start*, *mca_stop*)
Set the parameters used to save the MCA data to the file. This method calculates the number of lines used to store the MCA data from the number of columns and the

required input aguments:

mca_column_formatnumber of columns used to save the data

mca_channelsnumber of MCA channels stored

mca_startfirst channel that is stored

mca_stoplast channel that is stored

plot(**args*, ***keyargs*)

Plot scan data to a matplotlib figure. If newfig=True a new figure instance will be created. If logy=True (default is False) the y-axis will be plotted with a logarithmic scale.

xrutils.io.spec.**geth5_scan**(*h5f*, *scans*, **args*, ***kwargs*)

function to obtain the angular cooridinates as well as intensity values saved in an HDF5 file, which was created from a spec file by the Save2HDF5 method. Especially usefull for reciprocal space map measurements.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables or its filename

scansnumber of the scans of the reciprocal space map (int,tuple or list)

*args: names of the motors (optional) (strings) to read reciprocal space maps measured in coplanar diffraction give: :omname: e.g. name of the omega motor (or its equivalent) :ttname: e.g. name of the two theta motor (or its equivalent)

****kwargs (optional):**

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used

Returns

MAP

or

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

Example

```
>>> [om,tt],MAP = xu.io.geth5_scan(h5file,36,'omega','gamma')
```

spectra Module module to handle spectra data

class xrutils.io.spectra.**SPECTRAFile**(*filename*, *mcatmp=None*, *mcastart=None*, *mcastop=None*)

Bases: object

Represents a SPECTRA data file. The file is read during the Constructor call. This class should work for data stored at beamlines P08 and BW2 at HASYLAB.

Required constructor arguments:

filenamea string with the name of the SPECTRA file

Optional keyword arguments:

mcatmptemplate for the MCA files

mcastart,mcastopstart and stop index for the MCA files, if not given, the class tries to determine the start and stop index automatically.

Read()

Read the data from the file.

ReadMCA()

Save2HDF5(*h5file*, *name*, *group='/'*, *description='SPECTRA scan'*, *mcaname='MCA'*)
Saves the scan to an HDF5 file. The scan is saved to a seperate group of name "name". h5file is either a string for the file name or a HDF5 file object. If the mca attribute is not None mca data will be stored to an chunked array of with name mcaname.

required input arguments:

h5filestring or HDF5 file object

namename of the group where to store the data

optional keyword arguments:

grouproot group where to store the data

descriptionstring with a description of the scan

Return value: The method returns None in the case of everything went fine, True otherwise.

class xrutils.io.spectra.**SPECTRAFileComments**
Bases: dict

Class that describes the comments in the header of a SPECTRA file. The different comments are accessible via the comment keys.

class xrutils.io.spectra.**SPECTRAFileData**
Bases: object

append(*col*)

class xrutils.io.spectra.**SPECTRAFileDataColumn**(*index*, *name*, *unit*, *type*)
Bases: object

class xrutils.io.spectra.**SPECTRAFileParameters**
Bases: dict

class xrutils.io.spectra.**Spectra**(*data_dir*)
Bases: object

abs_corr(*data*, *f*, ***keyargs*)
Perform absorber correction. Data can be either a 1 dimensional data (point detector) or a 2D MCA array. In the case of an array the data array should be of shape (N,NChannels) where N is the number of points in the scan an NChannels the number of channels of the MCA. The absorber values are passed to the function as a 1D array of N elements.

By default the absorber values are taken form a global variable stored in the module called _absorver_factors. Despite this, costume values can be passed via optional keyword arguments.

required input arguments:

mcamatrix with the MCA data

ffilter values along the scan

optional keyword arguments:

ffcustome filter factors

return value: Array with the same shape as mca with the corrected MCA data.

recarray2hdf5(*h5g*, *rec*, *name*, *desc*, ***keyargs*)
Save a record array in an HDF5 file. A pytables table object is used to store the data.

required input arguments:

h5gHDF5 group object or path

recrecord array

namename of the table in the file

descdescription of the table in the file

optional keyword arguments:

return value:

taba HDF5 table object

set_abs_factors (*ff*)
Set the global absorber factors in the module.

spectra2hdf5 (*dir*, *fname*, *mcatemp*, ***keyargs*)
Convert SPECTRA scan data to a HDF5 format.

required input arguments:

dirdirectory where the scan is stored

fnamename of the SPECTRA data file

mcatemptemplate for the MCA file names

optional keyword arguments:

nameoptional name under which to save the data

descoptional description of the scan

xrutils.io.spectra.**get_spectra_files** (*dirname*)
Return a list of spectra files within a directory.

required input arguments:

dirnamename of the directory to search

return values:list with filenames

xrutils.io.spectra.**geth5_spectra_map** (*h5file*, *scans*, **args*, ***kwargs*)
function to obtain the omega and twotheta as well as intensity values for a reciprocal space map saved in an HDF5 file, which was created from a spectra file by the Save2HDF5 method.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables

scansnumber of the scans of the reciprocal space map (int,tuple or list)

***args: names of the motors (strings)**

omnamename of the omega motor (or its equivalent)

ttnamename of the two theta motor (or its equivalent)

****kwargs (optional):**

mcaname of the mca data (if available) otherwise None (default: "MCA")

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used to determine the sample name

Returns

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

xrutils.io.spectra.**read_data** (*fname*)
Read a spectra data file (a file with now MCA data).

required input arguments:

fnamename of the file to read

return values: (data,hdr)

datanumpy record array where the keys are the column names

hdra dictionary with header information

xrutils.io.spectra.**read_mca**(*fname*)
Read a single SPECTRA MCA file.

required input arguments:

fnamename of the file to read

return value:

dataa numpy array witht the MCA data

xrutils.io.spectra.**read_mca_dir**(*dirname*, *filetemp*, ***keyargs*)
Read all MCA files within a directory

xrutils.io.spectra.**read_mcas**(*ftemp*, *cntstart*, *cntstop*)
Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

materials Package

_create_database Module script to create the HDF5 database from the raw data of XOP this file is only needed for administration

_create_database_alt Module script to create the HDF5 database from the raw data of XOP this file is only needed for administration

database Module module to handle access to the optical parameters database

class xrutils.materials.database.**DataBase**(*fname*)
Bases: object

Close()
Close an opend database file.

Create(*dbname*, *dbdesc*)
Creates a new database. If the database file already exists its content is delete.

required input arguments:

dbnamename of the database

dbdesca short description of the database

CreateMaterial(*name*, *description*)
This method creates a new material. If the material group already exists the procedure is aborted.

required input arguments:

namea string with the name of the material

descriptiona string with a description of the material

GetF0(*q*)
Obtain the f0 scattering factor component for a particular momentum transfer q.

required input argument:

qsingle float value or numpy array

GetF1(*en*)
Return the second, energy dependent, real part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

GetF2(*en*)
Return the imaginary part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

Open(*mode='r'*)
Open an existing database file.

SetF0(*parameters*)
Save f0 fit parameters for the set material. The fit parameters are stored in the following order: c,a1,b1,.....,a4,b4

required input argument:

parameterslist or numpy array with the fit parameters

SetF1(*en*, *f1*)
Set f1 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f1list or numpy array with f1 values

SetF2(*en*, *f2*)
Set f2 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f2list or numpy array with f2 values

SetMaterial(*name*)
Set a particular material in the database as the actual material. All operations like setting and getting optical constants are done for this particular material.

requiered input arguments:

namestring with the name of the material

SetWeight(*weight*)
Save weight of the element as float

required input argument:

weightatomic standard weight of the element (float)

xrutils.materials.database.**add_f0_from_intertab**(*db*, *itabfile*)
Read f0 data from international tables of crystallography and add it to the database.

xrutils.materials.database.**add_f0_from_xop**(*db*, *xopfile*)
Read f0 data from f0_xop.dat and add it to the database.

xrutils.materials.database.**add_f1f2_from_ascii_file**(*db*, *asciifile*, *element*)
Read f1 and f2 data for specific element from ASCII file (3 columns) and save it to the database.

xrutils.materials.database.**add_f1f2_from_henkedb**(*db*, *henkefile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_f1f2_from_kissel**(*db*, *kisselfile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_mass_from_NIST**(*db*, *nistfile*)
Read atoms standard mass and save it to the database.

xrutils.materials.database.**init_material_db**(*db*)

elements Module

lattice Module module handling crystal lattice structures

class xrutils.materials.lattice.**Atom**(*name*, *num*)
Bases: object

f(*q*, *en='config'*)
function to calculate the atomic structure factor F

Parameter

qmomentum transfer

enenergy for which F should be calculated, if omitted the value from the xrutils configuration is used

Returns

f (float)

f0(*q*)

f1(*en='config'*)

f2(*en='config'*)

xrutils.materials.lattice.**BCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**BCTLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**BaddeleyiteLattice**(*aa*, *ab*, *a*, *b*, *c*, *beta*, *deg=True*)

xrutils.materials.lattice.**CuMnAsLattice**(*aa*, *ab*, *ac*, *a*, *b*, *c*)

xrutils.materials.lattice.**CubicFm3mBaF2**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**CubicLattice**(*a*)
Returns a Lattice object representing a simple cubic lattice.

required input arguments:

alattice parameter

return value:an instance of Lattice class

xrutils.materials.lattice.**DiamondLattice**(*aa*, *a*)

xrutils.materials.lattice.**FCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**GeneralPrimitiveLattice**(*a*, *b*, *c*, *alpha*, *beta*, *gamma*)

xrutils.materials.lattice.**HCPLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal3CLattice**(*aa*, *ab*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal4HLattice**(*aa*, *ab*, *a*, *c*, *u=0.1875*, *v1=0.25*, *v2=0.4375*)

xrutils.materials.lattice.**Hexagonal6HLattice**(*aa*, *ab*, *a*, *c*)

class xrutils.materials.lattice.**Lattice**(*a1*, *a2*, *a3*, *base=None*)
Bases: object

class Lattice: This object represents a Bravais lattice. A lattice consists of a base

ApplyStrain(*eps*)
Applies a certain strain on a lattice. The result is a change in the base vectors.

requiered input arguments:

epsa 3x3 matrix independent strain components

GetPoint(**args*)
determine lattice points with indices given in the argument

Examples

```
>>> xu.materials.Si.lattice.GetPoint(0,0,4)
array([  0.     ,   0.     ,  21.72416])
```

or

```
>>> xu.materials.Si.lattice.GetPoint((1,1,1))
array([ 5.43104,  5.43104,  5.43104])
```

ReciprocalLattice()

UnitCellVolume()
function to calculate the unit cell volume of a lattice (angstrom^3)

class xrutils.materials.lattice.**LatticeBase**(**args*, ***keyargs*)
Bases: list

The LatticeBase class implements a container for a set of points that form the base of a crystal lattice. An instance of this class can be treated as a simple container object.

append(*atom*, *pos*, *occ=1.0*, *b=0.0*)
add new Atom to the lattice base

Parameter

atomatom object to be added

posposition of the atom

occoccupancy (default=1.0)

bb-factor of the atom used as exp(-b*q**2/(4*pi)**2) to reduce the intensity of this atom (only used in case of temp=0 in StructureFactor and chi calculation)

xrutils.materials.lattice.**NaumanniteLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**PerovskiteTypeRhombohedral**(*aa*, *ab*, *ac*, *a*, *ang*)

xrutils.materials.lattice.**QuartzLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**RockSaltLattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RockSalt_Cubic_Lattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RutileLattice**(*aa*, *ab*, *a*, *c*, *u*)

xrutils.materials.lattice.**TetragonalIndiumLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TetragonalTinLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TrigonalR3mh**(*aa*, *a*, *c*)

xrutils.materials.lattice.**WurtziteLattice**(*aa*, *ab*, *a*, *c*, *u=0.375*, *biso=0.0*)

xrutils.materials.lattice.**ZincBlendeLattice**(*aa*, *ab*, *a*)

material Module class module implements a certain material

class xrutils.materials.material.**Alloy**(*matA*, *matB*, *x*)
Bases: xrutils.materials.material.Material

RelaxationTriangle(*hkl*, *sub*, *exp*)
function which returns the relaxation trianlge for a Alloy of given composition. Reciprocal space coordinates are calculated using the user-supplied experimental class

Parameter

hklMiller Indices

subsubstrate material or lattice constant (Instance of Material class or float)

expExperiment class from which the Transformation object and ndir are needed

Returns

qy,qzreciprocal space coordinates of the corners of the relaxation triangle

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**Cij2Cijkl**(*cij*)

Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.materials.material.**Cijkl2Cij**(*cijkl*)

Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.materials.material.**CubicAlloy**(*matA*, *matB*, *x*)

Bases: xrutils.materials.material.Alloy

ContentBasym(*q_inp*, *q_perp*, *hkl*, *sur*)

function that determines the content of B in the alloy from the reciprocal space position of an asymmetric peak and also sets the content in the current material

Parameter

q_inpinplane peak position of reflection hkl of the alloy in reciprocal space

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured asymmetric reflection

surMiller indices of the surface (determines the perpendicular direction)

Returns

content,[a_inplane,a_perp,a_bulk_perp(x), eps_inplane, eps_perp] :the content of B in the alloy determined from the input variables and the lattice constants calculated from the reciprocal space positions as well as the strain (eps) of the layer

ContentBsym(*q_perp*, *hkl*, *inpr*, *asub*, *relax*)

function that determines the content of B in the alloy from the reciprocal space position of a symetric peak. As an additional input the substrates lattice parameter and the degree of relaxation must be given

Parameter

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured symmetric reflection (also defines the surface normal

inprMiller indices of a Bragg peak defining the inplane reference direction

asubsubstrate lattice constant

relaxdegree of relaxation (needed to obtain the content from symmetric reciprocal space position)

Returns

contentthe content of B in the alloy determined from the input variables

xrutils.materials.material.**CubicElasticTensor**(*c11*, *c12*, *c44*)

Assemble the 6x6 matrix of elastic constants for a cubic material from the three independent components of a cubic crystal

Parameter

c11,c12,c44independent components of the elastic tensor of cubic materials

Returns

6x6 materix with elastic constants

xrutils.materials.material.**GeneralUC**(*a=4*, *b=4*, *c=4*, *alpha=90*, *beta=90*, *gamma=90*, *name='General'*)

general material with primitive unit cell but possibility for different a,b,c and alpha,beta,gamma

Parameters

a,b,cunit cell extenstions (Angstrom)

alphaangle between unit cell vectors b,c

betaangle between unit cell vectors a,c

gammaangle between unit cell vectors a,b

returns a Material object with the specified properties

xrutils.materials.material.**HexagonalElasticTensor**(*c11*, *c12*, *c13*, *c33*, *c44*)

Assemble the 6x6 matrix of elastic constants for a hexagonal material from the five independent components of a hexagonal crystal

Parameter

c11,c12,c13,c33,c44independent components of the elastic tensor of a hexagonal material

Returns

6x6 materix with elastic constants

class xrutils.materials.material.**Material**(*name*, *lat*, *cij=None*, *thetaDebye=None*)

Bases: object

ApplyStrain(*strain*, ***keyargs*)

B

GetMismatch(*mat*)

Calculate the mismatch strain between the material and a second material

Q(**hkl*)

Return the Q-space position for a certain material.

required input arguments:

hkllist or numpy array with the Miller indices (or Q(h,k,l) is also possible)

StructureFactor(*q*, *en='config'*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and energy at a certain temperature of the material

Parameter

qvectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enenergy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

the complex structure factor

StructureFactorForEnergy(*q0*, *en*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and a bunch of energies

Parameter

q0vectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enlist, tuple or array of energy values in eV

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

StructureFactorForQ(*q*, *en0='config'*, *temp=0*)

caluclates the structure factor of a material for a bunch of momentum transfers and a certain energy

Parameter

qvectorial momentum transfers; list of vectores (list, tuple or array) of length 3 e.g.: (Si.Q(0,0,4),Si.Q(0,0,4.1),...) or numpy.array([Si.Q(0,0,4),Si.Q(0,0,4.1)])

en0energy value in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

a1

a2

a3

b1

b2

b3

beta(*en='config'*)

function to calculate the imaginary part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

beta (float)

chi0(*en='config'*)

calculates the complex chi_0 values often needed in simulations. They are closely related to delta and beta (n = 1 + chi_r0/2 + i*chi_i0/2 vs. n = 1 - delta + i*beta)

chih(*q*, *en='config'*, *temp=0*, *polarization='S'*)

calculates the complex polarizability of a material for a certain momentum transfer and energy

Parameter

qmomentum transfer in (1/A)

enxray energy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

polarizationeither 'S' (default) sigma or 'P' pi polarization

Returns

(abs(chih_real),abs(chih_imag)) complex polarizability

critical_angle(*en='config'*, *deg=True*)

calculate critical angle for total external reflection

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

degreturn angle in degree if True otherwise radians (default:True)

Returns

Angle of total external reflection

dTheta(*Q*, *en='config'*)

function to calculate the refractive peak shift

Parameter

Qmomentum transfer (1/A)

enx-ray energy (eV), if omitted the value from the xrutils configuration is used

Returns

deltaThetapeak shift in degree

delta(*en='config'*)

function to calculate the real part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

delta (float)

idx_refraction(*en='config'*)

function to calculate the complex index of refraction of a material in the x-ray range

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

Returns

n (complex)

lam

mu

nu

xrutils.materials.material.**PseudomorphicMaterial**(*submat*, *layermat*)

This function returns a material whos lattice is pseudomorphic on a particular substrate material. This function works meanwhile only for cubic materials.

required input arguments:

submatsubstrate material

layermatbulk material of the layer

return value: An instance of Material holding the new pseudomorphically strained material.

class xrutils.materials.material.**SiGe**(*x*)
Bases: xrutils.materials.material.CubicAlloy

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**index_map_ij2ijkl**(*ij*)

xrutils.materials.material.**index_map_ijkl2ij**(*i*, *j*)

math Package

fit Module module with a function wrapper to scipy.optimize.leastsq for fitting of a 2D function to a peak or a 1D Gauss fit with the odr package

xrutils.math.fit.**fit_peak2d**(*x*, *y*, *data*, *start*, *drange*, *fit_function*, *maxfev=2000*)
fit a two dimensional function to a two dimensional data set e.g. a reciprocal space map

Parameters

x,ydata coordinates (do NOT need to be regularly spaced)

datadata set used for fitting (e.g. intensity at the data coords)

startset of starting parameters for the fit used as first parameter of function fit_function

drangelimits for the data ranges used in the fitting algorithm e.g. it is clever to use only a small region around the peak which should be fitted: [xmin,xmax,ymin,ymax]

fit_functionfunction which should be fitted must accept the parameters (x,y,*params)

Returns

(fitparam,cov)the set of fitted parameters and covariance matrix

xrutils.math.fit.**gauss_fit**(*xdata*, *ydata*, *iparams=[]*, *maxit=200*)
Gauss fit function using odr-pack wrapper in scipy similar to :https: //github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting

Parameters

xdataxcoordinates of the data to be fitted

ydataycoordinates of the data which should be fit

keyword parameters:

iparamsinitial paramters for the fit (determined automatically if nothing is given

maxitmaximal iteration number of the fit

Returns

params,sd_params,itlim

the Gauss parameters as defined in function Gauss1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

functions Module module with several common function needed in xray data analysis

xrutils.math.functions.**Debye1**(*x*)
function to calculate the first Debye function as needed for the calculation of the thermal Debye-Waller-factor by numerical integration

for definition see: :http: //en.wikipedia.org/wiki/Debye_function

D1(x) = (1/x) int_0^x t/(exp(t)-1) dt

Parameters

xargument of the Debye function (float)

Returns

D1(x)float value of the Debye function

xrutils.math.functions.**Gauss1d**(*x*, **p*)
function to calculate a general one dimensional Gaussian

Parameters

plist of parameters of the Gaussian [XCEN,SIGMA,AMP,BACKGROUND] for information: SIGMA = FWHM / (2*sqrt(2*log(2)))

xcoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position x

xrutils.math.functions.**Gauss1d_der_p**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect the parameters p

for parameter description see Gauss1d

xrutils.math.functions.**Gauss1d_der_x**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect to x

for parameter description see Gauss1d

xrutils.math.functions.**Gauss2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Gaussian

Parameters

plist of parameters of the Gauss-function [XCEN,YCEN,SIGMAX,SIGMAY,AMP,BACKGROUND,ANGLE] SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of the Gaussian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz1d**(*x*, **p*)
function to calculate a general one dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE] ANGLE = rotation of the X,Y direction of the Lorentzian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**TwoGauss2d**(*x*, *y*, **p*)
function to calculate two general two dimensional Gaussians

Parameters

plist of parameters of the Gauss-function [XCEN1,YCEN1,SIGMAX1,SIGMAY1,AMP1,ANGLE1,XCEN2,YCEN
SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of
the Gaussian

x,ycoordinate(s) where the function should be evaluated

Return

the value of the Gaussian described by the parameters p at position (x,y)

transforms Module

class xrutils.math.transforms.**AxisToZ**(*newzaxis*)
Bases: xrutils.math.transforms.CoordinateTransform

Creates a coordinate transformation to move a certain axis to the z-axis. The rotation is done along the great circle. The x-axis of the new coordinate frame is created to be normal to the new and original z-axis. The new y-axis is create in order to obtain a right handed coordinate system.

xrutils.math.transforms.**Cij2Cijkl**(*cij*)
Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.math.transforms.**Cijkl2Cij**(*cijkl*)
Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.math.transforms.**CoordinateTransform**(*v1*, *v2*, *v3*)
Bases: xrutils.math.transforms.Transform

Create a Transformation object which transforms a point into a new coordinate frame. The new frame is determined by the three vectors v1/norm(v1), v2/norm(v2) and v3/norm(v3), which need to be orthogonal!

class xrutils.math.transforms.**Transform**(*matrix*)
Bases: object

xrutils.math.transforms.**XRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the x-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**YRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the y-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**ZRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the z-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**index_map_ij2ijkl**(*ij*)

xrutils.math.transforms.**index_map_ijkl2ij**(*i*, *j*)

xrutils.math.transforms.**mycross**(*vec*, *mat*)
function implements the cross-product of a vector with each column of a matrix

xrutils.math.transforms.**rotarb**(*vec*, *axis*, *ang*, *deg=True*)
function implements the rotation around an arbitrary axis by an angle ang positive rotation is anti-clockwise when looking from positive end of axis vector

Parameter

vecnumpy.array or list of length 3

axisnumpy.array or list of length 3

angrotation angle in degree (deg=True) or in rad (deg=False)

degboolean which determines the input format of ang (default: True)

Returns

rotvecrotated vector as numpy.array

Example

```
>>> rotarb([1,0,0],[0,0,1],90)
array([  6.12323400e-17,   1.00000000e+00,   0.00000000e+00])
```

xrutils.math.transforms.**tensorprod**(*vec1*, *vec2*)
function implements an elementwise multiplication of two vectors

vector Module module with vector operations, mostly numpy functionality is used for the vector operation itself, however custom error checking is done to ensure vectors of length 3.

xrutils.math.vector.**VecAngle**(*v1*, *v2*, *deg=False*)
calculate the angle between two vectors. The following formula is used v1.v2 = norm(v1)*norm(v2)*cos(alpha)

alpha = acos((v1.v2)/(norm(v1)*norm(v2)))

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

optional keyword arguments:

deg(default: false) return result in degree otherwise in radiants

return value:float value with the angle inclined by the two vectors

xrutils.math.vector.**VecDot**(*v1*, *v2*)
Calculate the vector dot product.

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

return value:float value

xrutils.math.vector.**VecNorm**(*v*)
Calculate the norm of a vector.

required input arguments:

vvector as list or numpy array

return value:float holding the vector norm

xrutils.math.vector.**VecUnit**(*v*)
Calculate the unit vector of v.

required input arguments:

vvector as list or numpy array

return value:numpy array with the unit vector

xrutils.math.vector.**getSyntax**(*vec*)
returns vector direction in the syntax 'x+' 'z-' or equivalents therefore works only for principle vectors of the coordinate system like e.g. [1,0,0] or [0,2,0]

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

xrutils.math.vector.**getVector**(*string*)
returns unit vector along a rotation axis given in the syntax 'x+' 'z-' or equivalents

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

4.3 analysis Package

xrutils.analysis is a package for assisting with the analysis of x-ray diffraction data, mainly reciprocal space maps

Routines for obtaining line cuts from gridded reciprocal space maps are offered, with the ability to integrate the intensity perpendicular to the line cut direction.

4.3.1 line_cuts Module

xrutils.analysis.line_cuts.**fwhm_exp**(*pos*, *data*)
function to determine the full width at half maximum value of experimental data. Please check the obtained value visually (noise influences the result)

Parameter

posposition of the data points

datadata values

Returns

fwhm value (single float)

xrutils.analysis.line_cuts.**get_omega_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)
extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_omega_scan_bounds_ang**(*omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of omega scan

Parameters

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qx,qzreciprocal space coordinates of the omega scan boundaries

Example

```
>>> qxb,qzb = get_omega_scan_bounds_ang(1.0,4.0,2.4,240,qrange=0.1)
```

xrutils.analysis.line_cuts.**get_omega_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *omrange*, *npoints*, ***kwargs*)

extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the omega scan should be extracted

qzcenterqz-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_qx_scan**(*qx*, *qz*, *intensity*, *qzpos*, ***kwargs*)

extract qx line scan at position qzpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qz

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

Returns

qx,qxintqx scan coordinates and intensities (bounds=False)

qx,qxint,(qxb,qyb)qx scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qxcut,qxcut_int = get_qx_scan(qx,qz,inten,5.0,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extract qz line scan at position qxpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qx

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan(qx,qz,inten,1.5,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan_int**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extracts a qz scan from a gridded reciprocal space map with integration along omega (sample rocking angle) or 2theta direction

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

angrangeintegration range in angular direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

intdirintegration direction 'omega': sample rocking angle (default) '2theta': scattering angle

Returns

qz,qzintqz scan coordinates and intensities (bounds=False)

qz,qzint,(qzb,qzb)qz scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qzcut,qzcut_int = get_qz_scan_int(qx,qz,inten,5.0,omrange=0.3)
```

xrutils.analysis.line_cuts.**get_radial_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_ang(qx,qz,intensity,32.0,64.0,30.0,800,omrange=0.2)
```

xrutils.analysis.line_cuts.**get_radial_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of radial scan

Parameters

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxrad,qzradreciprocal space boundaries of radial scan

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_radial_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the radial scan should be extracted

qzcenterqz-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_q(qx,qz,intensity,0.0,5.0,1.0,100,omrange=0.01)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_ang(qx,qz,intensity,32.0,64.0,4.0,400)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of 2theta scan

Parameters

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the 2theta scan to extract

npointsnumber of points of the 2theta scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxtt,qzttreciprocal space boundaries of 2theta scan (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_ttheta_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the 2theta scan should be extracted

qzcenterqz-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_q(qx,qz,intensity,0.0,4.0,4.4,440)
```

xrutils.analysis.line_cuts.**getindex**(*x*, *y*, *xgrid*, *ygrid*)

gives the indices of the point x,y in the grid given by xgrid ygrid xgrid,ygrid must be arrays containing equidistant points

Parameters

x,ycoordinates of the point of interest (float)

xgrid,ygridgrid coordinates in x and y direction (array)

Returns

ix,iyindex of the closest gridpoint (lower left) of the point (x,y)

4.3.2 `line_cuts3d` Module

xrutils.analysis.line_cuts3d.**get_qx_scan3d**(*gridder*, *qypos*, *qzpos*, ***kwargs*)
extract qx line scan at position y,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qypos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qx,qxintqx scan coordinates and intensities

Example

```
>>> qxcut,qxcut_int = get_qx_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qy_scan3d**(*gridder*, *qxpos*, *qzpos*, ***kwargs*)
extract qy line scan at position x,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qy,qyintqy scan coordinates and intensities

Example

```
>>> qycut,qycut_int = get_qy_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qz_scan3d**(*gridder*, *qxpos*, *qypos*, ***kwargs*)
extract qz line scan at position x,y from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qyposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**getindex3d**(*x*, *y*, *z*, *xgrid*, *ygrid*, *zgrid*)
gives the indices of the point x,y,z in the grid given by xgrid ygrid zgrid xgrid,ygrid,zgrid must be arrays containing equidistant points

Parameters

x,y,zcoordinates of the point of interest (float)

xgrid,ygrid,zgridgrid coordinates in x,y,z direction (array)

Returns

ix,iy,izindex of the closest gridpoint (lower left) of the point (x,y,z)

4.3.3 misc Module

miscellaneous functions helpful in the analysis and experiment

xrutils.analysis.misc.**getangles**(*peak*, *sur*, *inp*)
calculates the chi and phi angles for a given peak

Parameter

peakarray which gives hkl for the peak of interest

surhkl of the surface

inpinplane reference peak or direction

Returns

[chi,phi] for the given peak on surface sur with inplane direction inp as reference

Example

To get the angles for the -224 peak on a 111 surface type[chi,phi] = getangles([-2,2,4],[1,1,1],[2,2,4])

4.3.4 sample_align Module

functions to help with experimental alignment during experiments, especially for experiments with linear detectors

xrutils.analysis.sample_align.**area_detector_calib**(*angle1*, *angle2*, *ccdimages*, *detaxis*, *r_i*, *plot=True*, *cut_off=0.7*, *start=(0, 0, 0, 0)*, *fix=(False, False, False, False)*, *fig=None*, *wl=None*)
function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

parameters

angle1outer detector arm angle

angle2inner detector arm angle

ccdimagesimages of the ccd taken at the angles given above

detaxisdetector arm rotation axis :default: ['z+','y-']

r_iprimary beam direction [xyz][+-] default 'x+'

Keyword_arguments

plotflag to determine if results and intermediate results should be plotted :default: True

cut_offcut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%

startsequence of start values of the fit for parameters, which can not be estimated automatically these are: tiltazimuth,tilt,detector_rotation,outerangle_offset. By default (0,0,0,0) is used.

fixfix parameters of start (default: (False,False,False,False))

figmatplotlib figure used for plotting the error :default: None (creates own figure)

wlwavelength of the experiment in Angstrom (default: config.WAVELENGTH) value does not matter here and does only affect the scaling of the error

xrutils.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03*, *0.03)*, *plot=True*)

helper function to determine the Bragg peak position in a reciprocal space map used to obtain the position needed for correction of the data. the determination is done by fitting a two dimensional Gaussian (xrutils.math.Gauss2d)

PLEASE ALWAYS CHECK THE RESULT CAREFULLY!

Parameter

om,ttangular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]

psdintensity values needed for fitting

omalignaligned omega value, used as first guess in the fit

ttalignaligned two theta values used as first guess in the fit these values are also used to set the range for the fit: the peak should be within +/-frangeAA^{-1} of those values

exphxrdexperiment class used for the conversion between angular and reciprocal space.

frangedata range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})

plotif True (default) function will plot the result of the fit in comparison with the measurement.

Returns

Omfit,ttfit,params,covariance fitted angular values, and the fit parameters (of the Gaussian) as well as their errors

xrutils.analysis.sample_align.**linear_detector_calib**(*angle*, *mca_spectra*, ***kwargs*)

function to calibrate the detector distance/channel per degrees for a straigt linear detector mounted on a detector arm

parameters

anglearray of angles in degree of measured detector spectra

mca_spectracorresponding detector spectra :(shape: (len(angle),Nchannels)

****keyargs passed to psd_chdeg function used for the modelling additional options:**

r_iprimary beam direction as vector [xyz][+-]; default: 'y+'

detaxisdetector rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

returns

L/pixelwidth*pi/180 ~= channel/degree, center_channel[, detector_tilt]

The function also prints out how a linear detector can be initialized using the results obtained from this calibration.

Note: Note: distance of the detector is given by: channel_width*channelperdegree/tan(radians(1))

---

xrutils.analysis.sample_align.**miscut_calc**(*phi*, *aomega*, *zeros=None*, *plot=True*, *omega0=None*)

function to calculate the miscut direction and miscut angle of a sample by fitting a sinusoidal function to the variation of the aligned omega values of more than two reflections. The function can also be used to fit reflectivity alignment values in various azimuths.

Parameters

phiazimuths in which the reflection was aligned (deg)

aomegaaligned omega values (deg)

zeros(optional) angles at which surface is parallel to the beam (deg). For the analysis the angles (aomega-zeros) are used.

plotflag to specify if a visualization of the fit is wanted. :default: True

omega0if specified the nominal value of the reflection is not included as fit parameter, but is fixed to the specified value. This value is MANDATORY if ONLY TWO AZIMUTHs are given.

Returns

[omega0,phi0,miscut]

list with fitted values for

omega0the omega value of the reflection should be close to the nominal one

phi0the azimuth in which the primary beam looks upstairs

miscutamplitude of the sinusoidal variation == miscut angle

xrutils.analysis.sample_align.**psd_chdeg**(*angles*, *channels*, *stdev=None*, *usetilt=False*, *plot=True*)

function to determine the channels per degree using a linear fit of the function nchannel = center_ch+chdeg*tan(angles) or the equivalent including a detector tilt

Parameters

anglesdetector angles for which the position of the beam was measured

channelsdetector channels where the beam was found

keyword arguments:

stdevstandard deviation of the beam position

plotflag to specify if a visualization of the fit should be done

usetiltwhether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: False)

Returns (L/pixelwidth*pi/180 ,centerch[,tilt]):

L/pixelwidth*pi/180 = channel/degree for large detector distance with L sample detector disctance, and pixelwidth the width of one detector channel

Centerch center channel of the detector

Tilt tilt of the detector from perpendicular to the beam

Note: Note: distance of the detector is given by: channelwidth*channelperdegree/tan(radians(1))

---

xrutils.analysis.sample_align.**psd_refl_align**(*primarybeam*, *angles*, *channels*, *plot=True*)

function which calculates the angle at which the sample is parallel to the beam from various angles and detector channels from the reflected beam. The function can be used during the half beam alignment with a linear detector.

Parameters

primarybeamprimary beam channel number

angleslist or numpy.array with angles

channelslist or numpy.array with corresponding detector channels

plotflag to specify if a visualization of the fit is wanted :default: True

Returns

omegaangle at which the sample is parallel to the beam

Example

```
>>> psd_refl_align(500,[0,0.1,0.2,0.3],[550,600,640,700])
```

4.4 io Package

4.4.1 cif Module

class xrutils.io.cif.**CIFFile**(*filename*)

Bases: object

class for parsing CIF (Crystallographic Information File) files. The class aims to provide an additional way of creating material classes instead of manual entering of the information the lattice constants and unit cell structure are parsed from the CIF file

Lattice()

returns a lattice object with the structure from the CIF file

Parse()

function to parse a CIF file. The function reads the space group symmetry operations and the basic atom positions as well as the lattice constants and unit cell angles

SymStruct()

function to obtain the list of different atom positions in the unit cell for the different types of atoms. The data are obtained from the data parsed from the CIF file.

4.4.2 edf Module

class xrutils.io.edf.**EDFDirectory**(*datapath*, ***keyargs*)

Bases: object

Parses a directory for EDF files, which can be stored to a HDF5 file for further usage

Save2HDF5(*h5*, ***keyargs*)
Save2HDF5(h5,**keyargs): Saves the data stored in the EDF files in the specified directory in a HDF5 file as a HDF5 arrays in a subgroup. By default the data is stored in a group given by the foldername - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data (default: pathname) :comp: activate compression - true by default

class xrutils.io.edf.**EDFFile**(*fname*, ***keyargs*)
Bases: object

ReadData()
Read the CCD data into the .data object this function is called by the initialization

Save2HDF5(*h5*, ***keyargs*)
Save2HDF5(h5,**keyargs): Saves the data stored in the EDF file in a HDF5 file as a HDF5 array. By default the data is stored in the root group of the HDF5 file - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data :comp: activate compression - true by default

4.4.3 imagereader Module

class xrutils.io.imagereader.**ImageReader**(*nop1*, *nop2*, *hdrlen=0*, *flatfield=None*, *darkfield=None*, *dtype=<type 'numpy.int16'>*, *byte_swap=False*)
Bases: object

parse CCD frames in the form of tiffs or binary data (*.bin) to numpy arrays. ignore the header since it seems to contain no useful data

The routine was tested so far with RoperScientific files with 4096x4096 pixels created at Hasylab Hamburg, which save an 16bit integer per point. Perkin Elmer images created at Hasylab Hamburg with 2048x2048 pixels.

readImage(*filename*)
read image file and correct for dark- and flatfield in case the necessary data are available.

returned data = ((image data)-(darkfield))/flatfield*average(flatfield)

Parameter

filenamefilename of the image to be read. so far only single filenames are supported. The data might be compressed. supported extensions: .tiff, .bin and .bin.xz

class xrutils.io.imagereader.**PerkinElmer**(***keyargs*)
Bases: xrutils.io.imagereader.ImageReader

parse PerkinElmer CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 2048x2048 pixel images created at Hasylab Hamburg which save an 32bit float per point.

class xrutils.io.imagereader.**RoperCCD**(***keyargs*)
Bases: xrutils.io.imagereader.ImageReader

parse RoperScientific CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 4096x4096 pixel images created at Hasylab Hamburg which save an 16bit integer per point.

4.4.4 panalytical_xml Module

Panalytical XML (www.XRDML.com) data file parser

based on the native python xml.dom.minidom module. want to keep the number of dependancies as small as possible

class xrutils.io.panalytical_xml.**XRDMLFile**(*fname*)
Bases: object

class to handle XRDML data files. The class is supplied with a file name and uses the XRDMLScan class to parse the xrdMeasurement in the file

class xrutils.io.panalytical_xml.**XRDMLMeasurement**(*measurement*)
Bases: object

class to handle scans in a XRDML datafile

xrutils.io.panalytical_xml.**getOmPixcel**(*omraw*, *ttraw*)
function to reshape the Omega values into a form needed for further treatment with xrutils

xrutils.io.panalytical_xml.**getxrdml_map**(*filetemplate*, *scannrs=None*, *path='.'*, *roi=None*)
parses multiple XRDML file and concatenates the results for parsing the xrutils.io.XRDMLFile class is used. The function can be used for parsing maps measured with the PIXCel and point detector.

Parameter

filetemplatetemplate string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames

scannrsint or list of scan numbers

pathcommon path to the filenames

roiregion of interest for the PIXCel detector, for other measurements this i not usefull!

Returns

om,tt,psdas flattened numpy arrays

Example

```
>>> om,tt,psd = xrutils.io.getxrdml_map("samplename_%d.xrdml",[1,2],path="./data")
```

4.4.5 radicon Module

python module for converting radicon data to HDF5

xrutils.io.radicon.**hst2hdf5**(*h5*, *hstfile*, *nofchannels*, ***keyargs*)
Converts a HST file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

hstfilename of the HST file

nofchannelsnumber of channels

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

hstpathpath where the HST file is located (default is the current working directory)

xrutils.io.radicon.**rad2hdf5**(*h5*, *rdcfile*, ***keyargs*)
Converts a RDC file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

rdcfilename of the RDC file

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

rdcpathpath where the RDC file is located (default is the current working directory)

xrutils.io.radicon.**selecthst**(*et_limit*, *mca_info*, *mca_array*)
Select historgrams form the complete set of recorded MCA data and stores it into a new numpy array. The selection is done due to a exposure time limit. Spectra below this limit are ignored.

required input arguments:

et_limitexposure time limit

mca_infopytables table with the exposure data

mca_arrayarray with all the MCA spectra

return value:a numpy array with the selected mca spectra of shape (hstnr,channels).

4.4.6 rotanode_alignment Module

parser for the alignment log file of the rotating anode

class xrutils.io.rotanode_alignment.**RA_Alignment**(*filename*)
Bases: object

class to parse the data file created by the alignment routine (tpalign) at the rotating anode spec installation

this routine does an iterative alignment procedure and saves the center of mass values were it moves after each scan. It iterates between two different peaks and iteratively aligns at each peak between two different motors (om/chi at symmetric peaks, om/phi at asymmetric peaks)

Parse()
parser to read the alignment log and obtain the aligned values at every iteration.

get(*key*)

keys()
returns a list of keys for which aligned values were parsed

plot(*pname*)
function to plot the alignment history for a given peak

Parameters

pnamepeakname for which the alignment should be plotted

4.4.7 seifert Module

a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two posibilities how the data is stored (depending on the use detector):

1. as a simple line scan (using the point detector)
2. as a map using the PSD

In the first case the data ist stored

class xrutils.io.seifert.**SeifertHeader**
Bases: object

save_h5_attribs(*obj*)

class xrutils.io.seifert.**SeifertMultiScan**(*filename*, *m_scan*, *m2*)
Bases: object

dump2hdf5(*h5*, **args*, ***keyargs*)
Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument "group" must be used.

required arguments:

h5a HDF5 file object

optional positional arguments:name for the intensity matrix name for the scan motor name for the second motor more then three parameters are ignored.

optional keyword arguments:

grouppath to the HDF5 group where to store the data

dump2mlab(*fname*, **args*)
Store the data in a matlab file.

parse()

class xrutils.io.seifert.**SeifertScan**(*filename*)
Bases: object

dump2h5(*h5*, **args*, ***keyargs*)
Save the data stored in the Seifert ASCII file to a HDF5 file.

required input arguments:

h5HDF5 file object

optional arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

optional keyword arguments:

groupHDF5 group object where to store the data.

dump2mlab(*fname*, **args*)
Save the data from a Seifert scan to a matlab file.

required input arugments:

fnamename of the matlab file

optional position arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

parse()

xrutils.io.seifert.**repair_key**(*key*)
Repair a key string in the sense that the string is changed in a way that it can be used as a valid Python identifier. For that purpose all blanks within the string will be replaced by _ and leading numbers get an preceeding _.

4.4.8 spec Module

a threaded class for observing a SPEC data file

Motivation

SPEC files can become quite large. Therefore, subsequently reading the entire file to extract a single scan is a quite cumbersome procedure. This module is a proof of concept code to write a file observer starting a reread of the file starting from a stored offset (last known scan position)

class xrutils.io.spec.**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)
Bases: object

Save2HDF5(*h5*, ***keyargs*)

class xrutils.io.spec.**SPECFile**(*filename*, ***keyargs*)
Bases: object

This class represents a single SPEC file. The class provides methodes for updateing an already opened file which makes it particular interesting for interactive use.

Parse()
Parses the file from the starting at last_offset and adding found scans to the scan list.

Save2HDF5(*h5f*, ***keyargs*)
Save the entire file in an HDF5 file. For that purpose a group is set up in the root group of the file with the name of the file without extension and leading path. If the method is called after an previous update only the scans not written to the file meanwhile are saved.

required arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

compactivate compression - true by default

nameoptional name for the file group

Update()
reread the file and add newly added files. The parsing starts at the data offset of the last scan gathered during the last parsing run.

class xrutils.io.spec.**SPECLog**(*filename*, *prompt*, ***keyargs*)
Bases: object

Parse()

Update()

class xrutils.io.spec.**SPECMCA**(*nchan*, *roistart*, *roistop*)
Bases: object

SPECMCA - represents an MCA object in a SPEC file. This class is an abstract class not itended for being used directly. Instead use one of the derived classes SPECMCAFile or SPECMCAInline.

class xrutils.io.spec.**SPECMCAFile**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECMCAInline**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECScan**(*name*, *scannr*, *command*, *date*, *time*, *itime*, *colnames*, *hoffset*, *doffset*, *fid*, *imopnames*, *imopvalues*, *scan_status*)
Bases: object

Represents a single SPEC scan.

ClearData()
Delete the data stored in a scan after it is no longer used.

ReadData()
Set the data attribute of the scan class.

Save2HDF5(*h5f*, ***keyargs*)
Save a SPEC scan to an HDF5 file. The method creates a group with the name of the scan and stores the data there as a table object with name "data". By default the scan group is created under the root group of the HDF5 file. The title of the scan group is ususally the scan command. Metadata of the scan are stored as attributes to the scan group. Additional custom attributes to the scan group can be passed as a dictionary via the optattrs keyword argument.

input arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

groupname or group object of the HDF5 group where to store the data

titlea string with the title for the data

desca string with the description of the data

optattrsa dictionary with optional attributes to store for the data

compactivate compression - true by default

SetMCAParams(*mca_column_format*, *mca_channels*, *mca_start*, *mca_stop*)
Set the parameters used to save the MCA data to the file. This method calculates the number of lines used to store the MCA data from the number of columns and the

required input aguments:

mca_column_formatnumber of columns used to save the data

mca_channelsnumber of MCA channels stored

mca_startfirst channel that is stored

mca_stoplast channel that is stored

plot(**args*, ***keyargs*)
Plot scan data to a matplotlib figure. If newfig=True a new figure instance will be created. If logy=True (default is False) the y-axis will be plotted with a logarithmic scale.

xrutils.io.spec.**geth5_scan**(*h5f*, *scans*, **args*, ***kwargs*)
function to obtain the angular cooridinates as well as intensity values saved in an HDF5 file, which was created from a spec file by the Save2HDF5 method. Especially usefull for reciprocal space map measurements.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables or its filename

scansnumber of the scans of the reciprocal space map (int,tuple or list)

*args: names of the motors (optional) (strings) to read reciprocal space maps measured in coplanar diffraction give: :omname: e.g. name of the omega motor (or its equivalent) :ttname: e.g. name of the two theta motor (or its equivalent)

****kwargs (optional):**

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used

Returns

MAP

or

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

Example

```
>>> [om,tt],MAP = xu.io.geth5_scan(h5file,36,'omega','gamma')
```

4.4.9 `spectra` Module

module to handle spectra data

class `xrutils.io.spectra.`**`SPECTRAFile`**(*filename*, *mcatmp=None*, *mcastart=None*, *mcastop=None*)

Bases: `object`

Represents a SPECTRA data file. The file is read during the Constructor call. This class should work for data stored at beamlines P08 and BW2 at HASYLAB.

Required constructor arguments:

filenamea string with the name of the SPECTRA file

Optional keyword arguments:

mcatmptemplate for the MCA files

mcastart,mcastopstart and stop index for the MCA files, if not given, the class tries to determine the start and stop index automatically.

`Read`()
Read the data from the file.

`ReadMCA`()

`Save2HDF5`(*h5file*, *name*, *group='/'*, *description='SPECTRA scan'*, *mcaname='MCA'*)
Saves the scan to an HDF5 file. The scan is saved to a seperate group of name "name". h5file is either a string for the file name or a HDF5 file object. If the mca attribute is not None mca data will be stored to an chunked array of with name mcaname.

required input arguments:

h5filestring or HDF5 file object

namename of the group where to store the data

optional keyword arguments:

grouproot group where to store the data

descriptionstring with a description of the scan

Return value: The method returns None in the case of everything went fine, True otherwise.

class `xrutils.io.spectra.`**`SPECTRAFileComments`**
Bases: `dict`

Class that describes the comments in the header of a SPECTRA file. The different comments are accessible via the comment keys.

class `xrutils.io.spectra.`**`SPECTRAFileData`**
Bases: `object`

`append`(*col*)

class `xrutils.io.spectra.`**`SPECTRAFileDataColumn`**(*index*, *name*, *unit*, *type*)
Bases: `object`

class `xrutils.io.spectra.`**`SPECTRAFileParameters`**
Bases: `dict`

class xrutils.io.spectra.**Spectra**(*data_dir*)

Bases: object

abs_corr(*data*, *f*, ***keyargs*)

Perform absorber correction. Data can be either a 1 dimensional data (point detector) or a 2D MCA array. In the case of an array the data array should be of shape (N,NChannels) where N is the number of points in the scan an NChannels the number of channels of the MCA. The absorber values are passed to the function as a 1D array of N elements.

By default the absorber values are taken form a global variable stored in the module called _absorver_factors. Despite this, costume values can be passed via optional keyword arguments.

required input arguments:

mcamatrix with the MCA data

ffilter values along the scan

optional keyword arguments:

ffcustome filter factors

return value: Array with the same shape as mca with the corrected MCA data.

recarray2hdf5(*h5g*, *rec*, *name*, *desc*, ***keyargs*)

Save a record array in an HDF5 file. A pytables table object is used to store the data.

required input arguments:

h5gHDF5 group object or path

recrecord array

namename of the table in the file

descdescription of the table in the file

optional keyword arguments:

return value:

taba HDF5 table object

set_abs_factors(*ff*)

Set the global absorber factors in the module.

spectra2hdf5(*dir*, *fname*, *mcatemp*, ***keyargs*)

Convert SPECTRA scan data to a HDF5 format.

required input arguments:

dirdirectory where the scan is stored

fnamename of the SPECTRA data file

mcatemptemplate for the MCA file names

optional keyword arguments:

nameoptional name under which to save the data

descoptional description of the scan

xrutils.io.spectra.**get_spectra_files**(*dirname*)

Return a list of spectra files within a directory.

required input arguments:

dirnamename of the directory to search

return values:list with filenames

xrutils.io.spectra.**geth5_spectra_map**(*h5file*, *scans*, **args*, ***kwargs*)
function to obtain the omega and twotheta as well as intensity values for a reciprocal space map saved in an HDF5 file, which was created from a spectra file by the Save2HDF5 method.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables

scansnumber of the scans of the reciprocal space map (int,tuple or list)

***args: names of the motors (strings)**

omnamename of the omega motor (or its equivalent)

ttnamename of the two theta motor (or its equivalent)

****kwargs (optional):**

mcaname of the mca data (if available) otherwise None (default: "MCA")

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used to determine the sample name

Returns

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

xrutils.io.spectra.**read_data**(*fname*)
Read a spectra data file (a file with now MCA data).

required input arguments:

fnamename of the file to read

return values: (data,hdr)

datanumpy record array where the keys are the column names

hdra dictionary with header information

xrutils.io.spectra.**read_mca**(*fname*)
Read a single SPECTRA MCA file.

required input arguments:

fnamename of the file to read

return value:

dataa numpy array witht the MCA data

xrutils.io.spectra.**read_mca_dir**(*dirname*, *filetemp*, ***keyargs*)
Read all MCA files within a directory

xrutils.io.spectra.**read_mcas**(*ftemp*, *cntstart*, *cntstop*)
Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

4.5 materials Package

4.5.1 _create_database Module

script to create the HDF5 database from the raw data of XOP this file is only needed for administration

4.5.2 `_create_database_alt` Module

script to create the HDF5 database from the raw data of XOP this file is only needed for administration

4.5.3 `database` Module

module to handle access to the optical parameters database

class `xrutils.materials.database.`**`DataBase`**(*fname*)
Bases: `object`

`Close`()
Close an opend database file.

`Create`(*dbname*, *dbdesc*)
Creates a new database. If the database file already exists its content is delete.

required input arguments:

dbnamename of the database

dbdesca short description of the database

`CreateMaterial`(*name*, *description*)
This method creates a new material. If the material group already exists the procedure is aborted.

required input arguments:

namea string with the name of the material

descriptiona string with a description of the material

`GetF0`(*q*)
Obtain the f0 scattering factor component for a particular momentum transfer q.

required input argument:

qsingle float value or numpy array

`GetF1`(*en*)
Return the second, energy dependent, real part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

`GetF2`(*en*)
Return the imaginary part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

`Open`(*mode='r'*)
Open an existing database file.

`SetF0`(*parameters*)
Save f0 fit parameters for the set material. The fit parameters are stored in the following order: c,a1,b1,.....,a4,b4

required input argument:

parameterslist or numpy array with the fit parameters

`SetF1`(*en*, *f1*)
Set f1 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f1list or numpy array with f1 values

SetF2(*en*, *f2*)
Set f2 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f2list or numpy array with f2 values

SetMaterial(*name*)
Set a particular material in the database as the actual material. All operations like setting and getting optical constants are done for this particular material.

requiered input arguments:

namestring with the name of the material

SetWeight(*weight*)
Save weight of the element as float

required input argument:

weightatomic standard weight of the element (float)

xrutils.materials.database.**add_f0_from_intertab**(*db*, *itabfile*)
Read f0 data from international tables of crystallography and add it to the database.

xrutils.materials.database.**add_f0_from_xop**(*db*, *xopfile*)
Read f0 data from f0_xop.dat and add it to the database.

xrutils.materials.database.**add_f1f2_from_ascii_file**(*db*, *asciifile*, *element*)
Read f1 and f2 data for specific element from ASCII file (3 columns) and save it to the database.

xrutils.materials.database.**add_f1f2_from_henkedb**(*db*, *henkefile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_f1f2_from_kissel**(*db*, *kisselfile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_mass_from_NIST**(*db*, *nistfile*)
Read atoms standard mass and save it to the database.

xrutils.materials.database.**init_material_db**(*db*)

4.5.4 elements Module

4.5.5 lattice Module

module handling crystal lattice structures

class xrutils.materials.lattice.**Atom**(*name*, *num*)
Bases: object

f(*q*, *en='config'*)
function to calculate the atomic structure factor F

Parameter

qmomentum transfer

enenergy for which F should be calculated, if omitted the value from the xrutils configuration is used

Returns

f (float)

f0(*q*)

f1(*en='config'*)

f2(*en='config'*)

xrutils.materials.lattice.**BCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**BCTLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**BaddeleyiteLattice**(*aa*, *ab*, *a*, *b*, *c*, *beta*, *deg=True*)

xrutils.materials.lattice.**CuMnAsLattice**(*aa*, *ab*, *ac*, *a*, *b*, *c*)

xrutils.materials.lattice.**CubicFm3mBaF2**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**CubicLattice**(*a*)

Returns a Lattice object representing a simple cubic lattice.

required input arguments:

alattice parameter

return value:an instance of Lattice class

xrutils.materials.lattice.**DiamondLattice**(*aa*, *a*)

xrutils.materials.lattice.**FCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**GeneralPrimitiveLattice**(*a*, *b*, *c*, *alpha*, *beta*, *gamma*)

xrutils.materials.lattice.**HCPLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal3CLattice**(*aa*, *ab*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal4HLattice**(*aa*, *ab*, *a*, *c*, *u=0.1875*, *v1=0.25*, *v2=0.4375*)

xrutils.materials.lattice.**Hexagonal6HLattice**(*aa*, *ab*, *a*, *c*)

class xrutils.materials.lattice.**Lattice**(*a1*, *a2*, *a3*, *base=None*)

Bases: object

class Lattice: This object represents a Bravais lattice. A lattice consists of a base

ApplyStrain(*eps*)

Applies a certain strain on a lattice. The result is a change in the base vectors.

requiered input arguments:

epsa 3x3 matrix independent strain components

GetPoint(**args*)

determine lattice points with indices given in the argument

Examples

```
>>> xu.materials.Si.lattice.GetPoint(0,0,4)
array([  0.     ,   0.     ,  21.72416])
```

or

```
>>> xu.materials.Si.lattice.GetPoint((1,1,1))
array([ 5.43104,  5.43104,  5.43104])
```

ReciprocalLattice()

UnitCellVolume()

function to calculate the unit cell volume of a lattice (angstrom^3)

class xrutils.materials.lattice.**LatticeBase**(**args*, ***keyargs*)

Bases: list

The LatticeBase class implements a container for a set of points that form the base of a crystal lattice. An instance of this class can be treated as a simple container object.

append(*atom*, *pos*, *occ=1.0*, *b=0.0*)
add new Atom to the lattice base

Parameter

atomatom object to be added

posposition of the atom

occoccupancy (default=1.0)

bb-factor of the atom used as exp(-b*q**2/(4*pi)**2) to reduce the intensity of this atom (only used in case of temp=0 in StructureFactor and chi calculation)

xrutils.materials.lattice.**NaumanniteLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**PerovskiteTypeRhombohedral**(*aa*, *ab*, *ac*, *a*, *ang*)

xrutils.materials.lattice.**QuartzLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**RockSaltLattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RockSalt_Cubic_Lattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RutileLattice**(*aa*, *ab*, *a*, *c*, *u*)

xrutils.materials.lattice.**TetragonalIndiumLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TetragonalTinLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TrigonalR3mh**(*aa*, *a*, *c*)

xrutils.materials.lattice.**WurtziteLattice**(*aa*, *ab*, *a*, *c*, *u=0.375*, *biso=0.0*)

xrutils.materials.lattice.**ZincBlendeLattice**(*aa*, *ab*, *a*)

4.5.6 `material` Module

class module implements a certain material

class xrutils.materials.material.**Alloy**(*matA*, *matB*, *x*)
Bases: xrutils.materials.material.Material

RelaxationTriangle(*hkl*, *sub*, *exp*)
function which returns the relaxation trianlge for a Alloy of given composition. Reciprocal space coordinates are calculated using the user-supplied experimental class

Parameter

hklMiller Indices

subsubstrate material or lattice constant (Instance of Material class or float)

expExperiment class from which the Transformation object and ndir are needed

Returns

qy,qzreciprocal space coordinates of the corners of the relaxation triangle

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**Cij2Cijkl**(*cij*)
Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.materials.material.**Cijkl2Cij**(*cijkl*)

Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.materials.material.**CubicAlloy**(*matA*, *matB*, *x*)

Bases: xrutils.materials.material.Alloy

ContentBasym(*q_inp*, *q_perp*, *hkl*, *sur*)

function that determines the content of B in the alloy from the reciprocal space position of an asymmetric peak and also sets the content in the current material

Parameter

q_inpinplane peak position of reflection hkl of the alloy in reciprocal space

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured asymmetric reflection

surMiller indices of the surface (determines the perpendicular direction)

Returns

content,[a_inplane,a_perp,a_bulk_perp(x), eps_inplane, eps_perp] :the content of B in the alloy determined from the input variables and the lattice constants calculated from the reciprocal space positions as well as the strain (eps) of the layer

ContentBsym(*q_perp*, *hkl*, *inpr*, *asub*, *relax*)

function that determines the content of B in the alloy from the reciprocal space position of a symetric peak. As an additional input the substrates lattice parameter and the degree of relaxation must be given

Parameter

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured symmetric reflection (also defines the surface normal

inprMiller indices of a Bragg peak defining the inplane reference direction

asubsubstrate lattice constant

relaxdegree of relaxation (needed to obtain the content from symmetric reciprocal space position)

Returns

contentthe content of B in the alloy determined from the input variables

xrutils.materials.material.**CubicElasticTensor**(*c11*, *c12*, *c44*)

Assemble the 6x6 matrix of elastic constants for a cubic material from the three independent components of a cubic crystal

Parameter

c11,c12,c44independent components of the elastic tensor of cubic materials

Returns

6x6 materix with elastic constants

xrutils.materials.material.**GeneralUC**(*a=4*, *b=4*, *c=4*, *alpha=90*, *beta=90*, *gamma=90*, *name='General'*)

general material with primitive unit cell but possibility for different a,b,c and alpha,beta,gamma

Parameters

a,b,cunit cell extenstions (Angstrom)

alphaangle between unit cell vectors b,c

betaangle between unit cell vectors a,c

gammaangle between unit cell vectors a,b

returns a Material object with the specified properties

xrutils.materials.material.**HexagonalElasticTensor**(*c11*, *c12*, *c13*, *c33*, *c44*)

Assemble the 6x6 matrix of elastic constants for a hexagonal material from the five independent components of a hexagonal crystal

Parameter

c11,c12,c13,c33,c44independent components of the elastic tensor of a hexagonal material

Returns

6x6 materix with elastic constants

class xrutils.materials.material.**Material**(*name*, *lat*, *cij=None*, *thetaDebye=None*)

Bases: object

ApplyStrain(*strain*, ***keyargs*)

B

GetMismatch(*mat*)

Calculate the mismatch strain between the material and a second material

Q(**hkl*)

Return the Q-space position for a certain material.

required input arguments:

hkllist or numpy array with the Miller indices (or Q(h,k,l) is also possible)

StructureFactor(*q*, *en='config'*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and energy at a certain temperature of the material

Parameter

qvectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enenergy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

the complex structure factor

StructureFactorForEnergy(*q0*, *en*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and a bunch of energies

Parameter

q0vectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enlist, tuple or array of energy values in eV

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

StructureFactorForQ(*q*, *en0='config'*, *temp=0*)
caluclates the structure factor of a material for a bunch of momentum transfers and a certain energy

Parameter

qvectorial momentum transfers; list of vectores (list, tuple or array) of length 3 e.g.: (Si.Q(0,0,4),Si.Q(0,0,4.1),...) or numpy.array([Si.Q(0,0,4),Si.Q(0,0,4.1)])

en0energy value in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

a1

a2

a3

b1

b2

b3

beta(*en='config'*)
function to calculate the imaginary part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

beta (float)

chi0(*en='config'*)
calculates the complex chi_0 values often needed in simulations. They are closely related to delta and beta (n = 1 + chi_r0/2 + i*chi_i0/2 vs. n = 1 - delta + i*beta)

chih(*q*, *en='config'*, *temp=0*, *polarization='S'*)
calculates the complex polarizability of a material for a certain momentum transfer and energy

Parameter

qmomentum transfer in (1/A)

enxray energy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

polarizationeither 'S' (default) sigma or 'P' pi polarization

Returns

(abs(chih_real),abs(chih_imag)) complex polarizability

critical_angle(*en='config'*, *deg=True*)
calculate critical angle for total external reflection

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

degreturn angle in degree if True otherwise radians (default:True)

Returns

Angle of total external reflection

dTheta(*Q*, *en='config'*)
function to calculate the refractive peak shift

Parameter

Qmomentum transfer (1/A)

enx-ray energy (eV), if omitted the value from the xrutils configuration is used

Returns

deltaThetapeak shift in degree

delta(*en='config'*)
function to calculate the real part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

delta (float)

idx_refraction(*en='config'*)
function to calculate the complex index of refraction of a material in the x-ray range

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

Returns

n (complex)

lam

mu

nu

xrutils.materials.material.**PseudomorphicMaterial**(*submat*, *layermat*)
This function returns a material whos lattice is pseudomorphic on a particular substrate material. This function works meanwhile only for cubic materials.

required input arguments:

submatsubstrate material

layermatbulk material of the layer

return value: An instance of Material holding the new pseudomorphically strained material.

class xrutils.materials.material.**SiGe**(*x*)
Bases: xrutils.materials.material.CubicAlloy

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**index_map_ij2ijkl**(*ij*)

xrutils.materials.material.**index_map_ijkl2ij**(*i*, *j*)

4.6 math Package

4.6.1 fit Module

module with a function wrapper to scipy.optimize.leastsq for fitting of a 2D function to a peak or a 1D Gauss fit with the odr package

xrutils.math.fit.**fit_peak2d**(*x*, *y*, *data*, *start*, *drange*, *fit_function*, *maxfev=2000*)
fit a two dimensional function to a two dimensional data set e.g. a reciprocal space map

Parameters

x,ydata coordinates (do NOT need to be regularly spaced)

datadata set used for fitting (e.g. intensity at the data coords)

startset of starting parameters for the fit used as first parameter of function fit_function

drangelimits for the data ranges used in the fitting algorithm e.g. it is clever to use only a small region around the peak which should be fitted: [xmin,xmax,ymin,ymax]

fit_functionfunction which should be fitted must accept the parameters (x,y,*params)

Returns

(fitparam,cov)the set of fitted parameters and covariance matrix

xrutils.math.fit.**gauss_fit**(*xdata*, *ydata*, *iparams=[]*, *maxit=200*)
Gauss fit function using odr-pack wrapper in scipy similar to :https: //github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting

Parameters

xdataxcoordinates of the data to be fitted

ydataycoordinates of the data which should be fit

keyword parameters:

iparamsinitial paramters for the fit (determined automatically if nothing is given

maxitmaximal iteration number of the fit

Returns

params,sd_params,itlim

the Gauss parameters as defined in function Gauss1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

4.6.2 functions Module

module with several common function needed in xray data analysis

xrutils.math.functions.**Debye1**(*x*)
function to calculate the first Debye function as needed for the calculation of the thermal Debye-Waller-factor by numerical integration

for definition see: :http: //en.wikipedia.org/wiki/Debye_function

D1(x) = (1/x) int_0^x t/(exp(t)-1) dt

Parameters

xargument of the Debye function (float)

Returns

D1(x)float value of the Debye function

xrutils.math.functions.**Gauss1d**(*x*, **p*)
function to calculate a general one dimensional Gaussian

Parameters

plist of parameters of the Gaussian [XCEN,SIGMA,AMP,BACKGROUND] for information: SIGMA = FWHM / (2*sqrt(2*log(2)))

xcoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position x

xrutils.math.functions.**Gauss1d_der_p**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect the parameters p

for parameter description see Gauss1d

xrutils.math.functions.**Gauss1d_der_x**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect to x

for parameter description see Gauss1d

xrutils.math.functions.**Gauss2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Gaussian

Parameters

plist of parameters of the Gauss-function [XCEN,YCEN,SIGMAX,SIGMAY,AMP,BACKGROUND,ANGLE]
SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of
the Gaussian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz1d**(*x*, **p*)
function to calculate a general one dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE]
ANGLE = rotation of the X,Y direction of the Lorentzian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**TwoGauss2d**(*x*, *y*, **p*)
function to calculate two general two dimensional Gaussians

Parameters

plist of parameters of the Gauss-function [XCEN1,YCEN1,SIGMAX1,SIGMAY1,AMP1,ANGLE1,XCEN2,YCEN
SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of
the Gaussian

x,ycoordinate(s) where the function should be evaluated

Return

the value of the Gaussian described by the parameters p at position (x,y)

4.6.3 `transforms` Module

class xrutils.math.transforms.**AxisToZ**(*newzaxis*)
Bases: xrutils.math.transforms.CoordinateTransform

Creates a coordinate transformation to move a certain axis to the z-axis. The rotation is done along the great circle. The x-axis of the new coordinate frame is created to be normal to the new and original z-axis. The new y-axis is create in order to obtain a right handed coordinate system.

xrutils.math.transforms.**Cij2Cijkl**(*cij*)
Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.math.transforms.**Cijkl2Cij**(*cijkl*)
Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.math.transforms.**CoordinateTransform**(*v1*, *v2*, *v3*)
Bases: xrutils.math.transforms.Transform

Create a Transformation object which transforms a point into a new coordinate frame. The new frame is determined by the three vectors v1/norm(v1), v2/norm(v2) and v3/norm(v3), which need to be orthogonal!

class xrutils.math.transforms.**Transform**(*matrix*)
Bases: object

xrutils.math.transforms.**XRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the x-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**YRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the y-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**ZRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the z-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**index_map_ij2ijkl**(*ij*)

xrutils.math.transforms.**index_map_ijkl2ij**(*i*, *j*)

xrutils.math.transforms.**mycross**(*vec*, *mat*)
function implements the cross-product of a vector with each column of a matrix

xrutils.math.transforms.**rotarb**(*vec*, *axis*, *ang*, *deg=True*)
function implements the rotation around an arbitrary axis by an angle ang positive rotation is anti-clockwise when looking from positive end of axis vector

Parameter

vecnumpy.array or list of length 3

axisnumpy.array or list of length 3

angrotation angle in degree (deg=True) or in rad (deg=False)

degboolean which determines the input format of ang (default: True)

Returns

rotvecrotated vector as numpy.array

Example

```
>>> rotarb([1,0,0],[0,0,1],90)
array([  6.12323400e-17,   1.00000000e+00,   0.00000000e+00])
```

xrutils.math.transforms.**tensorprod**(*vec1*, *vec2*)
function implements an elementwise multiplication of two vectors

4.6.4 `vector` Module

module with vector operations, mostly numpy functionality is used for the vector operation itself, however custom error checking is done to ensure vectors of length 3.

xrutils.math.vector.**VecAngle**(*v1*, *v2*, *deg=False*)
calculate the angle between two vectors. The following formula is used v1.v2 = norm(v1)*norm(v2)*cos(alpha)

alpha = acos((v1.v2)/(norm(v1)*norm(v2)))

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

optional keyword arguments:

deg(default: false) return result in degree otherwise in radiants

return value:float value with the angle inclined by the two vectors

xrutils.math.vector.**VecDot**(*v1*, *v2*)
Calculate the vector dot product.

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

return value:float value

xrutils.math.vector.**VecNorm**(*v*)
Calculate the norm of a vector.

required input arguments:

vvector as list or numpy array

return value:float holding the vector norm

xrutils.math.vector.**VecUnit**(*v*)
Calculate the unit vector of v.

required input arguments:

vvector as list or numpy array

return value:numpy array with the unit vector

xrutils.math.vector.**getSyntax**(*vec*)
returns vector direction in the syntax 'x+' 'z-' or equivalents therefore works only for principle vectors of the coordinate system like e.g. [1,0,0] or [0,2,0]

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

xrutils.math.vector.**getVector**(*string*)
returns unit vector along a rotation axis given in the syntax 'x+' 'z-' or equivalents

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

4.7 Modules

4.7.1 API-documentation

xrutils Package

xrutils is a package for assisting with x-ray diffraction experiments

It helps with planning experiments as well as analyzing the data.

Authors Dominik Kriegner and Eugen Wintersberger

config Module

module to parse xrutils user-specific config file the parsed values are provide as global constants for the use in other parts of xrutils. The config file with the default constants is found in the python installation path of xrutils. It is however not recommended to change things there, instead the user-specific config file ~/.xrutils.conf or the local xrutils.conf file should be used.

exception Module

xrutils derives its own exceptions which are raised upon wrong input when calling one of xrutils functions. none of the pre-defined exceptions is made for that purpose.

exception xrutils.exception.**InputError**(*msg*)
Bases: exceptions.Exception

Exception raised for errors in the input. Either wrong datatype not handled by TypeError or missing mandatory keyword argument (Note that the obligation to give keyword arguments might depend on the value of the arguments itself)

Attibutesexpr – input expression in which the error occurred :msg: – explanation of the error

experiment Module

module helping with planning and analyzing experiments

various classes are provided for

* describing experiments * calculating angular coordinates of Bragg reflections * converting angular coordinates to Q-space and vice versa * simulating powder diffraction patterns for materials

class xrutils.experiment.**Experiment**(*ipdir*, *ndir*, ***keyargs*)
Bases: object

base class for describing experiments users should use the derived classes: HXRD, GID, Powder

Ang2HKL(**args*, ***kwargs*)
angular to (h,k,l) space conversion. It will set the UB argument to Ang2Q and pass all other parameters unchanged. See Ang2Q for description of the rest of the arguments.

Parameters

****kwargs: optional keyword arguments**

Breciprocal space conversion matrix of a Material. you can specify the matrix B (default identiy matrix) shape needs to be (3,3)

matMaterial object to use to obtain a B matrix (e.g. xu.materials.Si) can be used as alternative to the B keyword argument B is favored in case both are given

Uorientation matrix U can be given if none is given the orientation defined in the Experiment class is used.

dettypedetector type: one of ('point', 'linear', 'area') decides which routine of Ang2Q to call default 'point'

Returns

H K L coordinates as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input (*args)

Q2Ang(*qvec*)

TiltAngle(*q*, *deg=True*)
TiltAngle(q,deg=True): Return the angle between a q-space position and the surface normal.

Parameters

qlist or numpy array with the reciprocal space position

optional keyword arguments:

degTrue/False whether the return value should be in degree or radians :(default: True)

Transform(*v*, ***kwargs*)
transforms a vector, matrix or tensor of rank 4 (e.g. elasticity tensor) to the coordinate frame of the Experiment class.

Parameters

vobject to transform, list or numpy array of shape (n,) (n,n), (n,n,n,n) where n is the rank of the transformation matrix

Returns

transformed object of the same shape as v

energy

wavelength

class xrutils.experiment.**GID**(*idir*, *ndir*, ***keyargs*)
Bases: xrutils.experiment.Experiment

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a four circle (alpha_i,azimuth,twotheta,beta) goniometer to help with GID experiments at the ROTATING ANODE. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*ai*, *phi*, *tt*, *beta*, ***kwargs*)
angular to momentum space conversion for a point detector. Also see help GID.Ang2Q for procedures which treat line and area detectors

Parameters

ai,phi,tt,betasample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than ai,phi,tt,beta - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(*Q*, *trans=True*, *deg=True*, ***kwargs*)
calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

Qa list or numpy array of shape (3) with q-space vector components

optional keyword arguments:

transTrue/False apply coordinate transformation on Q

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

Returns

a numpy array of shape (4) with the four GID scattering angles which are [alpha_i,azimuth,twotheta,beta]

alpha_iincidence angle to surface (at the moment always 0)

azimuthsample rotation with respect to the inplane reference direction

twothetascattering angle

betaexit angle from surface (at the moment always 0)

class xrutils.experiment.**GID_ID10B**(*idir*, *ndir*, ***keyargs*)
Bases: xrutils.experiment.GID

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a four circle (theta,omega,delta,gamma) goniometer to help with GID experiments at ID10B / ESRF. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*th*, *om*, *delta*, *gamma*, ***kwargs*)
angular to momentum space conversion for a point detector. Also see help GID_ID10B.Ang2Q for procedures which treat line and area detectors

Parameters

th,om,delta,gammasample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than th,om,delta,gamma - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(*Q*, *trans=True*, *deg=True*, ***kwargs*)

calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

Qa list or numpy array of shape (3) with q-space vector components

optional keyword arguments:

transTrue/False apply coordinate transformation on Q

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

Returns

a numpy array of shape (4) with the four GID scattering angles which are (theta,omega,delta,gamma)

thetaincidence angle to surface (at the moment always 0)

omegasample rotation with respect to the inplane reference direction

deltaexit angle from surface (at the moment always 0)

gammascattering angle

class xrutils.experiment.**GISAXS**(*idir*, *ndir*, ***keyargs*)

Bases: xrutils.experiment.Experiment

class describing grazing incidence x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as it helps with analyzing measured data

the class describes a three circle (alpha_i,twotheta,beta) goniometer to help with GISAXS experiments at the ROTATING ANODE. 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*ai*, *tt*, *beta*, ***kwargs*)

angular to momentum space conversion for a point detector. Also see help GISAXS.Ang2Q for procedures which treat line and area detectors

Parameters

ai,tt,betasample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 3. used angles are than ai,tt,beta - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(*Q*, *trans=True*, *deg=True*, ***kwargs*)

class xrutils.experiment.**HXRD**(*idir*, *ndir*, ***keyargs*)
Bases: xrutils.experiment.Experiment

class describing high angle x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as helps with analyzing measured data

the class describes a two circle (omega,twotheta) goniometer to help with coplanar x-ray diffraction experiments. Nevertheless 3D data can be treated with the use of linear and area detectors. see help self.Ang2Q

Ang2Q(*om*, *tt*, ***kwargs*)
angular to momentum space conversion for a point detector. Also see help HXRD.Ang2Q for procedures which treat line and area detectors

Parameters

om,ttsample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 2. used angles are than om,tt - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang(**Q*, ***keyargs*)
Convert a reciprocal space vector Q to COPLANAR scattering angles. The keyword argument trans determines whether Q should be transformed to the experimental coordinate frame or not.

Parameters

Qa list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz

optional keyword arguments:

transTrue/False apply coordinate transformation on Q (default True)

degTrue/Flase (default True) determines if the angles are returned in radians or degrees

geometrydetermines the scattering geometry:

- "hi_lo" high incidence and low exit
- "lo_hi" low incidence and high exit

•**"real" general geometry with angles determined by q-coordinates (azimuth); this and upper geomet**

–"realTilt" general geometry with angles determined by q-coordinates (tilt); returns [omega,chi,phi,twotheta]

defaultself.geometry

refracboolean to determine if refraction is taken into account :default: False if True then also a material must be given

matMaterial object; needed to obtain its optical properties for refraction correction, otherwise not used

full_outputboolean to determine if additional output is given to determine scattering angles more accurately in case refraction is set to True :default: False

fi,fdif refraction correction is applied one can optionally specify the facet through which the beam enters (fi) and exits (fd) fi, fd must be the surface normal vectors (not transformed & not necessarily normalized). If omitted the normal direction of the experiment is used.

Returns

a numpy array of shape (4) with four scattering angles which are [omega,chi,phi,twotheta]

omegaincidence angle with respect to surface

chisample tilt for the case of non-coplanar geometry

phisample azimuth with respect to inplane reference direction

twothetascattering angle

if full_output: a numpy array of shape (6) with five angles which are [omega,chi,phi,twotheta,psi_i,psi_d]

psi_ioffset of the incidence beam from the scattering plane due to refraction

pdi_doffset ot the diffracted beam from the scattering plane due to refraction

class xrutils.experiment.**NonCOP**(*idir*, *ndir*, ***keyargs*)

Bases: xrutils.experiment.Experiment

class describing high angle x-ray diffraction experiments the class helps with calculating the angles of Bragg reflections as well as helps with analyzing measured data for NON-COPLANAR measurements, where the tilt is used to align asymmetric peaks, like in the case of a polefigure measurement.

the class describes a four circle (omega,twotheta) goniometer to help with x-ray diffraction experiments. Linear and area detectors can be treated as described in "help self.Ang2Q"

Ang2Q(*om*, *chi*, *phi*, *tt*, ***kwargs*)

angular to momentum space conversion for a point detector. Also see help NonCOP.Ang2Q for procedures which treat line and area detectors

Parameters

om,chi,phi,ttsample and detector angles as numpy array, lists or Scalars must be given. all arguments must have the same shape or length

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of length 4. used angles are than om,chi,phi,tt - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

Q2Ang (**Q*, ***keyargs*)

Convert a reciprocal space vector Q to NON-COPLANAR scattering angles. The keyword argument trans determines whether Q should be transformed to the experimental coordinate frame or not.

Parameters

Qa list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz

optional keyword arguments:

transTrue/False apply coordinate transformation on Q (default True)

degTrue/Flase (default True) determines if the angles are returned in radians or degree

Returns

a numpy array of shape (4) with four scattering angles which are [omega,chi,phi,twotheta]

omegasample rocking angle

chisample tilt

phisample azimuth

twothetascattering angle (detector)

class xrutils.experiment.**Powder** (*mat*, ***keyargs*)

Bases: xrutils.experiment.Experiment

Experimental class for powder diffraction This class is able to simulate a powder spectrum for the given material

Convolute (*stepwidth*, *width*, *min=0*, *max=None*)

Convolutes the intensity positions with Gaussians with width in momentum space of "width". returns array of angular positions with corresponding intensity

thetaarray with angular positions

intintensity at the positions ttheta

PowderIntensity (*tt_cutoff=180*)

Calculates the powder intensity and positions up to an angle of tt_cutoff (deg) and stores the result in:

dataarray with intensities

angangular position of intensities

qposreciprocal space position of intensities

Q2Ang (*qpos*, *deg=True*)

Converts reciprocal space values to theta angles

class xrutils.experiment.**QConversion** (*sampleAxis*, *detectorAxis*, *r_i*, ***kwargs*)

Bases: object

Class for the conversion of angular coordinates to momentum space for arbitrary goniometer geometries

the class is configured with the initialization and does provide three distinct routines for conversion to momentum space for

* point detector: point(...) or __call__() * linear detector: linear(...) * area detector: area(...)

linear() and area() can only be used after the init_linear() or init_area() routines were called

UB

area(**args*, ***kwargs*)

angular to momentum space conversion for a area detector the center pixel defined by the init_area routine must be in direction of self.r_i when detector angles are zero

the detector geometry must be initialized by the init_area(...) routine

Parameters

***args: sample and detector angles as numpy array, lists or Scalars**

in total len(self.sampleAxis)+len(detectorAxis) must be given always starting with the outer most circle all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: possible keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

roiregion of interest for the detector pixels; e.g. [100,900,200,800] :(default: self._area_roi)

Navnumber of channels to average to reduce data size e.g. [2,2] :(default: self._area_nav)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape ((*)*(self._area_roi[1]-self._area_roi[0]+1)*(self._area_roi[3]-self._area_roi[2]+1) , 3) were detectorDir1 is the fastest varing

detectorAxis

property handler for _detectorAxis

Returns

list of detector axis following the syntax /[xyz][+-]/

energy

init_area(*detectorDir1*, *detectorDir2*, *cch1*, *cch2*, *Nch1*, *Nch2*, *distance=None*, *pwidth1=None*, *pwidth2=None*, *chpdeg1=None*, *chpdeg2=None*, *detrot=0*, *tiltazimuth=0*, *tilt=0*, ***kwargs*)

initialization routine for area detectors detector direction as well as distance and pixel size or channels per degree must be given. Two separate pixel sizes and channels per degree for the two orthogonal directions can be given

Parameters

detectorDir1direction of the detector (along the pixel direction 1); e.g. 'z+' means higher pixel numbers at larger z positions

detectorDir2direction of the detector (along the pixel direction 2); e.g. 'x+'

cch1,2center pixel, in direction of self.r_i at zero detectorAngles

Nch1number of detector pixels along direction 1

Nch2number of detector pixels along direction 2

distancedistance of center pixel from center of rotation

pwidth1,2width of one pixel (same unit as distance)

chpdeg1,2channels per degree (only absolute value is relevant) sign determined through detectorDir1,2

detrotdetector rotation around primary beam direction

tiltazimuthdirection of the tilt vector in the detector plane (in degree)

tilttilt of the detector plane around an axis normal to the direction given by the tiltazimuth

---

Note: Note: Either distance and pwidth1,2 or chpdeg1,2 must be given !!

****kwargs: optional keyword arguments**

Navnumber of channels to average to reduce data size (default: [1,1])

roiregion of interest for the detector pixels; e.g. [100,900,200,800]

init_linear(*detectorDir*, *cch*, *Nchannel*, *distance=None*, *pixelwidth=None*, *chpdeg=None*, *tilt=0*, ***kwargs*)

initialization routine for linear detectors detector direction as well as distance and pixel size or channels per degree must be given.

Parameters

detectorDirdirection of the detector (along the pixel array); e.g. 'z+'

cchcenter channel, in direction of self.r_i at zero detectorAngles

Nchanneltotal number of detector channels

distancedistance of center channel from center of rotation

pixelwidthwidth of one pixel (same unit as distance)

chpdegchannels per degree (only absolute value is relevant) sign determined through detectorDir

!! Either distance and pixelwidth or chpdeg must be given !!

tilttilt of the detector axis from the detectorDir (in degree)

****kwargs: optional keyword arguments**

Navnumber of channels to average to reduce data size (default: 1)

roiregion of interest for the detector pixels; e.g. [100,900]

linear(**args*, ***kwargs*)

angular to momentum space conversion for a linear detector the cch of the detector must be in direction of self.r_i when detector angles are zero

the detector geometry must be initialized by the init_linear(...) routine

Parameters

***args: sample and detector angles as numpy array, lists or Scalars**

in total len(self.sampleAxis)+len(detectorAxis) must be given always starting with the outer most circle all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: possible keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

Navnumber of channels to average to reduce data size (default: self._linear_nav)

roiregion of interest for the detector pixels; e.g. [100,900] (default: self._linear_roi)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree (default: True)

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape ((*)*(self._linear_roi[1]-self._linear_roi[0]+1) , 3)

point (**args*, ***kwargs*)

angular to momentum space conversion for a point detector located in direction of self.r_i when detector angles are zero

Parameters

***args: sample and detector angles as numpy array, lists**

or Scalars in total len(self.sampleAxis)+len(detectorAxis) must be given, always starting with the outer most circle. all arguments must have the same shape or length

sAnglessample circle angles, number of arguments must correspond to len(self.sampleAxis)

dAnglesdetector circle angles, number of arguments must correspond to len(self.detectorAxis)

****kwargs: optional keyword arguments**

deltagiving delta angles to correct the given ones for misalignment delta must be an numpy array or list of len(*args) used angles are than *args - delta

UBmatrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)

wlx-ray wavelength in angstroem (default: self._wl)

degflag to tell if angles are passed as degree :(default: True)

Returns

reciprocal space positions as numpy.ndarray with shape (* , 3) where * corresponds to the number of points given in the input

sampleAxis

property handler for _sampleAxis

Returns

list of sample axis following the syntax /[xyzk][+-]/

wavelength

gridder Module

class xrutils.gridder.**Gridder**(***keyargs*)
Bases: object

KeepData(*bool*)

Normalize(*bool*)

SetChunkSize(*n*)

SetChunkUnit(*u*)

SetThreads(*n*)

class xrutils.gridder.**Gridder1D**(*nx*, ***keyargs*)
Bases: xrutils.gridder.Gridder

Clear()

data

xaxis

class xrutils.gridder.**Gridder2D**(*nx*, *ny*, ***keyargs*)
Bases: xrutils.gridder.Gridder

Clear()

SetResolution(*nx*, *ny*)

data

xaxis

xmatrix

yaxis

ymatrix

class xrutils.gridder.**Gridder3D**(*nx*, *ny*, *nz*, ***keyargs*)
Bases: xrutils.gridder.Gridder2D

SetResolution(*nx*, *ny*, *nz*)

zaxis

zmatrix

libxrayutils Module

this module uses the ctypes package to provide access to the functions implemented in the libxrayutils C library.

the functions provided by this module are low level. Users should use the derived functions in the corresponding submodules

normalize Module

module to provide functions that perform block averaging of intensity arrays to reduce the amount of data (mainly for PSD and CCD measurements

and

provide functions for normalizing intensities for

* count time * absorber (user-defined function) * monitor * flatfield correction

class xrutils.normalize.**IntensityNormalizer**(*det*, ***keyargs*)

Bases: object

generic class for correction of intensity (point detector,or MCA, single CCD frames) for count time and absorber factors the class must be supplied with a absorber correction function and works with data structures provided by xrutils.io classes or the corresponding objects from hdf5 files read by pytables

absfun
absfun property handler returns the costum correction function or None

avmon
av_mon property handler returns the value of the average monitor or None if average is calculated from the monitor field

darkfield
flatfield property handler returns the current set darkfield of the detector or None if not set

det
det property handler returns the detector field name

flatfield
flatfield property handler returns the current set flatfield of the detector or None if not set

mon
mon property handler returns the monitor field name or None if not set

time
time property handler returns the count time or the field name of the count time or None if time is not set

xrutils.normalize.**blockAverage1D**(*data*, *Nav*)

perform block average for 1D array/list of Scalar values all data are used. at the end of the array a smaller cell may be used by the averaging algorithm

Parameter

datadata which should be contracted (length N)

Navnumber of values which should be averaged

Returns

block averaged numpy array of data type numpy.double (length ceil(N/Nav))

xrutils.normalize.**blockAverage2D**(*data2d*, *Nav1*, *Nav2*, ***kwargs*)

perform a block average for 2D array of Scalar values all data are used therefore the margin cells may differ in size

Parameter

data2darray of 2D data shape (N,M)

Nav1,2a field of (Nav1 x Nav2) values is contracted

****kwargs: optional keyword argument**

roiregion of interest for the 2D array. e.g. [20,980,40,960] N = 980-20; M = 960-40

Returns

block averaged numpy array with type numpy.double with shape (ceil(N/Nav1), ceil(M/Nav2))

xrutils.normalize.**blockAveragePSD**(*psddata*, *Nav*, ***kwargs*)

perform a block average for serveral PSD spectra all data are used therefore the last cell used for averaging may differ in size

Parameter

psddataarray of 2D data shape (Nspectra,Nchannels)

Navnumber of channels which should be averaged

****kwargs: optional keyword argument**

roiregion of interest for the 2D array. e.g. [20,980] Nchannels = 980-20

Returns

block averaged psd spectra as numpy array with type numpy.double of shape (Nspectra , ceil(Nchannels/Nav))

`utilities` Module

xrutils utilities contains a conglomeration of useful functions which do not fit into one of the other files

`xrutils.utilities.`**`maplog`**(*inte*, *dynlow='config'*, *dynhigh='config'*, ***keyargs*)

clips values smaller and larger as the given bounds and returns the log10 of the input array. The bounds are given as exponent with base 10 with respect to the maximum in the input array. The function is implemented in analogy to J. Stangl's matlab implementation.

Parameters

intenumpy.array, values to be cut in range

dynlow10^(-dynlow) will be the minimum cut off

dynhigh10^(-dynhigh) will be the maximum cut off

optional keyword arguments (NOT IMPLEMENTED):

abslow10^(abslow) will be taken as lower boundary

abshigh10^(abshigh) will be taken as higher boundary

Returns

numpy.array of the same shape as inte, where values smaller/larger then 10^(-dynlow,dynhigh) were replaced by 10^(-dynlow,dynhigh)

Example

```
>>> lint = maplog(int,5,2)
```

`utilities_noconf` Module

xrutils utilities contains a conglomeration of useful functions this part of utilities does not need the config class

`xrutils.utilities_noconf.`**`energy`**(*en*)

convert common energy names to energies in eV

so far this works with CuKa1, CuKa2, CuKa12, CuKb, MoKa1

Parameter

enenergy (scalar (energy in eV will be returned unchanged) or string with name of emission line)

Returns

energy in eV as float

`xrutils.utilities_noconf.`**`lam2en`**(*inp*)

converts the input energy in eV to a wavelength in Angstrom or the input wavelength in Angstrom to an energy in eV

Parameter

inpeither an energy in eV or an wavelength in Angstrom

Returns

float, energy in eV or wavlength in Angstrom

Examples

```
>>> lambda = lam2en(8048)
>>> energy = lam2en(1.5406)
```

xrutils.utilities_noconf.**wavelength**(*wl*)

convert common energy names to energies in eV

so far this works with CuKa1, CuKa2, CuKa12, CuKb, MoKa1

Parameter

wlwavelength (scalar (wavelength in Angstrom will be returned unchanged) or string with name of emission line)

Returns

wavelength in Angstrom as float

Subpackages

analysis Package xrutils.analysis is a package for assisting with the analysis of x-ray diffraction data, mainly reciprocal space maps

Routines for obtaining line cuts from gridded reciprocal space maps are offered, with the ability to integrate the intensity perpendicular to the line cut direction.

line_cuts Module

xrutils.analysis.line_cuts.**fwhm_exp**(*pos*, *data*)

function to determine the full width at half maximum value of experimental data. Please check the obtained value visually (noise influences the result)

Parameter

posposition of the data points

datadata values

Returns

fwhm value (single float)

xrutils.analysis.line_cuts.**get_omega_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)

extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_omega_scan_bounds_ang**(*omcenter*, *ttcenter*, *omrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of omega scan

Parameters

omcenteromega-position at which the omega scan should be extracted

ttcenter2theta-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qx,qzreciprocal space coordinates of the omega scan boundaries

Example

```
>>> qxb,qzb = get_omega_scan_bounds_ang(1.0,4.0,2.4,240,qrange=0.1)
```

xrutils.analysis.line_cuts.**get_omega_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *omrange*, *npoints*, ***kwargs*)

extracts an omega scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the omega scan should be extracted

qzcenterqz-position at which the omega scan should be extracted

omrangerange of the omega scan to extract

npointsnumber of points of the omega scan

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative omega positions are returned (default: True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,omintomega scan coordinates and intensities (bounds=False)

om,omint,(qxb,qzb)omega scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omcut, intcut = get_omega_scan(qx,qz,intensity,0.0,5.0,2.0,200)
```

xrutils.analysis.line_cuts.**get_qx_scan**(*qx*, *qz*, *intensity*, *qzpos*, ***kwargs*)

extract qx line scan at position qzpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qz

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

Returns

qx,qxintqx scan coordinates and intensities (bounds=False)

qx,qxint,(qxb,qyb)qx scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qxcut,qxcut_int = get_qx_scan(qx,qz,inten,5.0,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extract qz line scan at position qxpos from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given range along qx

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan(qx,qz,inten,1.5,qrange=0.03)
```

xrutils.analysis.line_cuts.**get_qz_scan_int**(*qx*, *qz*, *intensity*, *qxpos*, ***kwargs*)

extracts a qz scan from a gridded reciprocal space map with integration along omega (sample rocking angle) or 2theta direction

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

angrangeintegration range in angular direction

qmin,qmaxminimum and maximum value of extracted scan axis

boundsflag to specify if the scan bounds of the extracted scan should be returned (default:False)

intdirintegration direction 'omega': sample rocking angle (default) '2theta': scattering angle

Returns

qz,qzintqz scan coordinates and intensities (bounds=False)

qz,qzint,(qzb,qzb)qz scan coordinates and intensities + scan bounds for plotting

Example

```
>>> qzcut,qzcut_int = get_qz_scan_int(qx,qz,inten,5.0,omrange=0.3)
```

xrutils.analysis.line_cuts.**get_radial_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_ang(qx,qz,intensity,32.0,64.0,30.0,800,omrange=0.2)
```

xrutils.analysis.line_cuts.**get_radial_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of radial scan

Parameters

omcenterom-position at which the radial scan should be extracted

ttcentertt-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxrad,qzradreciprocal space boundaries of radial scan

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_radial_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a radial scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the radial scan should be extracted

qzcenterqz-position at which the radial scan should be extracted

ttrangetwo theta range of the radial scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range perpendicular to scan direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

om,tt,radintomega,two theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> omc,ttc,cut_int = get_radial_scan_q(qx,qz,intensity,0.0,5.0,1.0,100,omrange=0.01)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_ang**(*qx*, *qz*, *intensity*, *omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_ang(qx,qz,intensity,32.0,64.0,4.0,400)
```

xrutils.analysis.line_cuts.**get_ttheta_scan_bounds_ang**(*omcenter*, *ttcenter*, *ttrange*, *npoints*, ***kwargs*)

return reciprocal space boundaries of 2theta scan

Parameters

omcenterom-position at which the 2theta scan should be extracted

ttcentertt-position at which the 2theta scan should be extracted

ttrangetwo theta range of the 2theta scan to extract

npointsnumber of points of the 2theta scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

lamwavelength for use in the conversion to angular coordinates

Returns

qxtt,qzttreciprocal space boundaries of 2theta scan (bounds=False)

tt,ttint,(qxb,qzb)2theta scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>>
```

xrutils.analysis.line_cuts.**get_ttheta_scan_q**(*qx*, *qz*, *intensity*, *qxcenter*, *qzcenter*, *ttrange*, *npoints*, ***kwargs*)

extracts a twotheta scan from a gridded reciprocal space map

Parameters

qxequidistant array of qx momentum transfer

qzequidistant array of qz momentum transfer

intensity2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxcenterqx-position at which the 2theta scan should be extracted

qzcenterqz-position at which the 2theta scan should be extracted

ttrangetwo theta range of the scan to extract

npointsnumber of points of the radial scan

****kwargs: possible keyword arguments:**

omrangeintegration range in omega direction

Nintnumber of subscans used for the integration (optionally)

lamwavelength for use in the conversion to angular coordinates

relativedetermines if absolute or relative two theta positions are returned (default=True)

boundsflag to specify if the scan bounds should be returned (default: False)

Returns

tt,ttinttwo theta scan coordinates and intensities (bounds=False)

om,tt,radint,(qxb,qzb)radial scan coordinates and intensities + reciprocal space bounds of the extraced scan (bounds=True)

Example

```
>>> ttc,cut_int = get_ttheta_scan_q(qx,qz,intensity,0.0,4.0,4.4,440)
```

xrutils.analysis.line_cuts.**getindex**(*x*, *y*, *xgrid*, *ygrid*)

gives the indices of the point x,y in the grid given by xgrid ygrid xgrid,ygrid must be arrays containing equidistant points

Parameters

x,ycoordinates of the point of interest (float)

xgrid,ygridgrid coordinates in x and y direction (array)

Returns

ix,iyindex of the closest gridpoint (lower left) of the point (x,y)

line_cuts3d Module

xrutils.analysis.line_cuts3d.**get_qx_scan3d**(*gridder*, *qypos*, *qzpos*, ***kwargs*)

extract qx line scan at position y,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qypos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qx,qxintqx scan coordinates and intensities

Example

```
>>> qxcut,qxcut_int = get_qx_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qy_scan3d**(*gridder*, *qxpos*, *qzpos*, ***kwargs*)

extract qy line scan at position x,z from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qzposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qy,qyintqy scan coordinates and intensities

Example

```
>>> qycut,qycut_int = get_qy_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**get_qz_scan3d**(*gridder*, *qxpos*, *qypos*, ***kwargs*)

extract qz line scan at position x,y from a gridded reciprocal space map by taking the closest line of the intensity matrix, or summing up a given area around this position

Parameters

gridder3d xrutils.Gridder3D object containing the data

qxpos,qyposposition at which the line scan should be extracted

****kwargs: possible keyword arguments:**

qrangeintegration range perpendicular to scan direction

qmin,qmaxminimum and maximum value of extracted scan axis

Returns

qz,qzintqz scan coordinates and intensities

Example

```
>>> qzcut,qzcut_int = get_qz_scan3d(gridder,0,0,qrange=0.03)
```

xrutils.analysis.line_cuts3d.**getindex3d**(*x*, *y*, *z*, *xgrid*, *ygrid*, *zgrid*)
gives the indices of the point x,y,z in the grid given by xgrid ygrid zgrid xgrid,ygrid,zgrid must be arrays containing equidistant points

Parameters

x,y,zcoordinates of the point of interest (float)

xgrid,ygrid,zgridgrid coordinates in x,y,z direction (array)

Returns

ix,iy,izindex of the closest gridpoint (lower left) of the point (x,y,z)

misc Module miscellaneous functions helpful in the analysis and experiment

xrutils.analysis.misc.**getangles**(*peak*, *sur*, *inp*)
calculates the chi and phi angles for a given peak

Parameter

peakarray which gives hkl for the peak of interest

surhkl of the surface

inpinplane reference peak or direction

Returns

[chi,phi] for the given peak on surface sur with inplane direction inp as reference

Example

To get the angles for the -224 peak on a 111 surface type[chi,phi] = getangles([-2,2,4],[1,1,1],[2,2,4])

sample_align Module functions to help with experimental alignment during experiments, especially for experiments with linear detectors

xrutils.analysis.sample_align.**area_detector_calib**(*angle1*, *angle2*, *ccdimages*, *detaxis*, *r_i*, *plot=True*, *cut_off=0.7*, *start=(0, 0, 0, 0)*, *fix=(False, False, False, False)*, *fig=None*, *wl=None*)
function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

parameters

angle1outer detector arm angle

angle2inner detector arm angle

ccdimagesimages of the ccd taken at the angles given above

detaxisdetector arm rotation axis :default: ['z+','y-']

r_iprimary beam direction [xyz][+-] default 'x+'

Keyword_arguments

plotflag to determine if results and intermediate results should be plotted :default: True

cut_offcut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%

startsequence of start values of the fit for parameters, which can not be estimated automatically these are: tiltazimuth,tilt,detector_rotation,outerangle_offset. By default (0,0,0,0) is used.

fixfix parameters of start (default: (False,False,False,False))

figmatplotlib figure used for plotting the error :default: None (creates own figure)

wlwavelength of the experiment in Angstrom (default: config.WAVELENGTH) value does not matter here and does only affect the scaling of the error

xrutils.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03*, *0.03)*, *plot=True*)

helper function to determine the Bragg peak position in a reciprocal space map used to obtain the position needed for correction of the data. the determination is done by fitting a two dimensional Gaussian (xrutils.math.Gauss2d)

PLEASE ALWAYS CHECK THE RESULT CAREFULLY!

Parameter

om,ttangular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]

psdintensity values needed for fitting

omalignaligned omega value, used as first guess in the fit

ttalignaligned two theta values used as first guess in the fit these values are also used to set the range for the fit: the peak should be within +/-frangeAA^{-1} of those values

exphxrdexperiment class used for the conversion between angular and reciprocal space.

frangedata range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})

plotif True (default) function will plot the result of the fit in comparison with the measurement.

Returns

Omfit,ttfit,params,covariance fitted angular values, and the fit parameters (of the Gaussian) as well as their errors

xrutils.analysis.sample_align.**linear_detector_calib**(*angle*, *mca_spectra*, ***keyargs*)

function to calibrate the detector distance/channel per degrees for a straigt linear detector mounted on a detector arm

parameters

anglearray of angles in degree of measured detector spectra

mca_spectracorresponding detector spectra :(shape: (len(angle),Nchannels)

****keyargs passed to psd_chdeg function used for the modelling additional options:**

r_iprimary beam direction as vector [xyz][+-]; default: 'y+'

detaxisdetector rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

returns

L/pixelwidth*pi/180 ~= channel/degree, center_channel[, detector_tilt]

The function also prints out how a linear detector can be initialized using the results obtained from this calibration.

Note: Note: distance of the detector is given by: channel_width*channelperdegree/tan(radians(1))

---

xrutils.analysis.sample_align.**miscut_calc**(*phi*, *aomega*, *zeros=None*, *plot=True*, *omega0=None*)

function to calculate the miscut direction and miscut angle of a sample by fitting a sinusoidal function to the variation of the aligned omega values of more than two reflections. The function can also be used to fit reflectivity alignment values in various azimuths.

Parameters

phiazimuths in which the reflection was aligned (deg)

aomegaaligned omega values (deg)

zeros(optional) angles at which surface is parallel to the beam (deg). For the analysis the angles (aomega-zeros) are used.

plotflag to specify if a visualization of the fit is wanted. :default: True

omega0if specified the nominal value of the reflection is not included as fit parameter, but is fixed to the specified value. This value is MANDATORY if ONLY TWO AZIMUTHs are given.

Returns

[omega0,phi0,miscut]

list with fitted values for

omega0the omega value of the reflection should be close to the nominal one

phi0the azimuth in which the primary beam looks upstairs

miscutamplitude of the sinusoidal variation == miscut angle

xrutils.analysis.sample_align.**psd_chdeg**(*angles*, *channels*, *stdev=None*, *usetilt=False*, *plot=True*)

function to determine the channels per degree using a linear fit of the function nchannel = center_ch+chdeg*tan(angles) or the equivalent including a detector tilt

Parameters

anglesdetector angles for which the position of the beam was measured

channelsdetector channels where the beam was found

keyword arguments:

stdevstandard deviation of the beam position

plotflag to specify if a visualization of the fit should be done

usetiltwhether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: False)

Returns (L/pixelwidth*pi/180 ,centerch[,tilt]):

L/pixelwidth*pi/180 = channel/degree for large detector distance with L sample detector disctance, and pixelwidth the width of one detector channel

Centerch center channel of the detector

Tilt tilt of the detector from perpendicular to the beam

Note: Note: distance of the detector is given by: channelwidth*channelperdegree/tan(radians(1))

xrutils.analysis.sample_align.**psd_refl_align**(*primarybeam*, *angles*, *channels*, *plot=True*)

function which calculates the angle at which the sample is parallel to the beam from various angles and detector channels from the reflected beam. The function can be used during the half beam alignment with a linear detector.

Parameters

primarybeamprimary beam channel number

angleslist or numpy.array with angles

channelslist or numpy.array with corresponding detector channels

plotflag to specify if a visualization of the fit is wanted :default: True

Returns

omegaangle at which the sample is parallel to the beam

Example

```
>>> psd_refl_align(500,[0,0.1,0.2,0.3],[550,600,640,700])
```

io Package

cif Module

class xrutils.io.cif.**CIFFile**(*filename*)

Bases: object

class for parsing CIF (Crystallographic Information File) files. The class aims to provide an additional way of creating material classes instead of manual entering of the information the lattice constants and unit cell structure are parsed from the CIF file

Lattice()

returns a lattice object with the structure from the CIF file

Parse()

function to parse a CIF file. The function reads the space group symmetry operations and the basic atom positions as well as the lattice constants and unit cell angles

SymStruct()

function to obtain the list of different atom positions in the unit cell for the different types of atoms. The data are obtained from the data parsed from the CIF file.

edf Module

class xrutils.io.edf.**EDFDirectory**(*datapath*, ***keyargs*)

Bases: object

Parses a directory for EDF files, which can be stored to a HDF5 file for further usage

Save2HDF5(*h5*, ***keyargs*)

Save2HDF5(h5,**keyargs): Saves the data stored in the EDF files in the specified directory in a HDF5 file as a HDF5 arrays in a subgroup. By default the data is stored in a group given by the foldername - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data (default: pathname) :comp: activate compression - true by default

class xrutils.io.edf.**EDFFile**(*fname*, ***keyargs*)
Bases: object

ReadData()
Read the CCD data into the .data object this function is called by the initialization

Save2HDF5(*h5*, ***keyargs*)
Save2HDF5(h5,**keyargs): Saves the data stored in the EDF file in a HDF5 file as a HDF5 array. By default the data is stored in the root group of the HDF5 file - this can be changed by passing the name of a target group or a path to the target group via the "group" keyword argument.

required arguments. :h5: a HDF5 file object

optional keyword arguments: :group: group where to store the data :comp: activate compression - true by default

imagereader Module

class xrutils.io.imagereader.**ImageReader**(*nop1*, *nop2*, *hdrlen=0*, *flatfield=None*, *darkfield=None*, *dtype=<type 'numpy.int16'>*, *byte_swap=False*)
Bases: object

parse CCD frames in the form of tiffs or binary data (*.bin) to numpy arrays. ignore the header since it seems to contain no useful data

The routine was tested so far with RoperScientific files with 4096x4096 pixels created at Hasylab Hamburg, which save an 16bit integer per point. Perkin Elmer images created at Hasylab Hamburg with 2048x2048 pixels.

readImage(*filename*)
read image file and correct for dark- and flatfield in case the necessary data are available.

returned data = ((image data)-(darkfield))/flatfield*average(flatfield)

Parameter

filenamefilename of the image to be read. so far only single filenames are supported. The data might be compressed. supported extensions: .tiff, .bin and .bin.xz

class xrutils.io.imagereader.**PerkinElmer**(***keyargs*)
Bases: xrutils.io.imagereader.ImageReader

parse PerkinElmer CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 2048x2048 pixel images created at Hasylab Hamburg which save an 32bit float per point.

class xrutils.io.imagereader.**RoperCCD**(***keyargs*)
Bases: xrutils.io.imagereader.ImageReader

parse RoperScientific CCD frames (*.bin) to numpy arrays Ignore the header since it seems to contain no useful data

The routine was tested only for files with 4096x4096 pixel images created at Hasylab Hamburg which save an 16bit integer per point.

panalytical_xml Module Panalytical XML (www.XRDML.com) data file parser

based on the native python xml.dom.minidom module. want to keep the number of dependancies as small as possible

class xrutils.io.panalytical_xml.**XRDMLFile**(*fname*)
Bases: object

class to handle XRDML data files. The class is supplied with a file name and uses the XRDMLScan class to parse the xrdMeasurement in the file

class xrutils.io.panalytical_xml.**XRDMLMeasurement**(*measurement*)
Bases: object

class to handle scans in a XRDML datafile

xrutils.io.panalytical_xml.**getOmPixcel**(*omraw*, *ttraw*)
function to reshape the Omega values into a form needed for further treatment with xrutils

xrutils.io.panalytical_xml.**getxrdml_map**(*filetemplate*, *scannrs=None*, *path='.'*, *roi=None*)
parses multiple XRDML file and concatenates the results for parsing the xrutils.io.XRDMLFile class is used. The function can be used for parsing maps measured with the PIXCel and point detector.

Parameter

filetemplatetemplate string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames

scannrsint or list of scan numbers

pathcommon path to the filenames

roiregion of interest for the PIXCel detector, for other measurements this i not usefull!

Returns

om,tt,psdas flattened numpy arrays

Example

```
>>> om,tt,psd = xrutils.io.getxrdml_map("samplename_%d.xrdml",[1,2],path="./data")
```

radicon Module python module for converting radicon data to HDF5

xrutils.io.radicon.**hst2hdf5**(*h5*, *hstfile*, *nofchannels*, ***keyargs*)
Converts a HST file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

hstfilename of the HST file

nofchannelsnumber of channels

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

hstpathpath where the HST file is located (default is the current working directory)

xrutils.io.radicon.**rad2hdf5**(*h5*, *rdcfile*, ***keyargs*)
Converts a RDC file to an HDF5 file.

Required input arguments:

h5HDF5 object where to store the data

rdcfilename of the RDC file

optional (named) input arguments:

h5path Path in the HDF5 file where to store the data

rdcpathpath where the RDC file is located (default is the current working directory)

xrutils.io.radicon.**selecthst**(*et_limit*, *mca_info*, *mca_array*)
Select historgrams form the complete set of recorded MCA data and stores it into a new numpy array. The selection is done due to a exposure time limit. Spectra below this limit are ignored.

required input arguments:

et_limitexposure time limit

mca_infopytables table with the exposure data

mca_arrayarray with all the MCA spectra

return value:a numpy array with the selected mca spectra of shape (hstnr,channels).

rotanode_alignment Module parser for the alignment log file of the rotating anode

class xrutils.io.rotanode_alignment.**RA_Alignment**(*filename*)
Bases: object

class to parse the data file created by the alignment routine (tpalign) at the rotating anode spec installation

this routine does an iterative alignment procedure and saves the center of mass values were it moves after each scan. It iterates between two different peaks and iteratively aligns at each peak between two different motors (om/chi at symmetric peaks, om/phi at asymmetric peaks)

Parse()
parser to read the alignment log and obtain the aligned values at every iteration.

get(*key*)

keys()
returns a list of keys for which aligned values were parsed

plot(*pname*)
function to plot the alignment history for a given peak

Parameters

pnamepeakname for which the alignment should be plotted

seifert Module a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two posibilities how the data is stored (depending on the use detector):

1. as a simple line scan (using the point detector)
2. as a map using the PSD

In the first case the data ist stored

class xrutils.io.seifert.**SeifertHeader**
Bases: object

save_h5_attribs(*obj*)

class xrutils.io.seifert.**SeifertMultiScan**(*filename*, *m_scan*, *m2*)
Bases: object

dump2hdf5(*h5*, **args*, ***keyargs*)
Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument "group" must be used.

required arguments:

h5a HDF5 file object

optional positional arguments:name for the intensity matrix name for the scan motor name for the second motor more then three parameters are ignored.

optional keyword arguments:

grouppath to the HDF5 group where to store the data

dump2mlab(*fname*, **args*)
Store the data in a matlab file.

parse()

class xrutils.io.seifert.**SeifertScan**(*filename*)
Bases: object

dump2h5(*h5*, **args*, ***keyargs*)
Save the data stored in the Seifert ASCII file to a HDF5 file.

required input arguments:

h5HDF5 file object

optional arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

optional keyword arguments:

groupHDF5 group object where to store the data.

dump2mlab(*fname*, **args*)
Save the data from a Seifert scan to a matlab file.

required input arugments:

fnamename of the matlab file

optional position arguments:

names to use to store the motors. The first must be the name for the intensity array. The number of names must be equal to the second element of the shape of the data object.

parse()

xrutils.io.seifert.**repair_key**(*key*)
Repair a key string in the sense that the string is changed in a way that it can be used as a valid Python identifier. For that purpose all blanks within the string will be replaced by _ and leading numbers get an preceeding _.

spec Module a threaded class for observing a SPEC data file

Motivation

SPEC files can become quite large. Therefore, subsequently reading the entire file to extract a single scan is a quite cumbersome procedure. This module is a proof of concept code to write a file observer starting a reread of the file starting from a stored offset (last known scan position)

class xrutils.io.spec.**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)
Bases: object

Save2HDF5(*h5*, ***keyargs*)

class xrutils.io.spec.**SPECFile**(*filename*, ***keyargs*)
Bases: object

This class represents a single SPEC file. The class provides methodes for updateing an already opened file which makes it particular interesting for interactive use.

Parse()
Parses the file from the starting at last_offset and adding found scans to the scan list.

Save2HDF5(*h5f*, ***keyargs*)
Save the entire file in an HDF5 file. For that purpose a group is set up in the root group of the file with the name of the file without extension and leading path. If the method is called after an previous update only the scans not written to the file meanwhile are saved.

required arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

compactivate compression - true by default

nameoptional name for the file group

Update()
reread the file and add newly added files. The parsing starts at the data offset of the last scan gathered during the last parsing run.

class xrutils.io.spec.**SPECLog**(*filename*, *prompt*, ***keyargs*)
Bases: object

Parse()

Update()

class xrutils.io.spec.**SPECMCA**(*nchan*, *roistart*, *roistop*)
Bases: object

SPECMCA - represents an MCA object in a SPEC file. This class is an abstract class not itended for being used directly. Instead use one of the derived classes SPECMCAFile or SPECMCAInline.

class xrutils.io.spec.**SPECMCAFile**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECMCAInline**
Bases: xrutils.io.spec.SPECMCA

ReadData()

class xrutils.io.spec.**SPECScan**(*name*, *scannr*, *command*, *date*, *time*, *itime*, *colnames*, *hoffset*, *doffset*, *fid*, *imopnames*, *imopvalues*, *scan_status*)
Bases: object

Represents a single SPEC scan.

ClearData()
Delete the data stored in a scan after it is no longer used.

ReadData()
Set the data attribute of the scan class.

Save2HDF5(*h5f*, ***keyargs*)
Save a SPEC scan to an HDF5 file. The method creates a group with the name of the scan and stores the data there as a table object with name "data". By default the scan group is created under the root group of the HDF5 file. The title of the scan group is ususally the scan command. Metadata of the scan are stored as attributes to the scan group. Additional custom attributes to the scan group can be passed as a dictionary via the optattrs keyword argument.

input arguments:

h5fa HDF5 file object or its filename

optional keyword arguments:

groupname or group object of the HDF5 group where to store the data

titlea string with the title for the data

desca string with the description of the data

optattrsa dictionary with optional attributes to store for the data

compactivate compression - true by default

SetMCAParams(*mca_column_format*, *mca_channels*, *mca_start*, *mca_stop*)
Set the parameters used to save the MCA data to the file. This method calculates the number of lines used to store the MCA data from the number of columns and the

required input aguments:

mca_column_formatnumber of columns used to save the data

mca_channelsnumber of MCA channels stored

mca_startfirst channel that is stored

mca_stoplast channel that is stored

plot(**args*, ***keyargs*)

Plot scan data to a matplotlib figure. If newfig=True a new figure instance will be created. If logy=True (default is False) the y-axis will be plotted with a logarithmic scale.

xrutils.io.spec.**geth5_scan**(*h5f*, *scans*, **args*, ***kwargs*)

function to obtain the angular cooridinates as well as intensity values saved in an HDF5 file, which was created from a spec file by the Save2HDF5 method. Especially usefull for reciprocal space map measurements.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables or its filename

scansnumber of the scans of the reciprocal space map (int,tuple or list)

*args: names of the motors (optional) (strings) to read reciprocal space maps measured in coplanar diffraction give: :omname: e.g. name of the omega motor (or its equivalent) :ttname: e.g. name of the two theta motor (or its equivalent)

****kwargs (optional):**

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used

Returns

MAP

or

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

Example

```
>>> [om,tt],MAP = xu.io.geth5_scan(h5file,36,'omega','gamma')
```

spectra Module module to handle spectra data

class xrutils.io.spectra.**SPECTRAFile**(*filename*, *mcatmp=None*, *mcastart=None*, *mcastop=None*)

Bases: object

Represents a SPECTRA data file. The file is read during the Constructor call. This class should work for data stored at beamlines P08 and BW2 at HASYLAB.

Required constructor arguments:

filenamea string with the name of the SPECTRA file

Optional keyword arguments:

mcatmptemplate for the MCA files

mcastart,mcastopstart and stop index for the MCA files, if not given, the class tries to determine the start and stop index automatically.

Read()

Read the data from the file.

ReadMCA()

Save2HDF5(*h5file*, *name*, *group='/'*, *description='SPECTRA scan'*, *mcaname='MCA'*)
Saves the scan to an HDF5 file. The scan is saved to a seperate group of name "name". h5file is either a string for the file name or a HDF5 file object. If the mca attribute is not None mca data will be stored to an chunked array of with name mcaname.

required input arguments:

h5filestring or HDF5 file object

namename of the group where to store the data

optional keyword arguments:

grouproot group where to store the data

descriptionstring with a description of the scan

Return value: The method returns None in the case of everything went fine, True otherwise.

class xrutils.io.spectra.**SPECTRAFileComments**
Bases: dict

Class that describes the comments in the header of a SPECTRA file. The different comments are accessible via the comment keys.

class xrutils.io.spectra.**SPECTRAFileData**
Bases: object

append(*col*)

class xrutils.io.spectra.**SPECTRAFileDataColumn**(*index*, *name*, *unit*, *type*)
Bases: object

class xrutils.io.spectra.**SPECTRAFileParameters**
Bases: dict

class xrutils.io.spectra.**Spectra**(*data_dir*)
Bases: object

abs_corr(*data*, *f*, ***keyargs*)
Perform absorber correction. Data can be either a 1 dimensional data (point detector) or a 2D MCA array. In the case of an array the data array should be of shape (N,NChannels) where N is the number of points in the scan an NChannels the number of channels of the MCA. The absorber values are passed to the function as a 1D array of N elements.

By default the absorber values are taken form a global variable stored in the module called _absorber_factors. Despite this, costume values can be passed via optional keyword arguments.

required input arguments:

mcamatrix with the MCA data

ffilter values along the scan

optional keyword arguments:

ffcustome filter factors

return value: Array with the same shape as mca with the corrected MCA data.

recarray2hdf5(*h5g*, *rec*, *name*, *desc*, ***keyargs*)
Save a record array in an HDF5 file. A pytables table object is used to store the data.

required input arguments:

h5gHDF5 group object or path

recrecord array

namename of the table in the file

descdescription of the table in the file

optional keyword arguments:

return value:

taba HDF5 table object

set_abs_factors (*ff*)
Set the global absorber factors in the module.

spectra2hdf5 (*dir*, *fname*, *mcatemp*, ***keyargs*)
Convert SPECTRA scan data to a HDF5 format.

required input arguments:

dirdirectory where the scan is stored

fnamename of the SPECTRA data file

mcatemptemplate for the MCA file names

optional keyword arguments:

nameoptional name under which to save the data

descoptional description of the scan

xrutils.io.spectra.**get_spectra_files** (*dirname*)
Return a list of spectra files within a directory.

required input arguments:

dirnamename of the directory to search

return values:list with filenames

xrutils.io.spectra.**geth5_spectra_map** (*h5file*, *scans*, **args*, ***kwargs*)
function to obtain the omega and twotheta as well as intensity values for a reciprocal space map saved in an HDF5 file, which was created from a spectra file by the Save2HDF5 method.

further more it is possible to obtain even more positions from the data file if more than two string arguments with its names are given

Parameters

h5ffile object of a HDF5 file opened using pytables

scansnumber of the scans of the reciprocal space map (int,tuple or list)

***args: names of the motors (strings)**

omnamename of the omega motor (or its equivalent)

ttnamename of the two theta motor (or its equivalent)

****kwargs (optional):**

mcaname of the mca data (if available) otherwise None (default: "MCA")

samplenamestring with the hdf5-group containing the scan data if ommited the first child node of h5f.root will be used to determine the sample name

Returns

[ang1,ang2,...],MAP:angular positions of the center channel of the position sensitive detector (numpy.ndarray 1D) together with all the data values as stored in the data file (includes the intensities e.g. MAP['MCA']).

xrutils.io.spectra.**read_data** (*fname*)
Read a spectra data file (a file with now MCA data).

required input arguments:

fnamename of the file to read

return values: (data,hdr)

datanumpy record array where the keys are the column names

hdra dictionary with header information

xrutils.io.spectra.**read_mca**(*fname*)
Read a single SPECTRA MCA file.

required input arguments:

fnamename of the file to read

return value:

dataa numpy array witht the MCA data

xrutils.io.spectra.**read_mca_dir**(*dirname*, *filetemp*, ***keyargs*)
Read all MCA files within a directory

xrutils.io.spectra.**read_mcas**(*ftemp*, *cntstart*, *cntstop*)
Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

materials Package

_create_database Module script to create the HDF5 database from the raw data of XOP this file is only needed for administration

_create_database_alt Module script to create the HDF5 database from the raw data of XOP this file is only needed for administration

database Module module to handle access to the optical parameters database

class xrutils.materials.database.**DataBase**(*fname*)
Bases: object

Close()
Close an opend database file.

Create(*dbname*, *dbdesc*)
Creates a new database. If the database file already exists its content is delete.

required input arguments:

dbnamename of the database

dbdesca short description of the database

CreateMaterial(*name*, *description*)
This method creates a new material. If the material group already exists the procedure is aborted.

required input arguments:

namea string with the name of the material

descriptiona string with a description of the material

GetF0(*q*)
Obtain the f0 scattering factor component for a particular momentum transfer q.

required input argument:

qsingle float value or numpy array

GetF1(*en*)
Return the second, energy dependent, real part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

GetF2(*en*)
Return the imaginary part of the scattering factor for a certain energy en.

required input arguments:

enfloat or numpy array with the energy

Open(*mode='r'*)
Open an existing database file.

SetF0(*parameters*)
Save f0 fit parameters for the set material. The fit parameters are stored in the following order: c,a1,b1,.....,a4,b4

required input argument:

parameterslist or numpy array with the fit parameters

SetF1(*en*, *f1*)
Set f1 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f1list or numpy array with f1 values

SetF2(*en*, *f2*)
Set f2 tabels values for the active material.

required input arguments:

enlist or numpy array with energy in (eV)

f2list or numpy array with f2 values

SetMaterial(*name*)
Set a particular material in the database as the actual material. All operations like setting and getting optical constants are done for this particular material.

requiered input arguments:

namestring with the name of the material

SetWeight(*weight*)
Save weight of the element as float

required input argument:

weightatomic standard weight of the element (float)

xrutils.materials.database.**add_f0_from_intertab**(*db*, *itabfile*)
Read f0 data from international tables of crystallography and add it to the database.

xrutils.materials.database.**add_f0_from_xop**(*db*, *xopfile*)
Read f0 data from f0_xop.dat and add it to the database.

xrutils.materials.database.**add_f1f2_from_ascii_file**(*db*, *asciifile*, *element*)
Read f1 and f2 data for specific element from ASCII file (3 columns) and save it to the database.

xrutils.materials.database.**add_f1f2_from_henkedb**(*db*, *henkefile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_f1f2_from_kissel**(*db*, *kisselfile*)
Read f1 and f2 data from Henke database and add it to the database.

xrutils.materials.database.**add_mass_from_NIST**(*db*, *nistfile*)
Read atoms standard mass and save it to the database.

xrutils.materials.database.**init_material_db**(*db*)

elements Module

lattice Module

module handling crystal lattice structures

class xrutils.materials.lattice.**Atom**(*name*, *num*)
Bases: object

f(*q*, *en='config'*)
function to calculate the atomic structure factor F

Parameter

qmomentum transfer

enenergy for which F should be calculated, if omitted the value from the xrutils configuration is used

Returns

f (float)

f0(*q*)

f1(*en='config'*)

f2(*en='config'*)

xrutils.materials.lattice.**BCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**BCTLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**BaddeleyiteLattice**(*aa*, *ab*, *a*, *b*, *c*, *beta*, *deg=True*)

xrutils.materials.lattice.**CuMnAsLattice**(*aa*, *ab*, *ac*, *a*, *b*, *c*)

xrutils.materials.lattice.**CubicFm3mBaF2**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**CubicLattice**(*a*)
Returns a Lattice object representing a simple cubic lattice.

required input arguments:

alattice parameter

return value:an instance of Lattice class

xrutils.materials.lattice.**DiamondLattice**(*aa*, *a*)

xrutils.materials.lattice.**FCCLattice**(*aa*, *a*)

xrutils.materials.lattice.**GeneralPrimitiveLattice**(*a*, *b*, *c*, *alpha*, *beta*, *gamma*)

xrutils.materials.lattice.**HCPLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal3CLattice**(*aa*, *ab*, *a*, *c*)

xrutils.materials.lattice.**Hexagonal4HLattice**(*aa*, *ab*, *a*, *c*, *u=0.1875*, *v1=0.25*, *v2=0.4375*)

xrutils.materials.lattice.**Hexagonal6HLattice**(*aa*, *ab*, *a*, *c*)

class xrutils.materials.lattice.**Lattice**(*a1*, *a2*, *a3*, *base=None*)
Bases: object

class Lattice: This object represents a Bravais lattice. A lattice consists of a base

ApplyStrain(*eps*)
Applies a certain strain on a lattice. The result is a change in the base vectors.

requiered input arguments:

epsa 3x3 matrix independent strain components

GetPoint(**args*)
determine lattice points with indices given in the argument

Examples

```
>>> xu.materials.Si.lattice.GetPoint(0,0,4)
array([  0.     ,   0.     ,  21.72416])
```

or

```
>>> xu.materials.Si.lattice.GetPoint((1,1,1))
array([ 5.43104,  5.43104,  5.43104])
```

ReciprocalLattice()

UnitCellVolume()
function to calculate the unit cell volume of a lattice (angstrom^3)

class xrutils.materials.lattice.**LatticeBase**(**args*, ***keyargs*)
Bases: list

The LatticeBase class implements a container for a set of points that form the base of a crystal lattice. An instance of this class can be treated as a simple container object.

append(*atom*, *pos*, *occ=1.0*, *b=0.0*)
add new Atom to the lattice base

Parameter

atomatom object to be added

posposition of the atom

occoccupancy (default=1.0)

bb-factor of the atom used as exp(-b*q**2/(4*pi)**2) to reduce the intensity of this atom (only used in case of temp=0 in StructureFactor and chi calculation)

xrutils.materials.lattice.**NaumanniteLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**PerovskiteTypeRhombohedral**(*aa*, *ab*, *ac*, *a*, *ang*)

xrutils.materials.lattice.**QuartzLattice**(*aa*, *ab*, *a*, *b*, *c*)

xrutils.materials.lattice.**RockSaltLattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RockSalt_Cubic_Lattice**(*aa*, *ab*, *a*)

xrutils.materials.lattice.**RutileLattice**(*aa*, *ab*, *a*, *c*, *u*)

xrutils.materials.lattice.**TetragonalIndiumLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TetragonalTinLattice**(*aa*, *a*, *c*)

xrutils.materials.lattice.**TrigonalR3mh**(*aa*, *a*, *c*)

xrutils.materials.lattice.**WurtziteLattice**(*aa*, *ab*, *a*, *c*, *u=0.375*, *biso=0.0*)

xrutils.materials.lattice.**ZincBlendeLattice**(*aa*, *ab*, *a*)

material Module class module implements a certain material

class xrutils.materials.material.**Alloy**(*matA*, *matB*, *x*)
Bases: xrutils.materials.material.Material

RelaxationTriangle(*hkl*, *sub*, *exp*)
function which returns the relaxation trianlge for a Alloy of given composition. Reciprocal space coordinates are calculated using the user-supplied experimental class

Parameter

hklMiller Indices

subsubstrate material or lattice constant (Instance of Material class or float)

expExperiment class from which the Transformation object and ndir are needed

Returns

qy,qzreciprocal space coordinates of the corners of the relaxation triangle

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**Cij2Cijkl**(*cij*)
Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.materials.material.**Cijkl2Cij**(*cijkl*)
Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.materials.material.**CubicAlloy**(*matA*, *matB*, *x*)
Bases: xrutils.materials.material.Alloy

ContentBasym(*q_inp*, *q_perp*, *hkl*, *sur*)
function that determines the content of B in the alloy from the reciprocal space position of an asymmetric peak and also sets the content in the current material

Parameter

q_inpinplane peak position of reflection hkl of the alloy in reciprocal space

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured asymmetric reflection

surMiller indices of the surface (determines the perpendicular direction)

Returns

content,[a_inplane,a_perp,a_bulk_perp(x), eps_inplane, eps_perp] :the content of B in the alloy determined from the input variables and the lattice constants calculated from the reciprocal space positions as well as the strain (eps) of the layer

ContentBsym(*q_perp*, *hkl*, *inpr*, *asub*, *relax*)
function that determines the content of B in the alloy from the reciprocal space position of a symetric peak. As an additional input the substrates lattice parameter and the degree of relaxation must be given

Parameter

q_perpperpendicular peak position of the reflection hkl of the alloy in reciprocal space

hklMiller indices of the measured symmetric reflection (also defines the surface normal

inprMiller indices of a Bragg peak defining the inplane reference direction

asubsubstrate lattice constant

relaxdegree of relaxation (needed to obtain the content from symmetric reciprocal space position)

Returns

contentthe content of B in the alloy determined from the input variables

xrutils.materials.material.**CubicElasticTensor**(*c11*, *c12*, *c44*)
Assemble the 6x6 matrix of elastic constants for a cubic material from the three independent components of a cubic crystal

Parameter

c11,c12,c44independent components of the elastic tensor of cubic materials

Returns

6x6 materix with elastic constants

xrutils.materials.material.**GeneralUC**(*a=4*, *b=4*, *c=4*, *alpha=90*, *beta=90*, *gamma=90*, *name='General'*)
general material with primitive unit cell but possibility for different a,b,c and alpha,beta,gamma

Parameters

a,b,cunit cell extenstions (Angstrom)

alphaangle between unit cell vectors b,c

betaangle between unit cell vectors a,c

gammaangle between unit cell vectors a,b

returns a Material object with the specified properties

xrutils.materials.material.**HexagonalElasticTensor**(*c11*, *c12*, *c13*, *c33*, *c44*)
Assemble the 6x6 matrix of elastic constants for a hexagonal material from the five independent components of a hexagonal crystal

Parameter

c11,c12,c13,c33,c44independent components of the elastic tensor of a hexagonal material

Returns

6x6 materix with elastic constants

class xrutils.materials.material.**Material**(*name*, *lat*, *cij=None*, *thetaDebye=None*)
Bases: object

ApplyStrain(*strain*, ***keyargs*)

B

GetMismatch(*mat*)
Calculate the mismatch strain between the material and a second material

Q(**hkl*)

Return the Q-space position for a certain material.

required input arguments:

hkllist or numpy array with the Miller indices (or Q(h,k,l) is also possible)

StructureFactor(*q*, *en='config'*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and energy at a certain temperature of the material

Parameter

qvectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enenergy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

the complex structure factor

StructureFactorForEnergy(*q0*, *en*, *temp=0*)

caluclates the structure factor of a material for a certain momentum transfer and a bunch of energies

Parameter

q0vectorial momentum transfer (vectors as list,tuple or numpy array are valid)

enlist, tuple or array of energy values in eV

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

StructureFactorForQ(*q*, *en0='config'*, *temp=0*)

caluclates the structure factor of a material for a bunch of momentum transfers and a certain energy

Parameter

qvectorial momentum transfers; list of vectores (list, tuple or array) of length 3 e.g.: (Si.Q(0,0,4),Si.Q(0,0,4.1),...) or numpy.array([Si.Q(0,0,4),Si.Q(0,0,4.1)])

en0energy value in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

Returns

complex valued structure factor array

a1

a2

a3

b1

b2

b3

beta(*en='config'*)

function to calculate the imaginary part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

beta (float)

chi0(*en='config'*)
calculates the complex chi_0 values often needed in simulations. They are closely related to delta and beta (n = 1 + chi_r0/2 + i*chi_i0/2 vs. n = 1 - delta + i*beta)

chih(*q*, *en='config'*, *temp=0*, *polarization='S'*)
calculates the complex polarizability of a material for a certain momentum transfer and energy

Parameter

qmomentum transfer in (1/A)

enxray energy in eV, if omitted the value from the xrutils configuration is used

temptemperature used for Debye-Waller-factor calculation

polarizationeither 'S' (default) sigma or 'P' pi polarization

Returns

(abs(chih_real),abs(chih_imag)) complex polarizability

critical_angle(*en='config'*, *deg=True*)
calculate critical angle for total external reflection

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

degreturn angle in degree if True otherwise radians (default:True)

Returns

Angle of total external reflection

dTheta(*Q*, *en='config'*)
function to calculate the refractive peak shift

Parameter

Qmomentum transfer (1/A)

enx-ray energy (eV), if omitted the value from the xrutils configuration is used

Returns

deltaThetapeak shift in degree

delta(*en='config'*)
function to calculate the real part of the deviation of the refractive index from 1 (n=1-delta+i*beta)

Parameter

enx-ray energy eV, if omitted the value from the xrutils configuration is used

Returns

delta (float)

idx_refraction(*en='config'*)
function to calculate the complex index of refraction of a material in the x-ray range

Parameter

enenergy of the x-rays, if omitted the value from the xrutils configuration is used

Returns

n (complex)

lam

mu

nu

xrutils.materials.material.**PseudomorphicMaterial**(*submat*, *layermat*)
This function returns a material whos lattice is pseudomorphic on a particular substrate material. This function works meanwhile only for cubic materials.

required input arguments:

submatsubstrate material

layermatbulk material of the layer

return value: An instance of Material holding the new pseudomorphically strained material.

class xrutils.materials.material.**SiGe**(*x*)
Bases: xrutils.materials.material.CubicAlloy

lattice_const_AB(*latA*, *latB*, *x*)

x

xrutils.materials.material.**index_map_ij2ijkl**(*ij*)

xrutils.materials.material.**index_map_ijkl2ij**(*i*, *j*)

math Package

fit Module module with a function wrapper to scipy.optimize.leastsq for fitting of a 2D function to a peak or a 1D Gauss fit with the odr package

xrutils.math.fit.**fit_peak2d**(*x*, *y*, *data*, *start*, *drange*, *fit_function*, *maxfev=2000*)
fit a two dimensional function to a two dimensional data set e.g. a reciprocal space map

Parameters

x,ydata coordinates (do NOT need to be regularly spaced)

datadata set used for fitting (e.g. intensity at the data coords)

startset of starting parameters for the fit used as first parameter of function fit_function

drangelimits for the data ranges used in the fitting algorithm e.g. it is clever to use only a small region around the peak which should be fitted: [xmin,xmax,ymin,ymax]

fit_functionfunction which should be fitted must accept the parameters (x,y,*params)

Returns

(fitparam,cov)the set of fitted parameters and covariance matrix

xrutils.math.fit.**gauss_fit**(*xdata*, *ydata*, *iparams=[ ]*, *maxit=200*)
Gauss fit function using odr-pack wrapper in scipy similar to :https://github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting

Parameters

xdataxcoordinates of the data to be fitted

ydataycoordinates of the data which should be fit

keyword parameters:

iparamsinitial paramters for the fit (determined automatically if nothing is given

maxitmaximal iteration number of the fit

Returns

params,sd_params,itlim

the Gauss parameters as defined in function Gauss1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

functions Module module with several common function needed in xray data analysis

xrutils.math.functions.**Debye1**(*x*)
function to calculate the first Debye function as needed for the calculation of the thermal Debye-Waller-factor by numerical integration

for definition see: :http: //en.wikipedia.org/wiki/Debye_function

D1(x) = (1/x) int_0^x t/(exp(t)-1) dt

Parameters

xargument of the Debye function (float)

Returns

D1(x)float value of the Debye function

xrutils.math.functions.**Gauss1d**(*x*, **p*)
function to calculate a general one dimensional Gaussian

Parameters

plist of parameters of the Gaussian [XCEN,SIGMA,AMP,BACKGROUND] for information: SIGMA = FWHM / (2*sqrt(2*log(2)))

xcoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position x

xrutils.math.functions.**Gauss1d_der_p**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect the parameters p

for parameter description see Gauss1d

xrutils.math.functions.**Gauss1d_der_x**(*x*, **p*)
function to calculate the derivative of a Gaussian with respect to x

for parameter description see Gauss1d

xrutils.math.functions.**Gauss2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Gaussian

Parameters

plist of parameters of the Gauss-function [XCEN,YCEN,SIGMAX,SIGMAY,AMP,BACKGROUND,ANGLE] SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of the Gaussian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Gaussian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz1d**(*x*, **p*)
function to calculate a general one dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**Lorentz2d**(*x*, *y*, **p*)
function to calculate a general two dimensional Lorentzian

Parameters

plist of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE]
ANGLE = rotation of the X,Y direction of the Lorentzian

x,ycoordinate(s) where the function should be evaluated

Returns

the value of the Lorentian described by the parameters p at position (x,y)

xrutils.math.functions.**TwoGauss2d**(*x*, *y*, **p*)
function to calculate two general two dimensional Gaussians

Parameters

plist of parameters of the Gauss-function [XCEN1,YCEN1,SIGMAX1,SIGMAY1,AMP1,ANGLE1,XCEN2,YCEN
SIGMA = FWHM / (2*sqrt(2*log(2))) ANGLE = rotation of the X,Y direction of
the Gaussian

x,ycoordinate(s) where the function should be evaluated

Return

the value of the Gaussian described by the parameters p at position (x,y)

transforms Module

class xrutils.math.transforms.**AxisToZ**(*newzaxis*)
Bases: xrutils.math.transforms.CoordinateTransform

Creates a coordinate transformation to move a certain axis to the z-axis. The rotation is done along the great circle. The x-axis of the new coordinate frame is created to be normal to the new and original z-axis. The new y-axis is create in order to obtain a right handed coordinate system.

xrutils.math.transforms.**Cij2Cijkl**(*cij*)
Converts the elastic constants matrix (tensor of rank 2) to the full rank 4 cijkl tensor.

required input arguments:

cij(6,6) cij matrix as a numpy array

return value:

cijkl(3,3,3,3) cijkl tensor as numpy array

xrutils.math.transforms.**Cijkl2Cij**(*cijkl*)
Converts the full rank 4 tensor of the elastic constants to the (6,6) matrix of elastic constants.

required input arguments:

cijkl(3,3,3,3) cijkl tensor as numpy array

return value:

cij(6,6) cij matrix as a numpy array

class xrutils.math.transforms.**CoordinateTransform**(*v1*, *v2*, *v3*)
Bases: xrutils.math.transforms.Transform

Create a Transformation object which transforms a point into a new coordinate frame. The new frame is determined by the three vectors v1/norm(v1), v2/norm(v2) and v3/norm(v3), which need to be orthogonal!

class xrutils.math.transforms.**Transform**(*matrix*)
Bases: object

xrutils.math.transforms.**XRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the x-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**YRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the y-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**ZRotation**(*alpha*, *deg=True*)
Returns a transform that represents a rotation about the z-axis by an angle alpha. If deg=True the angle is assumed to be in degree, otherwise the function expects radiants.

xrutils.math.transforms.**index_map_ij2ijkl**(*ij*)

xrutils.math.transforms.**index_map_ijkl2ij**(*i*, *j*)

xrutils.math.transforms.**mycross**(*vec*, *mat*)
function implements the cross-product of a vector with each column of a matrix

xrutils.math.transforms.**rotarb**(*vec*, *axis*, *ang*, *deg=True*)
function implements the rotation around an arbitrary axis by an angle ang positive rotation is anti-clockwise when looking from positive end of axis vector

Parameter

vecnumpy.array or list of length 3

axisnumpy.array or list of length 3

angrotation angle in degree (deg=True) or in rad (deg=False)

degboolean which determines the input format of ang (default: True)

Returns

rotvecrotated vector as numpy.array

Example

```
>>> rotarb([1,0,0],[0,0,1],90)
array([  6.12323400e-17,   1.00000000e+00,   0.00000000e+00])
```

xrutils.math.transforms.**tensorprod**(*vec1*, *vec2*)
function implements an elementwise multiplication of two vectors

vector Module module with vector operations, mostly numpy functionality is used for the vector operation itself, however custom error checking is done to ensure vectors of length 3.

xrutils.math.vector.**VecAngle**(*v1*, *v2*, *deg=False*)
calculate the angle between two vectors. The following formula is used v1.v2 = norm(v1)*norm(v2)*cos(alpha)

alpha = acos((v1.v2)/(norm(v1)*norm(v2)))

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

optional keyword arguments:

deg(default: false) return result in degree otherwise in radiants

return value:float value with the angle inclined by the two vectors

xrutils.math.vector.**VecDot**(*v1*, *v2*)
Calculate the vector dot product.

required input arguments:

v1vector as numpy array or list

v2vector as numpy array or list

return value:float value

xrutils.math.vector.**VecNorm**(*v*)
Calculate the norm of a vector.

required input arguments:

vvector as list or numpy array

return value:float holding the vector norm

xrutils.math.vector.**VecUnit**(*v*)
Calculate the unit vector of v.

required input arguments:

vvector as list or numpy array

return value:numpy array with the unit vector

xrutils.math.vector.**getSyntax**(*vec*)
returns vector direction in the syntax 'x+' 'z-' or equivalents therefore works only for principle vectors of the coordinate system like e.g. [1,0,0] or [0,2,0]

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

xrutils.math.vector.**getVector**(*string*)
returns unit vector along a rotation axis given in the syntax 'x+' 'z-' or equivalents

Parameters

string[xyz][+-]

Returns

vector along the given direction as numpy array

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

X

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z