xrayutilities Documentation

Release 1.0.6

Dominik Kriegner   **Eugen Wintersberger**

April 15, 2014

If you look for downloading the package go to Sourceforge (source distribution) or the Python package index (MS Windows binary). Installation instructions you find further down Installation.

Read more about *xrayutilities* below or in Journal of Applied Crystallography 2013, Volume 46, 1162-1170

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
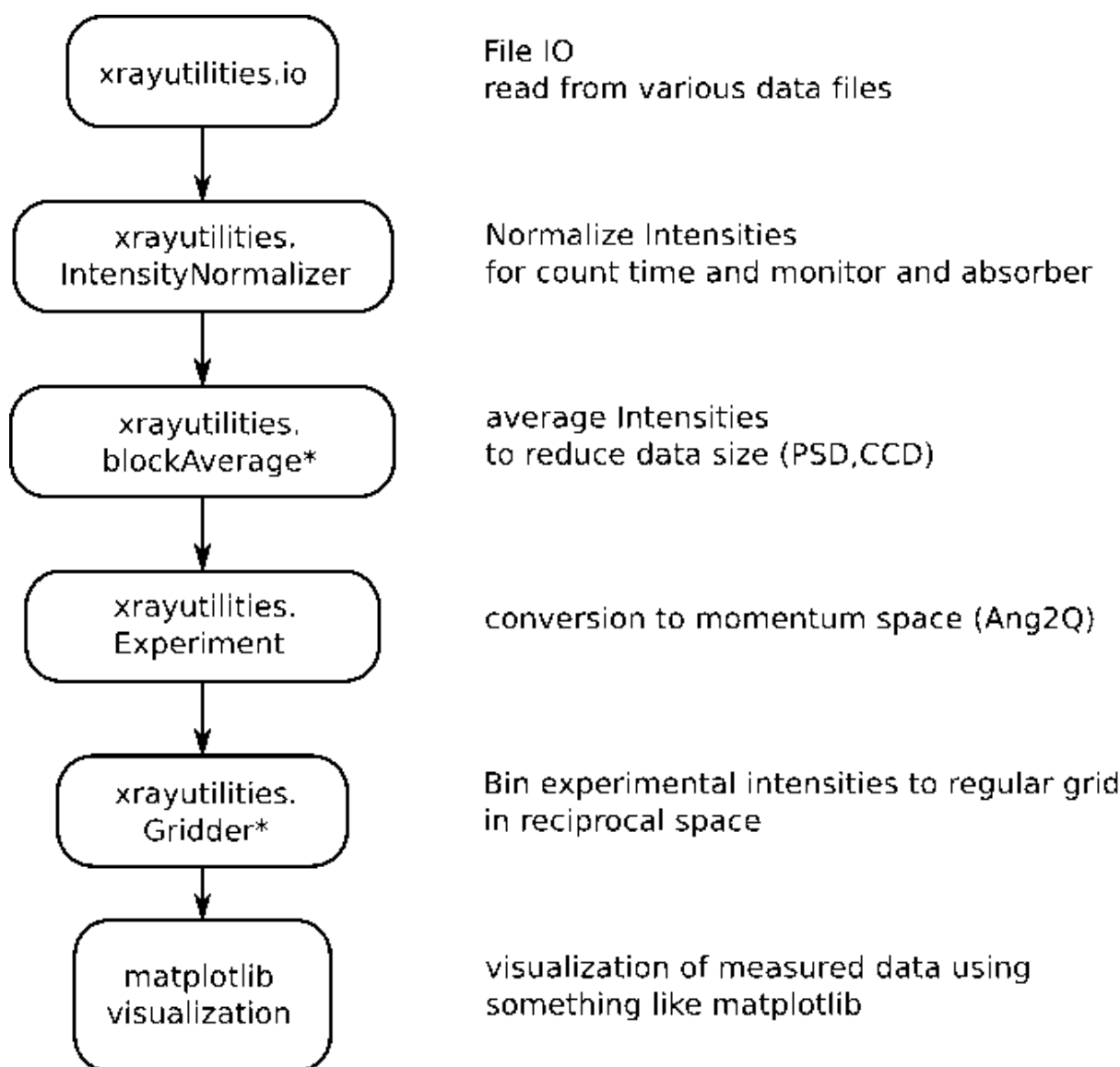
xrayutilities provides a set of functions to read experimental data from various data file formats. All of them are gathered in the `io`-subpackage. After reading data with a function from the io-submodule the data might be corrected for monitor counts and/or absorption factor of a beam attenuator. A special set of functions is provided to perform this for point, linear and area detectors.

Since the amount of data taken with modern detectors often is too large to be able to work with them properly, a functions for reducing the data from linear and area detectors are provided. They use block-averaging to reduce the amount of data. Use those carefully not to loose the features you are interested in 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 crystalline samples and to convert data between angular and reciprocal 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 this article. In standard cases, Users will only need the initialized routines, which predefine a certain goniometer geometry like the popular four-cirlce and six-circle geometries.

After the conversion to reciprocal space, it is convenient to transform the data to a regular grid for visualization. For this purpose the `gridder`-module has been included into *xrayutilities*. 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
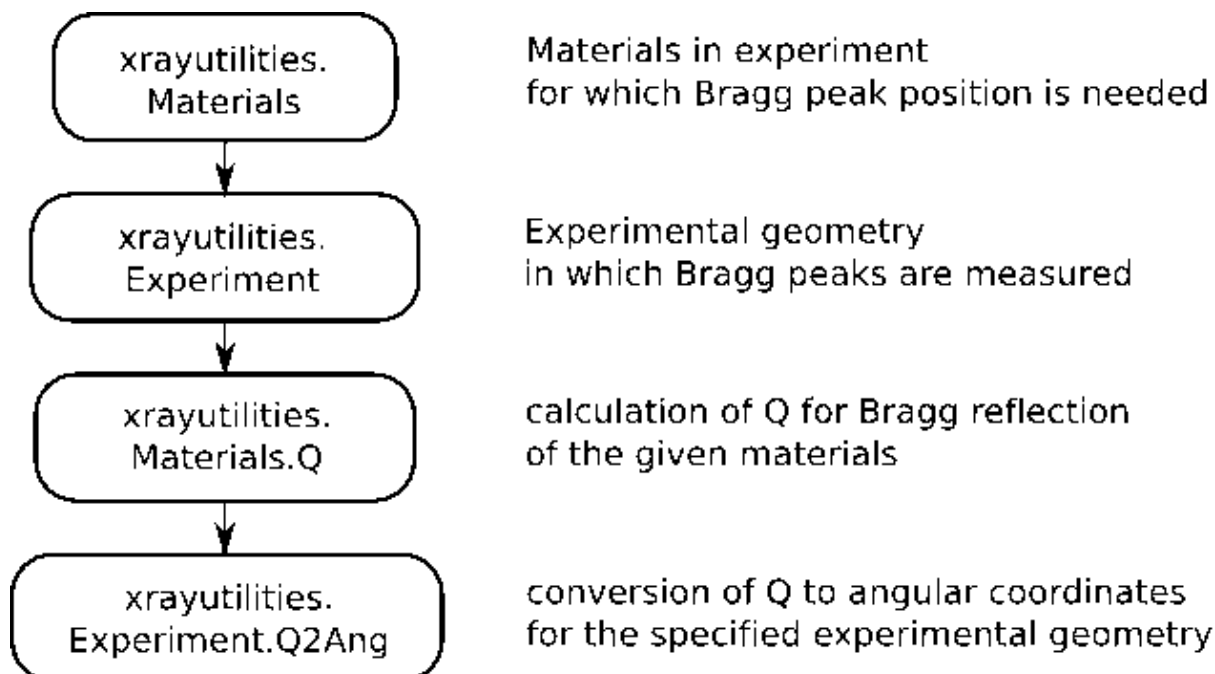
Calculation of angles needed to align Bragg reflections in various diffraction geometries is done using the Materials defined in the `materials`-package. This package provides a set of classes to describe crystal lattices and materials. Once such a material is properly defined one can calculate its properties, 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.

```
1   """
2   Example script to show how to use xrayutilities to read and plot
3   reciprocal space map scans from a spec file created at the ESRF/ID10B
4
5   for details about the measurement see:
6       D Kriegner et al. Nanotechnology 22 425704 (2011)
7       http://dx.doi.org/10.1088/0957-4484/22/42/425704
8   """
9
10  import numpy
11  import matplotlib.pyplot as plt
12  import xrayutilities as xu
13  import os
```

```python
14
15   # global setting for the experiment
16   sample = "test" # sample name used also as file name for the data file
17   energy = 8042.5 # x-ray energy in eV
18   center_ch = 715.9 # center channel of the linear detector
19   chpdeg = 345.28 # channels per degree of the linear detector
20   roi=[100,1340] # region of interest of the detector
21   nchannel = 1500 # number of channels of the detector
22
23   # intensity normalizer function responsible for count time and absorber correction
24   absfun = lambda d: d["detcorr"]/d["psd2"].astype(numpy.float)
25   normalizer_detcorr = xu.IntensityNormalizer("MCA",mon="Monitor",time="Seconds",absfun=absfun)
26
27   # substrate material used for Bragg peak calculation to correct for experimental offsets
28   InP = xu.materials.InP
29
30   # initialize experimental class to specify the reference directions of your crystal
31   # 11-2: inplane reference
32   # 111: surface normal
33   hxrd = xu.HXRD(InP.Q(1,1,-2),InP.Q(1,1,1),en=energy)
34
35   # configure linear detector
36   # detector direction + parameters need to be given
37   # mounted along z direction, which corresponds to twotheta
38   hxrd.Ang2Q.init_linear('z-',center_ch,nchannel,chpdeg=chpdeg,roi=roi)
39
40   # read spec file and save to HDF5-file
41   # since reading is much faster from HDF5 once the data are transformed
42   h5file = os.path.join("data",sample+".h5")
43   try: s  # try if spec file object already exist ("run -i" in ipython)
44   except NameError: s = xu.io.SPECFile(sample+".spec",path="data")
45   else: s.Update()
46   s.Save2HDF5(h5file)
47
48   #################################
49   # InP (333) reciprocal space map
50   omalign = 43.0529 # experimental aligned values
51   ttalign = 86.0733
52   [omnominal,dummy,dummy,ttnominal] = hxrd.Q2Ang(InP.Q(3,3,3)) # nominal values of the substrate pe
53
54   # read the data from the HDF5 file
55   #(scan number:36, names of motors in spec file: omega= sample rocking, gamma = twotheta)
56   [om,tt],MAP = xu.io.geth5_scan(h5file,36,'omega','gamma')
57   # normalize the intensity values (absorber and count time corrections)
58   psdraw = normalizer_detcorr(MAP)
59   # remove unusable detector channels/regions (no averaging of detector channels)
60   psd = xu.blockAveragePSD(psdraw, 1, roi=roi)
61
62   # convert angular coordinates to reciprocal space + correct for offsets
63   [qx,qy,qz] = hxrd.Ang2Q.linear(om,tt,delta=[omalign-omnominal, ttalign-ttnominal])
64
65   # calculate data on a regular grid of 200x201 points
66   gridder = xu.Gridder2D(200,201)
67   gridder(qy,qz,psd)
68   # maplog function limits the shown dynamic range to 8 orders of magnitude from the maxium
69   INT = xu.maplog(gridder.data.T,8.,0)
70
71   # plot the intensity as contour plot using matplotlib
72   plt.figure()
73   cf = plt.contourf(gridder.xaxis, gridder.yaxis,INT,100,extend='min')
74   plt.xlabel(r'$Q_{[11\bar2]}$ ($\AA^{-1}$)')
75   plt.ylabel(r'$Q_{[\bar1\bar1\bar1]}$ ($\AA^{-1}$)')
76   cb = plt.colorbar(cf)
```

```
77    cb.set_label(r"$\log($Int$)$ (cps)")
```

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

XRAYUTILITIES PYTHON PACKAGE

xrayutilities is a Python package for assisting with x-ray diffraction experiments. Its the python package included in *xrayutilities*.

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

Authors Dominik Kriegner <dominik.kriegner@gmail.com> and Eugen Wintersberger <eugen.wintersberger@desy.de>

for more details see the full API documentation of `xrayutilities` found here: *Examples and API-documentation*.

INSTALLATION

3.1 Express instructions

- install the dependencies (Windows: pythonxy; 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:

```
> python setup.py install
```

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

Note: The python package of *xrayutilities* was formerly called ''xrutils''

3.2 Detailed instructions

Installing *xrayutilities* is done using Python's distutils

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.

- **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 distutils packaging system to build and install all of its components. You can perform the installation by executing

```
>python setup.py install
```

or

```
>python setup.py install --prefix=INSTALLPATH
```

in the root directory of the source distribution.

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 all of the needed dependencies for installing *xrayutilities*.

The setup of the environment variables is also done by the python(x,y) installation. One can proceed with the installation of *xrayutilities* directly! The easiest way to do this on windows is to use the binaries distributed on the Python package index, otherwise one can follow the general installation instructions. Depending on your compiler on Microsoft Windows it might be necessary to perform the building of the Python extension separately and specify the compiler manually. This is done by

```
python setup.py build -c <compiler_name>
```

Using Python(x,y) you want to specify "mingw32" as compiler name.

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)

- 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.

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.:

```python
import tables
import xrayutilities 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)
s.scan10.ReadData()
scan10data = s.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)

```
1  s.Update() # reparse for new scans in open SPECFile instance
2
3  # reread data method (1)
4  s.scan10.ReadData()
5  scan10data = s.scan10.data
6
7  # reread data method (2)
8  s.Save2HDF5(h5) # save content of SPEC file to HDF5 file
9  # read data from HDF5 file
10 [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.:

```
1  import tables
2  import xrayutilities as xu
3  import numpy
4
5  specfile = "specfile.spec"
6  h5file = "h5file.h5"
7  h5 = tables.openFile(h5file,mode='a')
8
9  s = xu.io.SPECFile(specfile,path=specdir)
10 s.Save2HDF5(h5) # save to hdf5 file
11
12 # read ccd frames from EDF files
13 for i in range(1,1000,1):
14     efile = "edfdir/sample_%04d.edf" %i
15     e = xu.io.edf.EDFFile(efile,path=specdir)
16     e.ReadData()
17     g5 = h5.createGroup(h5.root,"frelon_%04d" %i)
18     e.Save2HDF5(h5,group=g5)
19
20 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.

```python
import xrayutilities as xu
Si = xu.materials.Si  # load material from materials submodule

# initialize experimental class with directions from experiment
hxrd = xu.HXRD(Si.Q(1,1,-2),Si.Q(1,1,1))
# calculate angles of Bragg reflections and print them to the screen
om,chi,phi,tt = hxrd.Q2Ang(Si.Q(1,1,1))
print("Si (111)")
print("om,tt: %8.3f %8.3f" %(om,tt))
om,chi,phi,tt = hxrd.Q2Ang(Si.Q(2,2,4))
print("Si (224)")
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:

```python
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:

```python
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.

```python
import xrayutilities as xu
import matplotlib.pyplot as plt

energy = (2*8048 + 8028)/3. # copper k alpha 1,2

# creating Indium powder
In_powder = xu.Powder(xu.materials.In,en=energy)
# calculating the reflection strength for the powder
In_powder.PowderIntensity()

# 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
```

```
21  plt.bar(In_powder.ang*2, In_powder.data/In_powder.data.max(), width=0.3, bottom=0,
22          linewidth=0, color='r',align='center', orientation='vertical',label="Indium bar plot")
23
24  plt.legend(); plt.set_xlim(15,100); plt.grid()
```

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

```
1   >>> print In_powder
2      Powder diffraction object
3      -------------------------
4      Material: In
5      Lattice:
6      a1 = (3.252300 0.000000 0.000000), 3.252300
7      a2 = (0.000000 3.252300 0.000000), 3.252300
8      a3 = (0.000000 0.000000 4.946100), 4.946100
9      alpha = 90.000000, beta = 90.000000, gamma = 90.000000
10     Lattice base:
11     Base point 0: In (49) (0.000000 0.000000 0.000000) occ=1.00 b=0.00
12     Base point 1: In (49) (0.500000 0.500000 0.500000) occ=1.00 b=0.00
13     Reflections:
14     --------------
15         h k l      |    tth    |   |Q|   |    Int    |   Int (%)
16        ----------------------------------------------------------------
17        [-1,  0, -1]    32.9611     2.312      217.75       100.00
18        [ 0,  0, -2]    36.3267     2.541       41.80        19.20
19        [-1, -1,  0]    39.1721     2.732       67.72        31.10
20        [-1, -1, -2]    54.4859     3.731       50.75        23.31
21        ....
```

4.1.3 Using the `Gridder` classes

xrayutilities provides Gridder classes for 1D, 2D, and 3D data sets. These Gridders map irregular spaced data onto a regular grid. This is often needed after transforming data measured at equally spaced angular positions to reciprocal space were their spacing is irregular.

In 1D this process actually equals the calculation of a histogramm. Below you find the most basic way of using the Gridder in 2D. Other dimensions work very similar.

The most easiest use (what most user might need) is:

:: import xrayutilities as xu # import python package g = xu.Gridder2D(100,101) # initialize the Gridder object, which will

> # perform Gridding to a regular grid with 100x101 points

> #====== load some data here ===== g(x,y,data) # call the gridder with the data griddata = g.data # the data object of the Gridder contains the gridded data.

_.. note: previously you could use the Gridder's gdata object, which was always an internal buffer and should not be used anymore!

A more complicated example showing also sequential gridding is shown below. You need sequential gridding when you can not load all data at the same time, which is often problematic with 3D data sets. In such cases you need to specify the data range before the first call to the gridder.

:: import xrayutilities as xu # import python package g = xu.Gridder2D(100,101) # initialize the Gridder object g.KeepData(True) g.dataRange((1,2),(3,4)) # (xgrd_min,xgrd_max),(ygrd_min,ygrd_max) #====== load some data here ===== g(x,y,data) # call the gridder with the data griddata = g.data # the data object of the Gridder contains the so far gridded data.

> #====== load some more data here ===== g(x,y,data) # call the gridder with the new data griddata = g.data # the data object of the Gridder contains the combined gridded data.

4.1.4 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:

```python
import xrayutilities 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:

```python
import xrayutilities 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:

4.1. Examples   **19**

```python
1  import matplotlib.pyplot as plt
2
3  energy= numpy.linspace(500,20000,5000) # 500 - 20000 eV
4  F = InAs.StructureFactorForEnergy(InAs.Q(1,1,1),energy)
5
6  plt.figure(); plt.clf()
7  plt.plot(energy,F.real,'k-',label='Re(F)')
8  plt.plot(energy,F.imag,'r-',label='Imag(F)')
9  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.:

```python
1  # f = f0(|Q|) + f1(en) + j * f2(en)
2  import xrayutilities as xu
3  import numpy
4
5  Fe = xu.materials.elements.Fe # iron atom
6  Q = numpy.array([0,0,1.9],dtype=numpy.double)
7  en = 10000 # energy in eV
8
9  print "Iron (Fe): E: %9.1f eV" % en
10 print "f0: %8.4g" % Fe.f0(numpy.linalg.norm(Q))
11 print "f1: %8.4g" % Fe.f1(en)
12 print "f2: %8.4g" % Fe.f2(en)
```

4.1.5 User-specific config file

Several options of *xrayutilities* can be changed by options in a config file. This includes the default x-ray energy as well as parameters to set the number of threads used by the parallel code and the verbosity of the output.

The default options are stored inside the installad Python module and should not be changed. Instead it is suggested to use a user-specific config file '~/.xrayutilities.conf' or a 'xrayutilities.conf' file in the working directory.

An example of such a user config file is shown below:

```
# begin of xrayutilities configuration
[xrayutilities]

# verbosity level of information and debugging outputs
#   0: no output
#   1: very import notes for users
#   2: less import notes for users (e.g. intermediate results)
#   3: debuging output (e.g. print everything, which could be interesing)
#   levels can be changed in the config file as well
verbosity = 1

# default wavelength in Angstrom,
wavelength = MoKa1 # Molybdenum K alpha1 radiation (17479.374eV)

# default energy in eV
# if energy is given wavelength settings will be ignored
#energy = 10000 #eV

# number of threads to use in parallel sections of the code
nthreads = 1
#   0: the maximum number of available threads will be used (as returned by omp_get_max_threads()
#   n: n-threads will be used
```

4.1.6 Determining detector parameters

In the following three examples of how to determine the detector parameters for linear and area detectors is given. The procedure we use is in more detail described in this article.

Linear detectors

To determine the detector parameters of a linear detector one needs to perform a scan with the detector angle through the primary beam and aquire a detector spectrum at any point.

Using the following script determines the parameters necessary for the detector initialization, which are:

- pixelwidth of one channel

- the center channel

- and the detector tilt (optional)

```python
1    """
2    example script to show how the detector parameters
3    such as pixel width, center channel and detector tilt
4    can be determined for a linear detector.
5    """
6
7    import xrayutilities as xu
8    import os
9
10   # load any data file with with the detector spectra of a reference scan
11   # in the primary beam, here I use spectra measured with a Seifert XRD
12   # diffractometer
13   dfile = os.path.join("data","primarybeam_alignment20130403_2_dis350.nja")
14   s = xu.io.SeifertScan(dfile)
15
16   ang = s.axispos["T"] # detector angles during the scan
17   spectra = s.data[:,:,1] # detector spectra aquired
18
19   # determine detector parameters
20   # this function accepts some optional arguments to describe the goniometer
21   # see the API documentation
22   pwidth,cch,tilt = xu.analysis.linear_detector_calib(ang,spectra,usetilt=True)
```

Area detector (Variant 1)

To determine the detector parameters of a area detector one needs to perform scans with the detector angles through the primary beam and aquire a detector images at any position. For the area detector at least two scans (one with the outer detector and and one with the inner detector angle) are required.

Using the following script determines the parameters necessary for the detector initialization from such scans in the primary beam only. Further down we discuss an other variant which is also able to use additionally detector images recorded at the Bragg reflection of a known reference crystal.

The determined detector parameters are:

- pixelwidth of the channels in both directions (2 parameters)

- center channels: position of the primary beam at the true zero position of the goniometer (considering the outer angle offset) (2 parameters)

- detector tilt azimuth in degree from 0 to 360

- detector tilt angle in degree (>0deg)

- detector rotation around the primary beam in degree

- outer angle offset, which describes a offset of the outer detector angle from its true zero position

The misalignment parameters can be fixed during the fitting.

```python
"""
example script to show the detector parameter determination for area detectors from images recorde
"""

import xrayutilities as xu
import os

en=10300.0 #eV
datadir = os.path.join("data","wire_") # data path for CCD files
filetmp = os.path.join(datadir,"wire_12_%05d.edf.gz") # template for the CCD file names

## manually selected images
# select images which have the primary beam fully on the CCD
imagenrs = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,

images = []
ang1 = []
ang2 = []

# read images and angular positions from the data file
# this might differ for data taken at different beamlines since
# they way how motor positions are stored is not always consistent
for imgnr in imagenrs:
    filename = filetmp%imgnr
    edf = xu.io.EDFFile(filename)
    images.append(edf.data)
    ang1.append(float(edf.header['ESRF_ID01_PSIC_NANO_NU']))
    ang2.append(float(edf.header['ESRF_ID01_PSIC_NANO_DEL']))


# call the fit for the detector parameters
# detector arm rotations and primary beam direction need to be given
# in total 8 parameters are fitted, however the 4 misalignment parameters can be fixed
# they are the detector tilt azimuth, the detector tilt angle, the detector rotation around the p
param,eps = xu.analysis.sample_align.area_detector_calib(ang1,ang2,images,['z+','y-'],'x+',start=
```

A possible output of this script could be

```
fitted parameters: epsilon: 8.0712e-08 (2,['Parameter convergence'])
param: (cch1,cch2,pwidth1,pwidth2,tiltazimuth,tilt,detrot,outerangle_offset)
param: 140.07 998.34 4.4545e-05 4.4996e-05 72.0 1.97 -0.792 -1.543
please check the resulting data (consider setting plot=True)
detector rotation axis / primary beam direction (given by user): ['z+', 'y-'] / x+
detector pixel directions / distance: z- y+ / 1
detector initialization with: init_area('z-','y+',cch1=140.07,cch2=998.34,Nch1=516,Nch2=516, pwidt
AND ALWAYS USE an (additional) OFFSET of -1.9741deg in the OUTER DETECTOR ANGLE!
```

The output gives the fitted detector parameters and compiles the python code line one needs to use to initialize the detector. Important to note is that the outer angle offset which was determined by the fit (-1.9741 degree in the aboves example) is not included in the initialization of the detector parameters BUT needs to be used in every call to the q-conversion function as offset. This step needs to be performed manually by the user!

Area detector (Variant 2)

In addition to scans in the primary beam this variant enables also the use of detector images recorded in scans at Bragg reflections of a known reference materials. However this also required that the sample orientation and x-ray wavelength need to be fit. To keep the additional parameters as small as possible we only implemented this for symmetric coplanar diffractions.

The advantage of this method is that it is more sensitive to the outer angle offset also at large detector distances. The additional parameters are:

- sample tilt angle in degree

- sample tilt azimuth in degree

- and the x-ray wavelength in Angstrom

```python
1   """
2   example script to show the detector parameter determination for area detectors from images recorde
3   and at known symmetric coplanar Bragg reflections of a reference crystal
4   """
5
6   import xrayutilities as xu
7   import os
8   import numpy
9
10  Si = xu.materials.Si
11
12  datadir = 'data'
13  specfile = "si_align.spec"
14
15  en=15000 #eV
16  wl = xu.lam2en(en)
17  imgdir = os.path.join(datadir,"si_align_") # data path for CCD files
18  filetmp = "si_align_12_%04d.edf.gz"
19
20  qconv = xu.QConversion(['z+','y-'],['z+','y-'],[1,0,0])
21  hxrd = xu.HXRD(Si.Q(1,1,-2),Si.Q(1,1,1),wl=wl,qconv=qconv)
22
23  ## manually selected images
24
25  s = xu.io.SPECFile(specfile,path=datadir)
26  for num in [61,62,63,20,21,26,27,28]:
27      s[num].ReadData()
28      try:
29          imagenrs = numpy.append(imagenrs,s[num].data['ccd_n'])
30      except:
31          imagenrs = s[num].data['ccd_n']
32
33  # avoid images which do not have to full beam on the detector as well as other which show signal o
34  avoid_images = [37,57,62,63,65,87,99,106,110,111,126,130,175,181,183,185,204,206,207,208,211,212,2
35
36  images = []
37  ang1 = [] # outer detector angle
38  ang2 = [] # inner detector angle
39  sang = [] # sample rocking angle
40  hkls = [] # Miller indices of the reference reflections
41
42  def hotpixelkill(ccd):
43      """
44      function to remove hot pixels from CCD frames
45      ADD REMOVE VALUES IF NEEDED!
46      """
47      ccd[304,97] = 0
48      ccd[303,96] = 0
49      return ccd
50
51  # read images and angular positions from the data file
52  # this might differ for data taken at different beamlines since
53  # they way how motor positions are stored is not always consistent
54  for imgnr in numpy.sort(list(set(imagenrs)-set(avoid_images))[::4]):
55      filename = os.path.join(imgdir,filetmp%imgnr)
56      edf = xu.io.EDFFile(filename)
57      ccd = hotpixelkill(edf.data)
58      images.append(ccd)
```

```
59      ang1.append(float(edf.header['motor_pos'].split()[4]))
60      ang2.append(float(edf.header['motor_pos'].split()[3]))
61      sang.append(float(edf.header['motor_pos'].split()[1]))
62      if imgnr > 1293.:
63          hkls.append((0,0,0))
64      elif imgnr < 139:
65          hkls.append((0,0,numpy.sqrt(27))) #(3,3,3))
66      else:
67          hkls.append((0,0,numpy.sqrt(75))) #(5,5,5))
68
69  # call the fit for the detector parameters
70  # detector arm rotations and primary beam direction need to be given
71  # in total 8 detector parameters + 2 additional parameters for the reference crystal orientation a
72  # however the 4 misalignment parameters of the detector and the 3 other parameters can be fixed
73  # the fixable parameters are detector tilt azimuth, the detector tilt angle, the detector rotatio
74  # sample tilt, sample tilt azimuth and the x-ray wavelength
75  param,eps = xu.analysis.area_detector_calib_hkl(sang,ang1,ang2,images,hkls,hxrd,Si,['z+','y-'],'x-
76
77  # Following is an example of the output of the summary of the area_detector_calib_hkl function
78  #total time needed for fit: 624.51sec
79  #fitted parameters: epsilon: 9.9159e-08 (2,['Parameter convergence'])
80  #param: (cch1,cch2,pwidth1,pwidth2,tiltazimuth,tilt,detrot,outerangle_offset,sampletilt,stazimuth,
81  #param: 367.12 349.27 6.8187e-05 6.8405e-05 131.4 2.87 -0.390 -0.061 1.201 318.44 0.8254
82  #please check the resulting data (consider setting plot=True)
83  #detector rotation axis / primary beam direction (given by user): ['z+', 'y-'] / x+
84  #detector pixel directions / distance: z- y+ / 1
85  #       detector initialization with: init_area('z-','y+',cch1=367.12,cch2=349.27,Nch1=516,Nch2=
86  #AND ALWAYS USE an (additional) OFFSET of -0.0611deg in the OUTER DETECTOR ANGLE!
87
88  #param,eps = xu.analysis.area_detector_calib(ang1,ang2,images,['z+','y-'],'x+',start=(45,0,0,0),f.
```

4.2 xrayutilities package

4.2.1 Subpackages

xrayutilities.analysis package

Submodules

xrayutilities.analysis.line_cuts module

xrayutilities.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

>> **pos** position of the data points

>> **data** data values

> Returns

>> fwhm value (single float)

xrayutilities.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

qx equidistant array of qx momentum transfer

qz equidistant array of qz momentum transfer

intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

omcenter omega-position at which the omega scan should be extracted

ttcenter 2theta-position at which the omega scan should be extracted

omrange range of the omega scan to extract

npoints number of points of the omega scan

**kwargs: possible keyword arguments:

qrange integration range perpendicular to scan direction

Nint number of subscans used for the integration (optionally)

lam wavelength for use in the conversion to angular coordinates

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

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

Returns

om,omint omega 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)
```

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

return reciprocal space boundaries of omega scan

Parameters

omcenter omega-position at which the omega scan should be extracted

ttcenter 2theta-position at which the omega scan should be extracted

omrange range of the omega scan to extract

npoints number of points of the omega scan

**kwargs: possible keyword arguments:

qrange integration range perpendicular to scan direction

lam wavelength for use in the conversion to angular coordinates

Returns

qx,qz reciprocal 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter** qx-position at which the omega scan should be extracted
>
> **qzcenter** qz-position at which the omega scan should be extracted
>
> **omrange** range of the omega scan to extract
>
> **npoints** number of points of the omega scan

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative omega positions are returned (default: True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,omint** omega 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qzpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis
>
> **bounds** flag to specify if the scan bounds of the extracted scan should be returned (default:False)

Returns

> **qx,qxint** qx 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qz,qzint** qz scan coordinates and intensities

Example

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

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **angrange** integration range in angular direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis
>
> **bounds** flag to specify if the scan bounds of the extracted scan should be returned (default:False)
>
> **intdir** integration direction 'omega': sample rocking angle (default) '2theta': scattering angle

Returns

> **qz,qzint** qz 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)
```

xrayutilities.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

>**qx** equidistant array of qx momentum transfer

>**qz** equidistant array of qz momentum transfer

>**intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

>**omcenter** om-position at which the radial scan should be extracted

>**ttcenter** tt-position at which the radial scan should be extracted

>**ttrange** two theta range of the radial scan to extract

>**npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

>**omrange** integration range perpendicular to scan direction

>**Nint** number of subscans used for the integration (optionally)

>**lam** wavelength for use in the conversion to angular coordinates

>**relative** determines if absolute or relative two theta positions are returned (default=True)

>**bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

>**om,tt,radint** omega,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)
```

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

return reciprocal space boundaries of radial scan

Parameters

>**omcenter** om-position at which the radial scan should be extracted

>**ttcenter** tt-position at which the radial scan should be extracted

>**ttrange** two theta range of the radial scan to extract

>**npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

>**omrange** integration range perpendicular to scan direction

>**lam** wavelength for use in the conversion to angular coordinates

Returns

>**qxrad,qzrad** reciprocal space boundaries of radial scan

Example

```
>>>
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter** qx-position at which the radial scan should be extracted
>
> **qzcenter** qz-position at which the radial scan should be extracted
>
> **ttrange** two theta range of the radial scan to extract
>
> **npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange** integration range perpendicular to scan direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative two theta positions are returned (default=True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,tt,radint** omega,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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **omcenter** om-position at which the 2theta scan should be extracted
>
> **ttcenter** tt-position at which the 2theta scan should be extracted
>
> **ttrange** two theta range of the scan to extract
>
> **npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange** integration range in omega direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative two theta positions are returned (default=True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **tt,ttint** two 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)
```

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

return reciprocal space boundaries of 2theta scan

Parameters

> **omcenter** om-position at which the 2theta scan should be extracted

> **ttcenter** tt-position at which the 2theta scan should be extracted

> **ttrange** two theta range of the 2theta scan to extract

> **npoints** number of points of the 2theta scan

**kwargs: possible keyword arguments:

> **omrange** integration range in omega direction

> **lam** wavelength for use in the conversion to angular coordinates

Returns

> **qxtt,qztt** reciprocal 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

```
>>>
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer

> **qz** equidistant array of qz momentum transfer

> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

> **qxcenter** qx-position at which the 2theta scan should be extracted

> **qzcenter** qz-position at which the 2theta scan should be extracted

> **ttrange** two theta range of the scan to extract

> **npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange** integration range in omega direction

> **Nint** number of subscans used for the integration (optionally)

> **lam** wavelength for use in the conversion to angular coordinates

> **relative** determines if absolute or relative two theta positions are returned (default=True)

> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **tt,ttint** 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

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

`xrayutilities.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,y** coordinates of the point of interest (float)
>
> **xgrid,ygrid** grid coordinates in x and y direction (array)

Returns

> **ix,iy** index of the closest gridpoint (lower left) of the point (x,y)

xrayutilities.analysis.line_cuts3d module

`xrayutilities.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

> **gridder** 3d xrayutilities.Gridder3D object containing the data
>
> **qypos,qzpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qx,qxint** qx scan coordinates and intensities

Example

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

`xrayutilities.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

> **gridder** 3d xrayutilities.Gridder3D object containing the data
>
> **qxpos,qzpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction

> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qy,qyint** qy scan coordinates and intensities

Example

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

xrayutilities.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

> **gridder** 3d xrayutilities.Gridder3D object containing the data
>
> **qxpos,qypos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qz,qzint** qz scan coordinates and intensities

Example

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

xrayutilities.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,z** coordinates of the point of interest (float)
>
> **xgrid,ygrid,zgrid** grid coordinates in x,y,z direction (array)

Returns

> **ix,iy,iz** index of the closest gridpoint (lower left) of the point (x,y,z)

xrayutilities.analysis.misc module

miscellaneous functions helpful in the analysis and experiment

xrayutilities.analysis.misc.**getangles**(*peak*, *sur*, *inp*)

calculates the chi and phi angles for a given peak

Parameter

> **peak** array which gives hkl for the peak of interest
>
> **sur** hkl of the surface
>
> **inp** inplane 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])

xrayutilities.analysis.sample_align module

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

xrayutilities.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*, *plotlog=False*, *debug=False*)

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

> **angle1** outer detector arm angle
>
> **angle2** inner detector arm angle
>
> **ccdimages** images of the ccd taken at the angles given above
>
> **detaxis** detector arm rotation axis :default: ['z+','y-']
>
> **r_i** primary beam direction [xyz][+-] default 'x+'

> **Keyword_arguments**
>
> > **plot** flag to determine if results and intermediate results should be plotted :default: True
> >
> > **cut_off** cut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%
> >
> > **start** sequence 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.
> >
> > **fix** fix parameters of start (default: (False,False,False,False))
> >
> > **fig** matplotlib figure used for plotting the error :default: None (creates own figure)
> >
> > **wl** wavelength of the experiment in Angstrom (default: config.WAVELENGTH) value does not matter here and does only affect the scaling of the error
> >
> > **plotlog** flag to specify if the created error plot should be on log-scale
> >
> > **debug** flag to specify that you want to see verbose output and saving of images to show if the CEN determination works

xrayutilities.analysis.sample_align.**area_detector_calib_hkl**(*sampleang, angle1, angle2, ccdimages, hkls, experiment, material, detaxis, r_i, plot=True, cut_off=0.1, start=(0, 0, 0, 0, 0, 0, 'config'), fix=(False, False, False, False, False, False, False), fig=None, plotlog=False, debug=False*)

function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

in this variant not only scans through the primary beam but also scans at a set of symmetric reflections can be used for the detector parameter determination. for this not only the detector parameters but in addition the sample orientation and wavelength need to be fit. Both images from the primary beam hkl = (0,0,0) and from a symmetric reflection hkl = (h,k,l) need to be given for a successful run.

parameters

> **sampleang** sample rocking angle (needed to align the reflections (same rotation direction as inner detector rotation)) other sample angle are not allowed to be changed during the scans
>
> **angle1** outer detector arm angle
>
> **angle2** inner detector arm angle
>
> **ccdimages** images of the ccd taken at the angles given above
>
> **hkls** array/list of hkl values for every image
>
> **experiment** Experiment class object needed to get the UB matrix for the hkl peak treatment
>
> **material** material used as reference crystal
>
> **detaxis** detector arm rotation axis :default: ['z+','y-']
>
> **r_i** primary beam direction [xyz][+-] default 'x+'

Keyword_arguments

> **plot** flag to determine if results and intermediate results should be plotted :default: True
>
> **cut_off** cut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%
>
> **start** sequence of start values of the fit for parameters, which can not be estimated automatically these are: tiltazimuth,tilt,detector_rotation,outerangle_offset, stilt,stazimuth, wavelength. By default (0,0,0,0,0,0,'config') is used.
>
> **fix** fix parameters of start (default: (False,False,False,False,False,False,False))
>
> **fig** matplotlib figure used for plotting the error :default: None (creates own figure)
>
> **plotlog** flag to specify if the created error plot should be on log-scale
>
> **debug** flag to tell if you want to see debug output of the script (switch this to true only if you can handle it :))

xrayutilities.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03, 0.03)*, *peaktype='Gauss'*, *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 (xrayutilities.math.Gauss2d) or Lorentzian (xrayutilities.math.Lorentz2d)

PLEASE ALWAYS CHECK THE RESULT CAREFULLY!

Parameter

> **om,tt** angular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]
>
> **psd** intensity values needed for fitting
>
> **omalign** aligned omega value, used as first guess in the fit
>
> **ttalign** aligned 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
>
> **exphxrd** experiment class used for the conversion between angular and reciprocal space.
>
> **frange** data range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})
>
> **peaktype** can be 'Gauss' or 'Lorentz' to fit either of the two peak shapes
>
> **plot** if 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/Lorentzian) as well as their errors

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

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

parameters

> **angle** array of angles in degree of measured detector spectra
>
> **mca_spectra** corresponding detector spectra :(shape: (len(angle),Nchannels)

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

> **r_i** primary beam direction as vector [xyz][+-]; default: 'y+'
>
> **detaxis** detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

selected options from psd_chdeg:

> **plot** flag to specify if a visualization of the fit should be done
>
> **usetilt** whether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: True)

Note: Note: see help of psd_chdeg for more options

---

returns

> pixelwidth (at one meter distance) , center_channel[, detector_tilt]

> **Note:** Note: L/pixelwidth*pi/180 ~= channel/degree, with the sample detector distance L

---

pixelwidth is negative in case the hit channel number decreases upon an increase of the detector angle The function also prints out how a linear detector can be initialized using the results obtained from this calibration. Carefully check the results

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

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

> **phi** azimuths in which the reflection was aligned (deg)
>
> **aomega** aligned omega values (deg)
>
> **zeros** (optional) angles at which surface is parallel to the beam (deg). For the analysis the angles (aomega-zeros) are used.
>
> **omega0** if 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.
>
> **plot** flag to specify if a visualization of the fit is wanted. :default: True

Returns

[omega0,phi0,miscut]

list with fitted values for

> **omega0** the omega value of the reflection should be close to the nominal one
>
> **phi0** the azimuth in which the primary beam looks upstairs
>
> **miscut** amplitude of the sinusoidal variation == miscut angle

`xrayutilities.analysis.sample_align.`**`psd_chdeg`**(*angles*, *channels*, *stdev=None*, *usetilt=True*, *plot=True*, *datap='kx'*, *modelline='r--'*, *modeltilt='b-'*, *fignum=None*, *mlabel='fit'*, *mtilt-label='fit w/tilt'*, *dlabel='data'*, *figtitle=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

> **angles** detector angles for which the position of the beam was measured
>
> **channels** detector channels where the beam was found

keyword arguments:

> **stdev** standard deviation of the beam position
>
> **plot** flag to specify if a visualization of the fit should be done
>
> **usetilt** whether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: True)
>
> **datap** plot format of data points
>
> **modelline** plot format of modelline
>
> **modeltilt** plot format of modeltilt

---

fignum figure number to use for the plot

mlabel label of the model w/o tilt to be used in the plot

mtiltlabel label of the model with tilt to be used in the plot

dlabel label of the data line to be used in the plot

figtitle boolean to tell if the figure title should show the fit parameters

Returns

(pixelwidth,centerch,tilt)

Pixelwidth the width of one detector channel @ 1m distance, which is negative in case the hit channel number decreases upon an increase of the detector angle.

Centerch center channel of the detector

Tilt tilt of the detector from perpendicular to the beam (will be zero in case of usetilt=False)

Note: Note: L/pixelwidth*pi/180 = channel/degree for large detector distance with the sample detector disctance L

---

xrayutilities.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

primarybeam primary beam channel number

angles list or numpy.array with angles

channels list or numpy.array with corresponding detector channels

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

Returns

omega angle 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])
```

Module contents

xrayutilities.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.

xrayutilities.io package

Submodules

xrayutilities.io.cbf module

class xrayutilities.io.cbf.**CBFDirectory**(*datapath*, *ext='cbf'*, ***keyargs*)

Bases: object

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

Save2HDF5 (*h5*, *group=''*, *comp=True*)
> Saves the data stored in the CBF 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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class `xrayutilities.io.cbf.`**CBFFile** (*fname*, *nxkey='X-Binary-Size-Fastest-Dimension'*, *nykey='X-Binary-Size-Second-Dimension'*, *dtkey='DataType'*, *path=None*)
> Bases: `object`

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

Save2HDF5 (*h5*, *group='/'*, *comp=True*)
> 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 (default to the root of the file) :comp: activate compression - true by default

xrayutilities.io.desy_tty08 module

class for reading data+header information from tty08 data files

tty08 is system used at beamline P08 at Hasylab Hamburg and creates simple ASCII files to save the data. Information is easily read from the multicolumn data file. the functions below enable also to parse the information of the header

`xrayutilities.io.desy_tty08.`**gettty08_scan** (*scanname*, *scannumbers*, **args*)
> function to obtain the angular cooridinates as well as intensity values saved in TTY08 datafiles. Especially usefull for reciprocal space map measurements, and to combine date from several scans

> 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

>> **scanname** name of the scans, for multiple scans this needs to be a template string

>> **scannumber** number 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)

> 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.gettty08_scan('text%05d.dat',36,'omega','gamma')
```

class `xrayutilities.io.desy_tty08.`**`tty08File`** (*filename*, *path=None*, *mcadir=None*)

Bases: `object`

Represents a tty08 data file. The file is read during the Constructor call. This class should work for data stored at beamline P08 using the tty08 acquisition system.

Required constructor arguments:

> **filename** a string with the name of the tty08-file

Optional keyword arguments:

> **mcadir** directory name of MCA files

`Read` ()

Read the data from the file

`ReadMCA` ()

xrayutilities.io.edf module

class `xrayutilities.io.edf.`**`EDFDirectory`** (*datapath*, *ext='edf'*, ***keyargs*)

Bases: `object`

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

`Save2HDF5` (*h5*, *group=''*, *comp=True*)

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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class `xrayutilities.io.edf.`**`EDFFile`** (*fname*, *nxkey='Dim_1'*, *nykey='Dim_2'*, *dtkey='DataType'*, *path=''*, *header=True*)

Bases: `object`

`ReadData` ()

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

`Save2HDF5` (*h5*, *group='/'*, *comp=True*)

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 (default to the root of the file) :comp: activate compression - true by default

xrayutilities.io.helper module

convenience functions to open files for various data file reader

these functions should be used in new parsers since they transparently allow to open gzipped and bzipped files

`xrayutilities.io.helper.`**`xu_open`** (*filename*, *mode='rb'*)

function to open a file no matter if zipped or not. Files with extension '.gz' or '.bz2' are assumed to be compressed and transparently opened to read like usual files.

Parameters

> > **filename** filename of the file to open (full including path)

> > **mode** mode in which the file should be opened

> Returns

> > file handle of the opened file

> If the file does not exist an IOError is raised by the open routine, which is not caught within the function

xrayutilities.io.imagereader module

class xrayutilities.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

> > > **filename** filename 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 xrayutilities.io.imagereader.**PerkinElmer**(***keyargs*)
> Bases: `xrayutilities.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 xrayutilities.io.imagereader.**RoperCCD**(***keyargs*)
> Bases: `xrayutilities.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.

xrayutilities.io.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 xrayutilities.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 `xrayutilities.io.panalytical_xml.`**`XRDMLMeasurement`**(*measurement*)

    Bases: `object`

    class to handle scans in a XRDML datafile

`xrayutilities.io.panalytical_xml.`**`getOmPixcel`**(*omraw*, *ttraw*)

    function to reshape the Omega values into a form needed for further treatment with xrayutilities

`xrayutilities.io.panalytical_xml.`**`getxrdml_map`**(*filetemplate*, *scannrs=None*, *path='.'*, *roi=None*)

    parses multiple XRDML file and concatenates the results for parsing the xrayutilities.io.XRDMLFile class is used. The function can be used for parsing maps measured with the PIXCel and point detector.

    Parameter

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

        scannrs int or list of scan numbers

        path common path to the filenames

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

    Returns

        om,tt,psd as flattened numpy arrays

    Example

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

`xrayutilities.io.panalytical_xml.`**`getxrdml_scan`**(*filetemplate*, **motors*, ***kwargs*)

    parses multiple XRDML file and concatenates the results for parsing the xrayutilities.io.XRDMLFile class is used. The function can be used for parsing arbitrary scans and will return the the motor values of the scan motor and additionally the positions of the motors given by in the "*motors" argument

    Parameter

        filetemplate template string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames given by the scannrs keyword argument

        *motors: motor names to return: e.g.: 'Omega','2Theta',... one can also use abbreviations 'Omega' = 'om' = 'o' '2Theta' = 'tt' = 't' 'Chi' = 'c' 'Phi' = 'p'

        **kwargs:

            scannrs int or list of scan numbers

            path common path to the filenames

    Returns

        scanmot,mot1,mot2,...,detectorint: as flattened numpy arrays

    Example

```
>>> scanmot,om,tt,inte = xrayutilities.io.getxrdml_scan("samplename_1.xrdml",'om','tt',path="
```

xrayutilities.io.radicon module

python module for converting radicon data to HDF5

`xrayutilities.io.radicon.`**`hst2hdf5`**(*h5*, *hstfile*, *nofchannels*, *h5path='/'*, *hstpath='.'*)

    Converts a HST file to an HDF5 file.

    Required input arguments:

        h5 HDF5 object where to store the data

> > **hstfile** name of the HST file
>
> > **nofchannels** number of channels
>
> > **optional input arguments:**
>
> > > **h5path** Path in the HDF5 file where to store the data
> >
> > > **hstpath** path where the HST file is located (default is the current working directory)

xrayutilities.io.radicon.**rad2hdf5**(*h5*, *rdcfile*, *h5path='/'*, *rdcpath='.'*)
> Converts a RDC file to an HDF5 file.
>
> > **Required input arguments:**
>
> > > **h5** HDF5 object where to store the data
> >
> > > **rdcfile** name of the RDC file
>
> > **optional input arguments:**
>
> > > **h5path** Path in the HDF5 file where to store the data (default to root)
> >
> > > **rdcpath** path where the RDC file is located (default is the current working directory)

xrayutilities.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_limit** exposure time limit
> >
> > > **mca_info** pytables table with the exposure data
> >
> > > **mca_array** array with all the MCA spectra
>
> > **return value:** a numpy array with the selected mca spectra of shape (hstnr,channels).

xrayutilities.io.rotanode_alignment module

parser for the alignment log file of the rotating anode

class xrayutilities.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
> >
> > > **pname** peakname for which the alignment should be plotted

xrayutilities.io.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 xrayutilities.io.seifert.**SeifertHeader**

    Bases: object

    save_h5_attribs(*obj*)

class xrayutilities.io.seifert.**SeifertMultiScan**(*filename*, *m_scan*, *m2*, *path=None*)

    Bases: object

    dump2hdf5(*h5*, *iname='INT'*, *group='/'*)

        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 keyword arguments:

            inamename for the intensity matrix

            grouppath to the HDF5 group where to store the data

    dump2mlab(*fname*, **args*)

        Store the data in a matlab file.

    parse()

class xrayutilities.io.seifert.**SeifertScan**(*filename*, *path=None*)

    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 aruguments:

            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()

xrayutilities.io.seifert.**getSeifert_map**(*filetemplate*, *scannrs=None*, *path='.'*, *scant-ype='map'*, *Nchannels=1280*)

> parses multiple Seifert *.nja files and concatenates the results. for parsing the xrayutili-ties.io.SeifertMultiScan class is used. The function can be used for parsing maps measured with the Me-teor1D and point detector.

> Parameter

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

>> **scannrs**int or list of scan numbers

>> **path**common path to the filenames

>> **scantype**type of datafile: "map": reciprocal space map measured with a regular Seifert job "tsk": MCA spectra measured using the TaskInterpreter

>> **Nchannels**number of channels of the MCA (needed for "tsk" measurements)

> Returns

>> **om,tt,psd**as flattened numpy arrays

> Example

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

xrayutilities.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 _.

xrayutilities.io.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 xrayutilities.io.spec.**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)

> Bases: object

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

class xrayutilities.io.spec.**SPECFile**(*filename*, *path=''*)

> 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*, *comp=True*)

>> 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:**

>>> **h5f**a HDF5 file object or its filename

>> **optional keyword arguments:**

> **comp**activate compression - true by default

>   **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 xrayutilities.io.spec.**SPECLog**(*filename*, *prompt*, *path=''*)
> Bases: object

>   **Parse()**

>   **Update()**

class xrayutilities.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 xrayutilities.io.spec.**SPECMCAFile**
> Bases: xrayutilities.io.spec.SPECMCA

>   **ReadData()**

class xrayutilities.io.spec.**SPECMCAInline**
> Bases: xrayutilities.io.spec.SPECMCA

>   **ReadData()**

class xrayutilities.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*, *group='/'*, *title=''*, *desc=''*, *optattrs={}*, *comp=True*)
>> 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:**

>>> **h5f**a HDF5 file object or its filename

>> **optional keyword arguments:**

>>> **group**name or group object of the HDF5 group where to store the data

>>> **title**a string with the title for the data, defaults to the name of scan if empty

>>> **desc**a string with the description of the data, defaults to the scan command if empty

>>> **optattrs**a dictionary with optional attributes to store for the data

>>> **comp**activate 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_format number of columns used to save the data

        mca_channels number of MCA channels stored

        mca_start first channel that is stored

        mca_stop last 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.

    Parameters

        *args: arguments for the plot: first argument is the name of x-value column the following pairs of arguments are the y-value names and plot styles allowed are 3,5,7,... number of arguments

        **keyargs:

            newfig if True a new figure instance will be created otherwise an existing one will be used

            logy if True a semilogy plot will be done

xrayutilities.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

        h5f file object of a HDF5 file opened using pytables or its filename

        scans number 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):

        samplename string 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')
```

xrayutilities.io.spectra module

module to handle spectra data

class xrayutilities.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:

> **filename** a string with the name of the SPECTRA file

Optional keyword arguments:

> **mcatmp** template for the MCA files
>
> **mcastart,mcastop** start 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:**
>
> > **h5file** string or HDF5 file object
> >
> > **name** name of the group where to store the data
>
> **optional keyword arguments:**
>
> > **group** root group where to store the data
> >
> > **description** string with a description of the scan
>
> Return value: The method returns None in the case of everything went fine, True otherwise.

class xrayutilities.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 xrayutilities.io.spectra.**SPECTRAFileData**
> Bases: object

> **append**(*col*)

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

class xrayutilities.io.spectra.**SPECTRAFileParameters**
> Bases: dict

class xrayutilities.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:**
> >
> > > **mca** matrix with the MCA data
> > >
> > > **f** filter values along the scan

> > **optional keyword arguments:**
>
> > > **ff** custome filter factors
>
> > **return value:** Array with the same shape as mca with the corrected MCA data.

> **recarray2hdf5** (*h5g*, *rec*, *name*, *desc*)
> > Save a record array in an HDF5 file. A pytables table object is used to store the data.
>
> > **required input arguments:**
>
> > > **h5g** HDF5 group object or path
> > >
> > > **rec** record array
> > >
> > > **name** name of the table in the file
> > >
> > > **desc** description of the table in the file
>
> > **return value:**
>
> > > **tab** a HDF5 table object

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

> **spectra2hdf5** (*dir*, *fname*, *mcatemp*, *name=''*, *desc='SPECTRA data'*)
> > Convert SPECTRA scan data to a HDF5 format.
>
> > **required input arguments:**
>
> > > **dir** directory where the scan is stored
> > >
> > > **fname** name of the SPECTRA data file
> > >
> > > **mcatemp** template for the MCA file names
>
> > **optional keyword arguments:**
>
> > > **name** optional name under which to save the data if empty the basename of the filename will be used
> > >
> > > **desc** optional description of the scan

xrayutilities.io.spectra.**get_spectra_files** (*dirname*)
> Return a list of spectra files within a directory.
>
> **required input arguments:**
>
> > **dirname** name of the directory to search
>
> **return values:** list with filenames

xrayutilities.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
>
> > **h5f** file object of a HDF5 file opened using pytables
> >
> > **scans** number of the scans of the reciprocal space map (int,tuple or list)
>
> ***args: arbitrary number of motor names (strings)**
>
> > **omname** name of the omega motor (or its equivalent)
> >
> > **ttname** name of the two theta motor (or its equivalent)
>
> ****kwargs (optional):**

> **mca** name of the mca data (if available) otherwise None (default: "MCA")
>
> **samplename** string 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']).

xrayutilities.io.spectra.**read_data**(*fname*)

> Read a spectra data file (a file with now MCA data).

> **required input arguments:**

> > **fname** name of the file to read

> **return values: (data,hdr)**

> > **data** numpy record array where the keys are the column names

> > **hdr** a dictionary with header information

xrayutilities.io.spectra.**read_mca**(*fname*)

> Read a single SPECTRA MCA file.

> **required input arguments:**

> > **fname** name of the file to read

> **return value:**

> > **data** a numpy array witht the MCA data

xrayutilities.io.spectra.**read_mca_dir**(*dirname*, *filetemp*, *sort=True*)

> Read all MCA files within a directory

xrayutilities.io.spectra.**read_mcas**(*ftemp*, *cntstart*, *cntstop*)

> Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

Module contents

xrayutilities.materials package

Submodules

xrayutilities.materials.cif module

class xrayutilities.materials.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.

xrayutilities.materials.database module

module to handle access to the optical parameters database

class `xrayutilities.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:

> **en** list or numpy array with energy in (eV)
>
> **f2** list 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:

> **name** string with the name of the material

SetWeight (*weight*)

Save weight of the element as float

required input argument:

> **weight** atomic standard weight of the element (float)

xrayutilities.materials.database.**add_f0_from_intertab** (*db*, *itabfile*)

Read f0 data from international tables of crystallography and add it to the database.

xrayutilities.materials.database.**add_f0_from_xop** (*db*, *xopfile*)

Read f0 data from f0_xop.dat and add it to the database.

xrayutilities.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.

xrayutilities.materials.database.**add_f1f2_from_henkedb** (*db*, *henkefile*)

Read f1 and f2 data from Henke database and add it to the database.

xrayutilities.materials.database.**add_f1f2_from_kissel** (*db*, *kisselfile*)

Read f1 and f2 data from Henke database and add it to the database.

xrayutilities.materials.database.**add_mass_from_NIST** (*db*, *nistfile*)

Read atoms standard mass and save it to the database.

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

xrayutilities.materials.elements module

xrayutilities.materials.lattice module

module handling crystal lattice structures

class xrayutilities.materials.lattice.**Atom** (*name*, *num*)

Bases: object

f (*q*, *en='config'*)

function to calculate the atomic structure factor F

Parameter

> **q** momentum transfer
>
> **en** energy for which F should be calculated, if omitted the value from the xrayutilities configuration is used

Returns

> f (float)

f0 (*q*)

f1 (*en='config'*)

f2 (*en='config'*)

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

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

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

xrayutilities.materials.lattice.**CsClLattice**(*aa*, *ab*, *a*)

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

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

xrayutilities.materials.lattice.**CubicLattice**(*a*)

> Returns a Lattice object representing a simple cubic lattice.

> **required input arguments:**

>> **a** lattice parameter

> **return value:** an instance of Lattice class

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

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

xrayutilities.materials.lattice.**GeTeRhombohedral**(*aa*, *ab*, *a*, *ang*, *x=0.237*)

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

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

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

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

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

class xrayutilities.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:**

>>> **eps** a 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 xrayutilities.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

>> **atom** atom object to be added

>> **pos** position of the atom

>> **occ** occupancy (default=1.0)

>> **b** b-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)

xrayutilities.materials.lattice.**MagnetiteLattice** (*aa*, *ab*, *ac*, *a*, *x=0.255*)

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

xrayutilities.materials.lattice.**NiAsLattice** (*aa*, *ab*, *a*, *c*, *biso=0.0*)

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

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

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

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

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

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

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

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

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

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

xrayutilities.materials.material module

class module implements a certain material

class xrayutilities.materials.material.**Alloy** (*matA*, *matB*, *x*)
> Bases: xrayutilities.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

>>> **hkl** Miller Indices

>>> **sub** substrate material or lattice constant (Instance of Material class or float)

>>> **exp** Experiment class from which the Transformation object and ndir are needed

>> Returns

>>> **qy,qz** reciprocal space coordinates of the corners of the relaxation triangle

> **lattice_const_AB** (*latA*, *latB*, *x*)
>> method to set the composition of the Alloy. x is the atomic fraction of the component B

> **x**

xrayutilities.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

xrayutilities.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 xrayutilities.materials.material.**CubicAlloy**(*matA*, *matB*, *x*)
> Bases: [xrayutilities.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_inp**inplane peak position of reflection hkl of the alloy in reciprocal space

>>> **q_perp**perpendicular peak position of the reflection hkl of the alloy in reciprocal space

>>> **hkl**Miller indices of the measured asymmetric reflection

>>> **sur**Miller 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_perp**perpendicular peak position of the reflection hkl of the alloy in reciprocal space

>>> **hkl**Miller indices of the measured symmetric reflection (also defines the surface normal

>>> **inpr**Miller indices of a Bragg peak defining the inplane reference direction

>>> **asub**substrate lattice constant

>>> **relax**degree of relaxation (needed to obtain the content from symmetric reciprocal space position)

>> Returns

>>> **content**the content of B in the alloy determined from the input variables

xrayutilities.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,c44** independent components of the elastic tensor of cubic materials

Returns

> 6x6 materix with elastic constants

xrayutilities.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,c** unit cell extenstions (Angstrom)
>
> **alpha** angle between unit cell vectors b,c
>
> **beta** angle between unit cell vectors a,c
>
> **gamma** angle between unit cell vectors a,b

returns a Material object with the specified properties

xrayutilities.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,c44** independent components of the elastic tensor of a hexagonal material

Returns

> 6x6 materix with elastic constants

class xrayutilities.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:**
>
> > **hkl** list 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

> **q** vectorial momentum transfer (vectors as list,tuple or numpy array are valid)
>
> **en** energy in eV, if omitted the value from the xrayutilities configuration is used
>
> **temp** temperature 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

q0 vectorial momentum transfer (vectors as list,tuple or numpy array are valid)

en list, tuple or array of energy values in eV

temp temperature 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

q vectorial 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)])

en0 energy value in eV, if omitted the value from the xrayutilities configuration is used

temp temperature 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

en x-ray energy eV, if omitted the value from the xrayutilities 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'*)

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

Parameter

q momentum transfer in (1/A)

en xray energy in eV, if omitted the value from the xrayutilities configuration is used

temp temperature used for Debye-Waller-factor calculation

polarization either '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 xrayutilities 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 xrayutilities 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 xrayutilities configuration is used

 Returns

   delta (float)

distances()
 function to obtain distances of atoms in the crystal up to the unit cell size (largest value of a,b,c is the cut-off)

 returns a list of tuples with distance d and number of occurence n [(d1,n1),(d2,n2),...]

 Note: Note: if the base of the material is empty the list will be empty

---

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 xrayutilities configuration is used

 Returns

   n (complex)

lam

mu

nu

xrayutilities.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 on (001) substrates

 required input arguments:

submat substrate material

layermat bulk material of the layer

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

`xrayutilities.materials.material.`**`index_map_ij2ijkl`**`(ij)`

`xrayutilities.materials.material.`**`index_map_ijkl2ij`**`(i, j)`

xrayutilities.materials.predefined_materials module

class `xrayutilities.materials.predefined_materials.`**`SiGe`**`(x)`
    Bases: `xrayutilities.materials.material.CubicAlloy`

`lattice_const_AB``(latA, latB, x)`
    method to calculate the lattice parameter of the SiGe alloy with composition Si_{1-x}Ge_x

`x`

Module contents

xrayutilities.math package

Submodules

xrayutilities.math.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

`xrayutilities.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,y data coordinates (do NOT need to be regularly spaced)

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

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

drange limits 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_function function which should be fitted must accept the parameters (x,y,*params)

Returns

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

`xrayutilities.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

xdata xcoordinates of the data to be fitted

ydata ycoordinates of the data which should be fit

keyword parameters:

iparams initial paramters for the fit (determined automatically if nothing is given

maxit maximal 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

`xrayutilities.math.fit.`**`multGaussFit`**(*args*, **kwargs*)
: convenience function to keep API stable see multPeakFit for documentation

`xrayutilities.math.fit.`**`multGaussPlot`**(*args*, **kwargs*)
: convenience function to keep API stable see multPeakPlot for documentation

`xrayutilities.math.fit.`**`multPeakFit`**(*x*, *data*, *peakpos*, *peakwidth*, *dranges=None*, *peaktype='Gaussian'*)
: function to fit multiple Gaussian/Lorentzian peaks with linear background to a set of data

Parameters

> **x** x-coordinate of the data
>
> **data** data array with same length as x
>
> **peakpos** initial parameters for the peak positions
>
> **peakwidth** initial values for the peak width
>
> **dranges** list of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as peakpos
>
> **peaktype** type of peaks to be used: can be either 'Gaussian' or 'Lorentzian'

Returns

> pos,sigma,amp,background
>
> **Pos** list of peak positions derived by the fit
>
> **Sigma** list of peak width derived by the fit
>
> **Amp** list of amplitudes of the peaks derived by the fit
>
> **Background** array of background values at positions x

`xrayutilities.math.fit.`**`multPeakPlot`**(*x*, *fpos*, *fwidth*, *famp*, *background*, *dranges=None*, *peaktype='Gaussian'*, *fig='xu_plot'*, *fact=1.0*)
: function to plot multiple Gaussian/Lorentz peaks with background values given by an array

Parameters

> **x** x-coordinate of the data
>
> **fpos** list of positions of the peaks
>
> **fwidth** list of width of the peaks
>
> **famp** list of amplitudes of the peaks
>
> **background** array with background values
>
> **dranges** list of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as fpos
>
> **peaktype** type of peaks to be used: can be either 'Gaussian' or 'Lorentzian'
>
> **fig** matplotlib figure number or name
>
> **fact** factor to use as multiplicator in the plot

`xrayutilities.math.fit.`**`peak_fit`**(*xdata*, *ydata*, *iparams=[ ]*, *peaktype='Gauss'*, *maxit=200*)
: fit function using odr-pack wrapper in scipy similar to :https:

//github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting for Gauss, Lorentz or Pseudovoigt-functions

Parameters

> **xdata** xcoordinates of the data to be fitted
>
> **ydata** ycoordinates of the data which should be fit

keyword parameters:

> **iparams** initial paramters for the fit (determined automatically if nothing is given)
>
> **peaktype** type of peak to fit ('Gauss','Lorentz','PseudoVoigt')
>
> **maxit** maximal iteration number of the fit

Returns

> params,sd_params,itlim

the parameters as defined in function Gauss1d/Lorentz1d or PseudoVoigt1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

xrayutilities.math.functions module

module with several common function needed in xray data analysis

xrayutilities.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
>
> > **x** argument of the Debye function (float)
>
> Returns
>
> > **D1(x)** float value of the Debye function

xrayutilities.math.functions.**Gauss1d**(*x*, **p*)

> function to calculate a general one dimensional Gaussian
>
> Parameters
>
> > **p** list of parameters of the Gaussian [XCEN,SIGMA,AMP,BACKGROUND] for information: SIGMA = FWHM / (2*sqrt(2*log(2)))
> >
> > **x** coordinate(s) where the function should be evaluated
>
> Returns
>
> the value of the Gaussian described by the parameters p at position x
>
> Example
>
> Calling with a list of parameters needs a call looking as shown below (note the '*') or explicit listing of the parameters: >>> Gauss1d(x,*p) >>> Gauss1d(numpy.linspace(0,10,100), 5, 1, 1e3, 0)

xrayutilities.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

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

>   for parameter description see Gauss1d

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

>   Parameters

>>   **p** list 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 in radians

>>   **x,y** coordinate(s) where the function should be evaluated

>   Returns

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

xrayutilities.math.functions.**Gauss3d**(*x*, *y*, *z*, **p*)
>   function to calculate a general three dimensional Gaussian

>   Parameters

>>   **p** list of parameters of the Gauss-function [XCEN,YCEN,ZCEN,SIGMAX,SIGMAY,SIGMAZ,AMP,BACKGROUN
>>   SIGMA = FWHM / (2*sqrt(2*log(2)))

>>   **x,y,z** coordinate(s) where the function should be evaluated

>   Returns

>   the value of the Gaussian described by the parameters p at positions (x,y,z)

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

>   Parameters

>>   **p** list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

>>   **x** coordinate(s) where the function should be evaluated

>   Returns

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

xrayutilities.math.functions.**Lorentz1d_der_p**(*x*, **p*)
>   function to calculate the derivative of a Gaussian with respect the parameters p

>   for parameter description see Lorentz1d

xrayutilities.math.functions.**Lorentz1d_der_x**(*x*, **p*)
>   function to calculate the derivative of a Gaussian with respect to x

>   for parameter description see Lorentz1d

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

>   Parameters

>>   **p** list of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE]
>>   ANGLE = rotation of the X,Y direction of the Lorentzian in radians

>>   **x,y** coordinate(s) where the function should be evaluated

>   Returns

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

`xrayutilities.math.functions.`**`PseudoVoigt1d`**`(*x*, *\*p*)`
    function to calculate a pseudo Voigt function as linear combination of a Gauss and Lorentz peak

    Parameters

        p list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
            :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

        x coordinate(s) where the function should be evaluated

    Returns

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

`xrayutilities.math.functions.`**`PseudoVoigt1dArea`**`(*\*p*)`
    function to calculate the area of a pseudo Voigt function with neglected background

    Parameters

        p list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
            :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

    Returns

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

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

    Parameters

        p list 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 in radians

        x,y coordinate(s) where the function should be evaluated

    Return

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

`xrayutilities.math.functions.`**`kill_spike`**`(*data*, *threshold=2.0*)`
    function to smooth **single** data points which differ from the average of the neighboring data points by
    more than the threshold factor. Such spikes will be replaced by the mean value of the next neighbors.

    > **Warning:** Use this function carefully not to manipulate your data!

    Parameters

        data 1d numpy array with experimental data

        threshold threshold factor to identify strange data points

    Returns

        1d data-array with spikes removed

`xrayutilities.math.functions.`**`smooth`**`(*x*, *n*)`
    function to smooth an array of data by averaging N adjacent data points

    Parameters

        x 1D data array

        n number of data points to average

    Returns

        xsmooth smoothed array with same length as x

xrayutilities.math.transforms module

class xrayutilities.math.transforms.**AxisToZ**(*newzaxis*)

> Bases: xrayutilities.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.

xrayutilities.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

xrayutilities.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 xrayutilities.math.transforms.**CoordinateTransform**(*v1*, *v2*, *v3*)

> Bases: xrayutilities.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 xrayutilities.math.transforms.**Transform**(*matrix*)

> Bases: object

> **inverse**(**args*)

>> performs inverse transformation a vector, matrix or tensor of rank 4

>> Parameters

>>> ***args: object to transform, list or numpy array of shape**(n,) (n,n), (n,n,n,n) where n is the rank of the transformation matrix

xrayutilities.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.

xrayutilities.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.

xrayutilities.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.

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

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

xrayutilities.math.transforms.**mycross**(*vec*, *mat*)

> function implements the cross-product of a vector with each column of a matrix

xrayutilities.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

> **vec**numpy.array or list of length 3
>
> **axis**numpy.array or list of length 3
>
> **ang**rotation angle in degree (deg=True) or in rad (deg=False)
>
> **deg**boolean which determines the input format of ang (default: True)

Returns

> **rotvec**rotated vector as numpy.array

Example

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

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

xrayutilities.math.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.

xrayutilities.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:**
>
> > **v1**vector as numpy array or list
> >
> > **v2**vector 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

xrayutilities.math.vector.**VecDot**(*v1*, *v2*)
> Calculate the vector dot product.
>
> **required input arguments:**
>
> > **v1**vector as numpy array or list
> >
> > **v2**vector as numpy array or list
>
> **return value:**float value

xrayutilities.math.vector.**VecNorm**(*v*)
> Calculate the norm of a vector.
>
> **required input arguments:**
>
> > **v**vector as list or numpy array
>
> **return value:**float holding the vector norm

xrayutilities.math.vector.**VecUnit**(*v*)
> Calculate the unit vector of v.
>
> **required input arguments:**
>
> > **v**vector as list or numpy array

return value: numpy array with the unit vector

xrayutilities.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

xrayutilities.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

Module contents

4.2.2 Submodules

4.2.3 xrayutilities.config module

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

4.2.4 xrayutilities.exception module

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

exception xrayutilities.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)

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

4.2.5 xrayutilities.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 xrayutilities.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**
> >
> > > **B** reciprocal space conversion matrix of a Material. you can specify the matrix B (default identiy matrix) shape needs to be (3,3)
> > >
> > > **mat** Material 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
> > >
> > > **U** orientation matrix U can be given if none is given the orientation defined in the Experiment class is used.
> > >
> > > **dettype** detector 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
>
> > **q** list or numpy array with the reciprocal space position
>
> **optional keyword arguments:**
>
> > **deg** True/False whether the return value should be in degree or radians :(default: True)

Transform (*v*)

> transforms a vector, matrix or tensor of rank 4 (e.g. elasticity tensor) to the coordinate frame of the Experiment class.
>
> Parameters
>
> > **v** object 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 xrayutilities.experiment.**GID** (*idir*, *ndir*, ***keyargs*)

> Bases: [xrayutilities.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,beta** sample 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**

> > **delta** giving 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

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

> > **wl** x-ray wavelength in angstroem (default: self._wl)

> > **deg** flag 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

> > **Q** a list or numpy array of shape (3) with q-space vector components

> **optional keyword arguments:**

> > **trans** True/False apply coordinate transformation on Q

> > **deg** True/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_i** incidence angle to surface (at the moment always 0)

> > **azimuth** sample rotation with respect to the inplane reference direction

> > **twotheta** scattering angle

> > **beta** exit angle from surface (at the moment always 0)

class xrayutilities.experiment.**GISAXS** (*idir*, *ndir*, ***keyargs*)

> Bases: [xrayutilities.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,beta** sample 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 xrayutilities.experiment.**HXRD**(*idir*, *ndir*, *geometry='hi_lo'*, ***keyargs*)
   Bases: xrayutilities.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]
> > >
> > > > **default** self.geometry
> >
> > **refrac** boolean to determine if refraction is taken into account :default: False if True then also a material must be given
> >
> > **mat** Material object; needed to obtain its optical properties for refraction correction, otherwise not used
> >
> > **full_output** boolean to determine if additional output is given to determine scattering angles more accurately in case refraction is set to True :default: False
> >
> > **fi,fd** if 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]
>
> > **omega** incidence angle with respect to surface
> >
> > **chi** sample tilt for the case of non-coplanar geometry
> >
> > **phi** sample azimuth with respect to inplane reference direction
> >
> > **twotheta** scattering angle/detector angle
>
> if full_output: a numpy array of shape (6) with five angles which are [omega,chi,phi,twotheta,psi_i,psi_d]
>
> > **psi_i** offset of the incidence beam from the scattering plane due to refraction
> >
> > **pdi_d** offset ot the diffracted beam from the scattering plane due to refraction

class `xrayutilities.experiment.`**`NonCOP`** (*idir*, *ndir*, ***keyargs*)
> Bases: `xrayutilities.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,tt** sample 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**
> >
> > > **delta** giving 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

---

> > **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)
>
> > **wl** x-ray wavelength in angstroem (default: self._wl)
>
> > **deg** flag 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
>
> > **Q** a list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz
>
> **optional keyword arguments:**
>
> > **trans** True/False apply coordinate transformation on Q (default True)
> >
> > **deg** True/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]
>
> > **omega** sample rocking angle
> >
> > **chi** sample tilt
> >
> > **phi** sample azimuth
> >
> > **twotheta** scattering angle (detector)

class xrayutilities.experiment.**Powder**(*mat, **keyargs*)
Bases: [xrayutilities.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
>
> > **theta** array with angular positions
> >
> > **int** intensity 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:
>
> > **data** array with intensities
> >
> > **ang** angular position of intensities
> >
> > **qpos** reciprocal space position of intensities
>
> **Q2Ang**(*qpos, deg=True*)
> Converts reciprocal space values to theta angles

class xrayutilities.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

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

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

 **kwargs: possible keyword arguments

   delta giving 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

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

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

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

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

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

   sampledis sample displacement vector in same units as the detector distance (default: (0,0,0))

 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

detectorDir1 direction of the detector (along the pixel direction 1); e.g. 'z+' means higher pixel numbers at larger z positions

detectorDir2 direction of the detector (along the pixel direction 2); e.g. 'x+'

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

Nch1 number of detector pixels along direction 1

Nch2 number of detector pixels along direction 2

distance distance of center pixel from center of rotation

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

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

detrot angle of the detector rotation around primary beam direction (used to correct misalignments)

tiltazimuth direction of the tilt vector in the detector plane (in degree)

tilt tilt 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 !!

Note: Note: the channel numbers run from 0 .. NchX-1

**kwargs: optional keyword arguments

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

roi region 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

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

cch center channel, in direction of self.r_i at zero detectorAngles

Nchannel total number of detector channels

distance distance of center channel from center of rotation

pixelwidth width of one pixel (same unit as distance)

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

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

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

Note: Note: the channel numbers run from 0 .. Nchannel-1

**kwargs: optional keyword arguments

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

roi region 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

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

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

**kwargs: possible keyword arguments

delta giving 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

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

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

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

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

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

sampledis sample displacement vector in same units as the detector distance (default: (0,0,0))

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

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

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

**kwargs: optional keyword arguments

delta giving 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

> > **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)
>
> > **wl** x-ray wavelength in angstroem (default: self._wl)
>
> > **deg** flag to tell if angles are passed as degree :(default: True)
>
> > **sampledis** sample displacement vector in relative units of the detector distance (default: (0,0,0))

> 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

4.2.6 xrayutilities.gridder module

class xrayutilities.gridder.**Gridder**
> Bases: object

Clear()
> Clear so far gridded data to reuse this instance of the Gridder

KeepData(*bool*)

Normalize(*bool*)
> set or unset the normalization flag. Normalization needs to be done to obtain proper gridding but may want to be disabled in certain cases when sequential gridding is performed

data
> return gridded data (performs normalization if switched on)

class xrayutilities.gridder.**Gridder1D**(*nx*)
> Bases: xrayutilities.gridder.Gridder

dataRange(*min*, *max*, *fixed=True*)
> define minimum and maximum data range, usually this is deduced from the given data automatically, however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside the range are simply ignored

> Parameters

> > **min** minimum value of the gridding range

> > **max** maximum value of the gridding range

> > **fixed** flag to turn fixed range gridding on (True (default)) or of (False)

xaxis
> Returns the xaxis of the gridder the returned values correspond to the center of the data bins used by the numpy.histogram function

xrayutilities.gridder.**axis**(*min_value*, *max_value*, *n*)
> Compute the a grid axis.

> Parameters

> > **Min_value** axis minimum value

> > **Max_value** axis maximum value

Nnumber of steps

xrayutilities.gridder.**check_array**(*a*, *dtype*)

Check if an array fits the requirements for the C-code and returns it back to the callee. Such arrays must be aligned and C_CONTIGUOUS which means that they have to follow C-ordering.

Parameters

Aarray to check

Dtypenumpy data type

xrayutilities.gridder.**delta**(*min_value*, *max_value*, *n*)

Compute the stepsize along an axis of a grid.

Parameters

Min_valueaxis minimum value

Max_valueaxis maximum value

Nnumber of steps

xrayutilities.gridder.**ones**(**args*)

Compute ones for matrix generation. The shape is determined by the number of input arguments.

4.2.7 xrayutilities.gridder2d module

class xrayutilities.gridder2d.**Gridder2D**(*nx*, *ny*)

Bases: xrayutilities.gridder.Gridder

SetResolution(*nx*, *ny*)

Reset the resolution of the gridder. In this case the original data stored in the object will be deleted.

Parameters

Nxnumber of points in x-direction

Nynumber of points in y-direction

dataRange(*(xmin, xmax)*, *(ymin, ymax)*, *fixed=True*)

define minimum and maximum data range, usually this is deduced from the given data automatically, however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside the range are simply ignored

Parameters

xmin,yminminimum value of the gridding range in x,y

xmax,ymaxmaximum value of the gridding range in x,y

fixedflag to turn fixed range gridding on (True (default)) or of (False)

xaxis

xmatrix

yaxis

ymatrix

4.2.8 xrayutilities.gridder3d module

class xrayutilities.gridder3d.**Gridder3D**(*nx*, *ny*, *nz*)

Bases: xrayutilities.gridder.Gridder

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

dataRange (*(xmin*, *xmax)*, *(ymin*, *ymax)*, *(zmin*, *zmax)*, *fixed=True*)
>   define minimum and maximum data range, usually this is deduced from the given data automatically,
>   however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside
>   the range are simply ignored
>
>   Parameters
>
>>   **xmin,ymin,zmin**minimum value of the gridding range in x,y,z
>>
>>   **xmax,ymax,zmax**maximum value of the gridding range in x,y,z
>>
>>   **fixed**flag to turn fixed range gridding on (True (default)) or of (False)

xaxis

xmatrix

yaxis

ymatrix

zaxis

zmatrix

4.2.9 xrayutilities.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 xrayutilities.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 xrayutilities.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

xrayutilities.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

> **data** data which should be contracted (length N)
>
> **Nav** number of values which should be averaged

Returns

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

`xrayutilities.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

> **data2d** array of 2D data shape (N,M)
>
> **Nav1,2** a field of (Nav1 x Nav2) values is contracted

**kwargs: optional keyword argument

> **roi** region 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))

`xrayutilities.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

> **psddata** array of 2D data shape (Nspectra,Nchannels)
>
> **Nav** number of channels which should be averaged

**kwargs: optional keyword argument

> **roi** region 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))

4.2.10 xrayutilities.q2ang_fit module

Module provides functions to convert a q-vector from reciprocal space to angular space. a simple implementation uses scipy optimize routines to perform a fit for a arbitrary goniometer.

The user is, however, expected to use the bounds variable to put restrictions to the number of free angles to obtain reproducible results. In general only 3 angles are needed to fit an arbitrary q-vector (2 sample + 1 detector angles or 1 sample + 2 detector). More complicated restrictions can be implemented using the lmfit package. (done upon request!)

The function is based on a fitting routine. For a speicific goniometer also analytic expressions from literature can be used.

```
Q2AngFit(qvec, expclass, bounds=None, ormat=array([[ 1.,  0.,  0.],
[ 0.,  1.,  0.],
[ 0.,  0.,  1.]]), startvalues=None)
```
Functions to convert a q-vector from reciprocal space to angular space. This implementation uses scipy optimize routines to perform a fit for a goniometer with arbitrary number of goniometer angles.

The user *must* use the bounds variable to put restrictions to the number of free angles to obtain reproducible results. In general only 3 angles are needed to fit an arbitrary q-vector (2 sample + 1 detector angles or 1 sample + 2 detector).

Parameters

> **qvec**q-vector which should be transformed to reciprocal space
>
> **expclass**experimental class used to define the goniometer for which the angles should be calculated.

> **keyword arguments(optional):**
>
>> **bounds**list of bounds of the goniometer angles. The number of bounds must correspond to the number of goniometer angles in the expclass. Angles can also be fixed by supplying only one value for a particular angle. e.g.: ((low,up),fix,(low2,up2),(low3,up3))
>>
>> **ormat**orientation matrix of the sample to be used in the conversion
>>
>> **startvalues**start values for the fit, which can significantly speed up the conversion. The number of values must correspond to the number of angles in the goniometer of the expclass

Returns

> **fittedangles,errcode**list of fitted goniometer angles and the errcode of the scipy minimize function. for a successful fit the error code should be X

4.2.11 xrayutilities.utilities module

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

xrayutilities.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)
```

4.2.12 xrayutilities.utilities_noconf module

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

xrayutilities.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

xrayutilities.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)
```

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

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

>   Parameter

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

>   Returns

>>   wavelength in Angstrom as float

4.2.13 Module contents

xrayutilities is a Python package for assisting with x-ray diffraction experiments. Its the python package included in *xrayutilities*.

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

>   **Authors** Dominik Kriegner <dominik.kriegner@gmail.com> and Eugen Wintersberger <eugen.wintersberger@desy.de>

4.3 xrayutilities.analysis package

4.3.1 Submodules

4.3.2 xrayutilities.analysis.line_cuts module

`xrayutilities.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
>
>> **pos** position of the data points
>>
>> **data** data values
>
> Returns
>
>> fwhm value (single float)

`xrayutilities.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
>
>> **qx** equidistant array of qx momentum transfer
>>
>> **qz** equidistant array of qz momentum transfer
>>
>> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>>
>> **omcenter** omega-position at which the omega scan should be extracted
>>
>> **ttcenter** 2theta-position at which the omega scan should be extracted
>>
>> **omrange** range of the omega scan to extract
>>
>> **npoints** number of points of the omega scan
>
> ****kwargs: possible keyword arguments:**
>
>> **qrange** integration range perpendicular to scan direction
>>
>> **Nint** number of subscans used for the integration (optionally)
>>
>> **lam** wavelength for use in the conversion to angular coordinates
>>
>> **relative** determines if absolute or relative omega positions are returned (default: True)
>>
>> **bounds** flag to specify if the scan bounds should be returned (default: False)
>
> Returns
>
>> **om,omint** omega 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)
> ```

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

> return reciprocal space boundaries of omega scan

Parameters

> **omcenter** omega-position at which the omega scan should be extracted
>
> **ttcenter** 2theta-position at which the omega scan should be extracted
>
> **omrange** range of the omega scan to extract
>
> **npoints** number of points of the omega scan

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **lam** wavelength for use in the conversion to angular coordinates

Returns

> **qx,qz** reciprocal 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter** qx-position at which the omega scan should be extracted
>
> **qzcenter** qz-position at which the omega scan should be extracted
>
> **omrange** range of the omega scan to extract
>
> **npoints** number of points of the omega scan

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative omega positions are returned (default: True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,omint** omega 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)
```

xrayutilities.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

        qx equidistant array of qx momentum transfer

        qz equidistant array of qz momentum transfer

        intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

        qzpos position at which the line scan should be extracted

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

        qrange integration range perpendicular to scan direction

        qmin,qmax minimum and maximum value of extracted scan axis

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

    Returns

        qx,qxint qx 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)
```

xrayutilities.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

        qx equidistant array of qx momentum transfer

        qz equidistant array of qz momentum transfer

        intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

        qxpos position at which the line scan should be extracted

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

        qrange integration range perpendicular to scan direction

        qmin,qmax minimum and maximum value of extracted scan axis

    Returns

        qz,qzint qz scan coordinates and intensities

    Example

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

xrayutilities.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

        qx equidistant array of qx momentum transfer

        qz equidistant 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)
```

xrayutilities.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)
```

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

> return reciprocal space boundaries of radial scan
>
> Parameters
>
> > **omcenter** om-position at which the radial scan should be extracted
> >
> > **ttcenter** tt-position at which the radial scan should be extracted
> >
> > **ttrange** two theta range of the radial scan to extract
> >
> > **npoints** number of points of the radial scan
>
> ****kwargs: possible keyword arguments:**
>
> > **omrange** integration range perpendicular to scan direction
> >
> > **lam** wavelength for use in the conversion to angular coordinates
>
> Returns
>
> > **qxrad,qzrad** reciprocal space boundaries of radial scan
>
> Example
>
> >>>

`xrayutilities.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
>
> > **qx** equidistant array of qx momentum transfer
> >
> > **qz** equidistant array of qz momentum transfer
> >
> > **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
> >
> > **qxcenter** qx-position at which the radial scan should be extracted
> >
> > **qzcenter** qz-position at which the radial scan should be extracted
> >
> > **ttrange** two theta range of the radial scan to extract
> >
> > **npoints** number of points of the radial scan
>
> ****kwargs: possible keyword arguments:**
>
> > **omrange** integration range perpendicular to scan direction
> >
> > **Nint** number of subscans used for the integration (optionally)
> >
> > **lam** wavelength for use in the conversion to angular coordinates
> >
> > **relative** determines if absolute or relative two theta positions are returned (default=True)
> >
> > **bounds** flag to specify if the scan bounds should be returned (default: False)
>
> Returns
>
> > **om,tt,radint** omega,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)
```

xrayutilities.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
>
> > **qx** equidistant array of qx momentum transfer
> >
> > **qz** equidistant array of qz momentum transfer
> >
> > **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
> >
> > **omcenter** om-position at which the 2theta scan should be extracted
> >
> > **ttcenter** tt-position at which the 2theta scan should be extracted
> >
> > **ttrange** two theta range of the scan to extract
> >
> > **npoints** number of points of the radial scan
>
> ****kwargs: possible keyword arguments:**
>
> > **omrange** integration range in omega direction
> >
> > **Nint** number of subscans used for the integration (optionally)
> >
> > **lam** wavelength for use in the conversion to angular coordinates
> >
> > **relative** determines if absolute or relative two theta positions are returned (default=True)
> >
> > **bounds** flag to specify if the scan bounds should be returned (default: False)
>
> Returns
>
> > **tt,ttint** two 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)
> ```

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

> return reciprocal space boundaries of 2theta scan
>
> Parameters
>
> > **omcenter** om-position at which the 2theta scan should be extracted
> >
> > **ttcenter** tt-position at which the 2theta scan should be extracted
> >
> > **ttrange** two theta range of the 2theta scan to extract
> >
> > **npoints** number of points of the 2theta scan
>
> ****kwargs: possible keyword arguments:**
>
> > **omrange** integration range in omega direction
> >
> > **lam** wavelength for use in the conversion to angular coordinates
>
> Returns
>
> > **qxtt,qztt** reciprocal 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

>>>

xrayutilities.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

> **qx**equidistant array of qx momentum transfer
>
> **qz**equidistant array of qz momentum transfer
>
> **intensity**2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter**qx-position at which the 2theta scan should be extracted
>
> **qzcenter**qz-position at which the 2theta scan should be extracted
>
> **ttrange**two theta range of the scan to extract
>
> **npoints**number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange**integration range in omega direction
>
> **Nint**number of subscans used for the integration (optionally)
>
> **lam**wavelength for use in the conversion to angular coordinates
>
> **relative**determines if absolute or relative two theta positions are returned (default=True)
>
> **bounds**flag to specify if the scan bounds should be returned (default: False)

Returns

> **tt,ttint**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

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

xrayutilities.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,y**coordinates of the point of interest (float)
>
> **xgrid,ygrid**grid coordinates in x and y direction (array)

Returns

> **ix,iy**index of the closest gridpoint (lower left) of the point (x,y)

4.3.3 xrayutilities.analysis.line_cuts3d module

xrayutilities.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

>> **gridder** 3d xrayutilities.Gridder3D object containing the data

>> **qypos,qzpos** position at which the line scan should be extracted

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

>> **qrange** integration range perpendicular to scan direction

>> **qmin,qmax** minimum and maximum value of extracted scan axis

>   Returns

>> **qx,qxint** qx scan coordinates and intensities

>   Example

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

xrayutilities.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

>> **gridder** 3d xrayutilities.Gridder3D object containing the data

>> **qxpos,qzpos** position at which the line scan should be extracted

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

>> **qrange** integration range perpendicular to scan direction

>> **qmin,qmax** minimum and maximum value of extracted scan axis

>   Returns

>> **qy,qyint** qy scan coordinates and intensities

>   Example

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

xrayutilities.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

>> **gridder** 3d xrayutilities.Gridder3D object containing the data

>> **qxpos,qypos** position at which the line scan should be extracted

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

>> **qrange** integration range perpendicular to scan direction

>> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qz,qzint** qz scan coordinates and intensities

Example

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

xrayutilities.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,z** coordinates of the point of interest (float)
>
> **xgrid,ygrid,zgrid** grid coordinates in x,y,z direction (array)

Returns

> **ix,iy,iz** index of the closest gridpoint (lower left) of the point (x,y,z)

4.3.4 xrayutilities.analysis.misc module

miscellaneous functions helpful in the analysis and experiment

xrayutilities.analysis.misc.**getangles**(*peak*, *sur*, *inp*)

> calculates the chi and phi angles for a given peak

Parameter

> **peak** array which gives hkl for the peak of interest
>
> **sur** hkl of the surface
>
> **inp** inplane 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])

4.3.5 xrayutilities.analysis.sample_align module

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

xrayutilities.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*, *plotlog=False*, *debug=False*)

> 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

> **angle1** outer detector arm angle

> **angle2** inner detector arm angle
>
> **ccdimages** images of the ccd taken at the angles given above
>
> **detaxis** detector arm rotation axis :default: ['z+','y-']
>
> **r_i** primary beam direction [xyz][+-] default 'x+'

Keyword_arguments

> **plot** flag to determine if results and intermediate results should be plotted :default: True
>
> **cut_off** cut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%
>
> **start** sequence 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.
>
> **fix** fix parameters of start (default: (False,False,False,False))
>
> **fig** matplotlib figure used for plotting the error :default: None (creates own figure)
>
> **wl** wavelength of the experiment in Angstrom (default: config.WAVELENGTH) value does not matter here and does only affect the scaling of the error
>
> **plotlog** flag to specify if the created error plot should be on log-scale
>
> **debug** flag to specify that you want to see verbose output and saving of images to show if the CEN determination works

xrayutilities.analysis.sample_align.**area_detector_calib_hkl**(*sampleang, angle1, angle2, ccdimages, hkls, experiment, material, detaxis, r_i, plot=True, cut_off=0.1, start=(0, 0, 0, 0, 0, 0, 'config'), fix=(False, False, False, False, False, False, False), fig=None, plotlog=False, debug=False*)

function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

in this variant not only scans through the primary beam but also scans at a set of symmetric reflections can be used for the detector parameter determination. for this not only the detector parameters but in addition the sample orientation and wavelength need to be fit. Both images from the primary beam hkl = (0,0,0) and from a symmetric reflection hkl = (h,k,l) need to be given for a successful run.

parameters

> **sampleang** sample rocking angle (needed to align the reflections (same rotation direction as inner detector rotation)) other sample angle are not allowed to be changed during the scans
>
> **angle1** outer detector arm angle
>
> **angle2** inner detector arm angle
>
> **ccdimages** images of the ccd taken at the angles given above

> > > **hkls**array/list of hkl values for every image
>
> > > **experiment**Experiment class object needed to get the UB matrix for the hkl peak treatment
>
> > > **material**material used as reference crystal
>
> > > **detaxis**detector arm rotation axis :default: ['z+','y-']
>
> > > **r_i**primary beam direction [xyz][+-] default 'x+'
>
> > **Keyword_arguments**
>
> > > **plot**flag to determine if results and intermediate results should be plotted :default: True
>
> > > **cut_off**cut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%
>
> > > **start**sequence of start values of the fit for parameters, which can not be estimated automatically these are: tiltazimuth,tilt,detector_rotation,outerangle_offset, stilt,stazimuth, wavelength. By default (0,0,0,0,0,0,'config') is used.
>
> > > **fix**fix parameters of start (default: (False,False,False,False,False,False,False))
>
> > > **fig**matplotlib figure used for plotting the error :default: None (creates own figure)
>
> > > **plotlog**flag to specify if the created error plot should be on log-scale
>
> > > **debug**flag to tell if you want to see debug output of the script (switch this to true only if you can handle it :))

xrayutilities.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03, 0.03)*, *peaktype='Gauss'*, *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 (xrayutilities.math.Gauss2d) or Lorentzian (xrayutilities.math.Lorentz2d)
>
> PLEASE ALWAYS CHECK THE RESULT CAREFULLY!
>
> Parameter
>
> > > **om,tt**angular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]
>
> > > **psd**intensity values needed for fitting
>
> > > **omalign**aligned omega value, used as first guess in the fit
>
> > > **ttalign**aligned 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
>
> > > **exphxrd**experiment class used for the conversion between angular and reciprocal space.
>
> > > **frange**data range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})
>
> > > **peaktype**can be 'Gauss' or 'Lorentz' to fit either of the two peak shapes
>
> > > **plot**if 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/Lorentzian) as well as their errors

---

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

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

parameters

> **angle** array of angles in degree of measured detector spectra
>
> **mca_spectra** corresponding detector spectra :(shape: (len(angle),Nchannels)

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

> **r_i** primary beam direction as vector [xyz][+-]; default: 'y+'
>
> **detaxis** detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

selected options from psd_chdeg:

> **plot** flag to specify if a visualization of the fit should be done
>
> **usetilt** whether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: True)

Note: Note: see help of psd_chdeg for more options

---

returns

> pixelwidth (at one meter distance) , center_channel[, detector_tilt]

Note: Note: L/pixelwidth*pi/180 ~= channel/degree, with the sample detector distance L

---

pixelwidth is negative in case the hit channel number decreases upon an increase of the detector angle The function also prints out how a linear detector can be initialized using the results obtained from this calibration. Carefully check the results

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

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

> **phi** azimuths in which the reflection was aligned (deg)
>
> **aomega** aligned omega values (deg)
>
> **zeros** (optional) angles at which surface is parallel to the beam (deg). For the analysis the angles (aomega-zeros) are used.
>
> **omega0** if 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.
>
> **plot** flag to specify if a visualization of the fit is wanted. :default: True

Returns

[omega0,phi0,miscut]

list with fitted values for

> **omega0** the omega value of the reflection should be close to the nominal one
>
> **phi0** the azimuth in which the primary beam looks upstairs
>
> **miscut** amplitude of the sinusoidal variation == miscut angle

---

xrayutilities.analysis.sample_align.**psd_chdeg**(*angles*, *channels*, *stdev=None*, *usetilt=True*, *plot=True*, *datap='kx'*, *modelline='r–'*, *modeltilt='b-'*, *fignum=None*, *mlabel='fit'*, *mtiltlabel='fit w/tilt'*, *dlabel='data'*, *figtitle=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

> **angles** detector angles for which the position of the beam was measured
>
> **channels** detector channels where the beam was found

keyword arguments:

> **stdev** standard deviation of the beam position
>
> **plot** flag to specify if a visualization of the fit should be done
>
> **usetilt** whether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: True)
>
> **datap** plot format of data points
>
> **modelline** plot format of modelline
>
> **modeltilt** plot format of modeltilt
>
> **fignum** figure number to use for the plot
>
> **mlabel** label of the model w/o tilt to be used in the plot
>
> **mtiltlabel** label of the model with tilt to be used in the plot
>
> **dlabel** label of the data line to be used in the plot
>
> **figtitle** boolean to tell if the figure title should show the fit parameters

Returns

> (pixelwidth,centerch,tilt)
>
> **Pixelwidth** the width of one detector channel @ 1m distance, which is negative in case the hit channel number decreases upon an increase of the detector angle.
>
> **Centerch** center channel of the detector
>
> **Tilt** tilt of the detector from perpendicular to the beam (will be zero in case of usetilt=False)

Note: Note: L/pixelwidth*pi/180 = channel/degree for large detector distance with the sample detector disctance L

---

xrayutilities.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

> **primarybeam** primary beam channel number
>
> **angles** list or numpy.array with angles
>
> **channels** list or numpy.array with corresponding detector channels
>
> **plot** flag to specify if a visualization of the fit is wanted :default: True

---

Returns

       omega angle 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])
```

4.3.6 Module contents

xrayutilities.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.

4.4 xrayutilities.io package

4.4.1 Submodules

4.4.2 xrayutilities.io.cbf module

class `xrayutilities.io.cbf.`**`CBFDirectory`**(*datapath*, *ext='cbf'*, ***keyargs*)

    Bases: `object`

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

    `Save2HDF5`(*h5*, *group=''*, *comp=True*)

        Saves the data stored in the CBF 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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class `xrayutilities.io.cbf.`**`CBFFile`**(*fname*,   *nxkey='X-Binary-Size-Fastest-Dimension'*, *nykey='X-Binary-Size-Second-Dimension'*, *dtkey='DataType'*, *path=None*)

    Bases: `object`

    `ReadData`()

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

    `Save2HDF5`(*h5*, *group='/'*, *comp=True*)

        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 (default to the root of the file) :comp: activate compression - true by default

4.4.3 xrayutilities.io.desy_tty08 module

class for reading data+header information from tty08 data files

tty08 is system used at beamline P08 at Hasylab Hamburg and creates simple ASCII files to save the data. Information is easily read from the multicolumn data file. the functions below enable also to parse the information of the header

`xrayutilities.io.desy_tty08.`**`gettty08_scan`**(*scanname*, *scannumbers*, **args*)

> function to obtain the angular cooriditates as well as intensity values saved in TTY08 datafiles. Especially usefull for reciprocal space map measurements, and to combine date from several scans
>
> 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
>
> > **scanname** name of the scans, for multiple scans this needs to be a template string
> >
> > **scannumber** number 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)
>
> 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.gettty08_scan('text%05d.dat',36,'omega','gamma')
> ```

class `xrayutilities.io.desy_tty08.`**`tty08File`**(*filename*, *path=None*, *mcadir=None*)

> Bases: `object`
>
> Represents a tty08 data file. The file is read during the Constructor call. This class should work for data stored at beamline P08 using the tty08 acquisition system.
>
> Required constructor arguments:
>
> > **filename** a string with the name of the tty08-file
>
> Optional keyword arguments:
>
> > **mcadir** directory name of MCA files
>
> **`Read`**()
> > Read the data from the file
>
> **`ReadMCA`**()

4.4.4 xrayutilities.io.edf module

class `xrayutilities.io.edf.`**`EDFDirectory`**(*datapath*, *ext='edf'*, ***keyargs*)

> Bases: `object`
>
> Parses a directory for EDF files, which can be stored to a HDF5 file for further usage
>
> **`Save2HDF5`**(*h5*, *group=''*, *comp=True*)
> > 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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class xrayutilities.io.edf.**EDFFile**(*fname*, *nxkey='Dim_1'*, *nykey='Dim_2'*, *dtkey='DataType'*, *path=''*, *header=True*)

> Bases: object

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

> **Save2HDF5**(*h5*, *group='/'*, *comp=True*)
>> 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 (default to the root of the file) :comp: activate compression - true by default

4.4.5 xrayutilities.io.helper module

convenience functions to open files for various data file reader

these functions should be used in new parsers since they transparently allow to open gzipped and bzipped files

xrayutilities.io.helper.**xu_open**(*filename*, *mode='rb'*)
> function to open a file no matter if zipped or not. Files with extension '.gz' or '.bz2' are assumed to be compressed and transparently opened to read like usual files.

> Parameters

>> **filename** filename of the file to open (full including path)

>> **mode** mode in which the file should be opened

> Returns

>> file handle of the opened file

> If the file does not exist an IOError is raised by the open routine, which is not caught within the function

4.4.6 xrayutilities.io.imagereader module

class xrayutilities.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

>>> **filename** filename 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 `xrayutilities.io.imagereader.`**`PerkinElmer`**(***keyargs*)
  Bases: `xrayutilities.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 `xrayutilities.io.imagereader.`**`RoperCCD`**(***keyargs*)
  Bases: `xrayutilities.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.

4.4.7 xrayutilities.io.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 `xrayutilities.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 `xrayutilities.io.panalytical_xml.`**`XRDMLMeasurement`**(*measurement*)
  Bases: `object`

  class to handle scans in a XRDML datafile

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

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

  Parameter

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

  > **scannrs** int or list of scan numbers

  > **path** common path to the filenames

  > **roi** region of interest for the PIXCel detector, for other measurements this i not usefull!

  Returns

  > **om,tt,psd** as flattened numpy arrays

  Example

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

`xrayutilities.io.panalytical_xml.`**`getxrdml_scan`**(*filetemplate*, **motors*, ***kwargs*)
  parses multiple XRDML file and concatenates the results for parsing the xrayutilities.io.XRDMLFile class is used. The function can be used for parsing arbitrary scans and will return the the motor values of the scan motor and additionally the positions of the motors given by in the "*motors" argument

  Parameter

> **filetemplate**template string for the file names, can contain a %d which is replaced by
> the scan number or be a list of filenames given by the scannrs keyword argument

> ***motors: motor names to return: e.g.: 'Omega','2Theta',...**one can also use abbreviations
> 'Omega' = 'om' = 'o' '2Theta' = 'tt' = 't' 'Chi' = 'c' 'Phi' = 'p'

> ****kwargs:**

> > **scannrs**int or list of scan numbers

> > **path**common path to the filenames

Returns

> scanmot,mot1,mot2,...,detectorint: as flattened numpy arrays

Example

```
>>> scanmot,om,tt,inte = xrayutilities.io.getxrdml_scan("samplename_1.xrdml",'om','tt',path="
```

4.4.8 xrayutilities.io.radicon module

python module for converting radicon data to HDF5

xrayutilities.io.radicon.**hst2hdf5**(*h5*, *hstfile*, *nofchannels*, *h5path='/'*, *hstpath='.'*)
    Converts a HST file to an HDF5 file.

> **Required input arguments:**

> > **h5**HDF5 object where to store the data

> > **hstfile**name of the HST file

> > **nofchannels**number of channels

> **optional input arguments:**

> > **h5path**Path in the HDF5 file where to store the data

> > **hstpath**path where the HST file is located (default is the current working directory)

xrayutilities.io.radicon.**rad2hdf5**(*h5*, *rdcfile*, *h5path='/'*, *rdcpath='.'*)
    Converts a RDC file to an HDF5 file.

> **Required input arguments:**

> > **h5**HDF5 object where to store the data

> > **rdcfile**name of the RDC file

> **optional input arguments:**

> > **h5path**Path in the HDF5 file where to store the data (default to root)

> > **rdcpath**path where the RDC file is located (default is the current working directory)

xrayutilities.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_limit**exposure time limit

> > **mca_info**pytables table with the exposure data

> > **mca_array**array with all the MCA spectra

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

4.4.9 xrayutilities.io.rotanode_alignment module

parser for the alignment log file of the rotating anode

class xrayutilities.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
> >
> > > **pname** peakname for which the alignment should be plotted

4.4.10 xrayutilities.io.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 xrayutilities.io.seifert.**SeifertHeader**
> Bases: object
>
> **save_h5_attribs** (*obj*)

class xrayutilities.io.seifert.**SeifertMultiScan** (*filename*, *m_scan*, *m2*, *path=None*)
> Bases: object
>
> **dump2hdf5** (*h5*, *iname='INT'*, *group='/'*)
> > 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:**
> >
> > > **h5** a HDF5 file object
> >
> > **optional keyword arguments:**
> >
> > > **iname** name for the intensity matrix
> > >
> > > **group** path to the HDF5 group where to store the data
>
> **dump2mlab** (*fname*, **args*)
> > Store the data in a matlab file.
>
> **parse** ()

class xrayutilities.io.seifert.**SeifertScan** (*filename*, *path=None*)
> Bases: object

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

> **required input arguments:**

>> **h5** HDF5 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:**

>> **group** HDF5 group object where to store the data.

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

> **required input aruguments:**

>> **fname** name 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 ()

xrayutilities.io.seifert.**getSeifert_map** (*filetemplate*, *scannrs=None*, *path='.'*, *scantype='map'*, *Nchannels=1280*)
> parses multiple Seifert *.nja files and concatenates the results. for parsing the xrayutilities.io.SeifertMultiScan class is used. The function can be used for parsing maps measured with the Meteor1D and point detector.

> Parameter

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

>> **scannrs** int or list of scan numbers

>> **path** common path to the filenames

>> **scantype** type of datafile: "map": reciprocal space map measured with a regular Seifert job "tsk": MCA spectra measured using the TaskInterpreter

>> **Nchannels** number of channels of the MCA (needed for "tsk" measurements)

> Returns

>> **om,tt,psd** as flattened numpy arrays

> Example

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

xrayutilities.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 _.

4.4.11 xrayutilities.io.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 `xrayutilities.io.spec.`**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)
>   Bases: `object`
>
>   **Save2HDF5**(*h5*, ***keyargs*)

class `xrayutilities.io.spec.`**SPECFile**(*filename*, *path=''*)
>   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*, *comp=True*)
>   >   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:**
>   >   >   **h5f** a HDF5 file object or its filename
>   >
>   >   **optional keyword arguments:**
>   >   >   **comp** activate compression - true by default
>
>   **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 `xrayutilities.io.spec.`**SPECLog**(*filename*, *prompt*, *path=''*)
>   Bases: `object`
>
>   **Parse**()
>
>   **Update**()

class `xrayutilities.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 `xrayutilities.io.spec.`**SPECMCAFile**
>   Bases: `xrayutilities.io.spec.SPECMCA`
>
>   **ReadData**()

class `xrayutilities.io.spec.`**SPECMCAInline**
>   Bases: `xrayutilities.io.spec.SPECMCA`
>
>   **ReadData**()

class `xrayutilities.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*, *group='/'*, *title=''*, *desc=''*, *optattrs={}*, *comp=True*)

>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:**

>>**h5f** a HDF5 file object or its filename

>**optional keyword arguments:**

>>**group** name or group object of the HDF5 group where to store the data

>>**title** a string with the title for the data, defaults to the name of scan if empty

>>**desc** a string with the description of the data, defaults to the scan command if empty

>>**optattrs** a dictionary with optional attributes to store for the data

>>**comp** activate 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_format** number of columns used to save the data

>>**mca_channels** number of MCA channels stored

>>**mca_start** first channel that is stored

>>**mca_stop** last 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.

>Parameters

>>***args: arguments for the plot: first argument is the name of x-value column** the following pairs of arguments are the y-value names and plot styles allowed are 3,5,7,... number of arguments

>>****keyargs:**

>>>**newfig** if True a new figure instance will be created otherwise an existing one will be used

>>>**logy** if True a semilogy plot will be done

xrayutilities.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

>>**h5f** file object of a HDF5 file opened using pytables or its filename

>>**scans** number 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):**

> > **samplename**string 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

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

4.4.12 xrayutilities.io.spectra module

module to handle spectra data

class `xrayutilities.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:

> > **filename**a string with the name of the SPECTRA file

> Optional keyword arguments:

> > **mcatmp**template for the MCA files

> > **mcastart,mcastop**start 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:**

> > > **h5file**string or HDF5 file object

> > > **name**name of the group where to store the data

> > **optional keyword arguments:**

> > > **group**root group where to store the data

> > > **description**string with a description of the scan

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

class `xrayutilities.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 `xrayutilities.io.spectra.`**`SPECTRAFileData`**
> Bases: `object`

> **`append`**(*col*)

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

class `xrayutilities.io.spectra.`**`SPECTRAFileParameters`**
> Bases: `dict`

class `xrayutilities.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:**

>>> **mca**matrix with the MCA data

>>> **f**filter values along the scan

>> **optional keyword arguments:**

>>> **ff**custome filter factors

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

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

>> **required input arguments:**

>>> **h5g**HDF5 group object or path

>>> **rec**record array

>>> **name**name of the table in the file

>>> **desc**description of the table in the file

>> **return value:**

>>> **tab**a HDF5 table object

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

> **`spectra2hdf5`**(*dir*, *fname*, *mcatemp*, *name=''*, *desc='SPECTRA data'*)
>> Convert SPECTRA scan data to a HDF5 format.

>> **required input arguments:**

>>> **dir**directory where the scan is stored

>>> **fname**name of the SPECTRA data file

>>> **mcatemp**template for the MCA file names

>> **optional keyword arguments:**

>>> **name**optional name under which to save the data if empty the basename of the filename will be used

>>> **desc**optional description of the scan

---

xrayutilities.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

xrayutilities.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: arbitrary number of motor names (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']).

xrayutilities.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

xrayutilities.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

xrayutilities.io.spectra.**read_mca_dir**(*dirname*, *filetemp*, *sort=True*)

    Read all MCA files within a directory

xrayutilities.io.spectra.**read_mcas**(*ftemp*, *cntstart*, *cntstop*)

    Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

4.4.13 Module contents

4.5 xrayutilities.materials package

4.5.1 Submodules

4.5.2 xrayutilities.materials.cif module

class `xrayutilities.materials.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.

4.5.3 xrayutilities.materials.database module

module to handle access to the optical parameters database

class `xrayutilities.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:**
> >
> > > **dbname** name of the database
> > >
> > > **dbdesc** a 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:**
> >
> > > **name** a string with the name of the material
> > >
> > > **description** a string with a description of the material
>
> **`GetF0`**(*q*)
>
> > Obtain the f0 scattering factor component for a particular momentum transfer q.
> >
> > **required input argument:**
> >
> > > **q** single 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:**

> > > > **en** float or numpy array with the energy

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

> > > > **required input arguments:**

> > > > > **en** float 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:**

> > > > > **parameters** list or numpy array with the fit parameters

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

> > > > **required input arguments:**

> > > > > **en** list or numpy array with energy in (eV)

> > > > > **f1** list or numpy array with f1 values

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

> > > > **required input arguments:**

> > > > > **en** list or numpy array with energy in (eV)

> > > > > **f2** list 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.

> > > > **reqiered input arguments:**

> > > > > **name** string with the name of the material

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

> > > > **required input argument:**

> > > > > **weight** atomic standard weight of the element (float)

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

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

xrayutilities.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.

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

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

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

---

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

4.5.4 xrayutilities.materials.elements module

4.5.5 xrayutilities.materials.lattice module

module handling crystal lattice structures

class xrayutilities.materials.lattice.**Atom**(*name*, *num*)

    Bases: object

    f(*q*, *en='config'*)

        function to calculate the atomic structure factor F

        Parameter

                q momentum transfer

                en energy for which F should be calculated, if omitted the value from the xrayutilities configuration is used

        Returns

            f (float)

    f0(*q*)

    f1(*en='config'*)

    f2(*en='config'*)

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

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

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

xrayutilities.materials.lattice.**CsClLattice**(*aa*, *ab*, *a*)

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

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

xrayutilities.materials.lattice.**CubicLattice**(*a*)

    Returns a Lattice object representing a simple cubic lattice.

    required input arguments:

        a lattice parameter

    return value: an instance of Lattice class

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

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

xrayutilities.materials.lattice.**GeTeRhombohedral**(*aa*, *ab*, *a*, *ang*, *x=0.237*)

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

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

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

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

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

class `xrayutilities.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:**
> >
> > > **eps** a 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 `xrayutilities.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
> >
> > > **atom** atom object to be added
> > >
> > > **pos** position of the atom
> > >
> > > **occ** occupancy (default=1.0)
> > >
> > > **b** b-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)

`xrayutilities.materials.lattice.`**`MagnetiteLattice`**(*aa*, *ab*, *ac*, *a*, *x=0.255*)

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

`xrayutilities.materials.lattice.`**`NiAsLattice`**(*aa*, *ab*, *a*, *c*, *biso=0.0*)

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

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

`xrayutilities.materials.lattice.`**`RockSaltLattice`**(*aa*, *ab*, *a*)

`xrayutilities.materials.lattice.`**`RockSalt_Cubic_Lattice`**(*aa*, *ab*, *a*)

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

`xrayutilities.materials.lattice.`**`TetragonalIndiumLattice`**(*aa*, *a*, *c*)

`xrayutilities.materials.lattice.`**`TetragonalTinLattice`**(*aa*, *a*, *c*)

`xrayutilities.materials.lattice.`**`TrigonalR3mh`**(*aa*, *a*, *c*)

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

---

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

4.5.6 xrayutilities.materials.material module

class module implements a certain material

class xrayutilities.materials.material.**Alloy**(*matA*, *matB*, *x*)

> Bases: xrayutilities.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
> >
> > > **hkl** Miller Indices
> > >
> > > **sub** substrate material or lattice constant (Instance of Material class or float)
> > >
> > > **exp** Experiment class from which the Transformation object and ndir are needed
> >
> > Returns
> >
> > > **qy,qz** reciprocal space coordinates of the corners of the relaxation triangle
>
> **lattice_const_AB**(*latA*, *latB*, *x*)
>
> > method to set the composition of the Alloy. x is the atomic fraction of the component B
>
> **x**

xrayutilities.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

xrayutilities.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 xrayutilities.materials.material.**CubicAlloy**(*matA*, *matB*, *x*)

> Bases: xrayutilities.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_inp** inplane peak position of reflection hkl of the alloy in reciprocal space
> > >
> > > **q_perp** perpendicular peak position of the reflection hkl of the alloy in reciprocal space
> > >
> > > **hkl** Miller indices of the measured asymmetric reflection
> > >
> > > **sur** Miller 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_perp perpendicular peak position of the reflection hkl of the alloy in reciprocal space

hkl Miller indices of the measured symmetric reflection (also defines the surface normal

inpr Miller indices of a Bragg peak defining the inplane reference direction

asub substrate lattice constant

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

Returns

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

xrayutilities.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,c44 independent components of the elastic tensor of cubic materials

Returns

6x6 materix with elastic constants

xrayutilities.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,c unit cell extenstions (Angstrom)

alpha angle between unit cell vectors b,c

beta angle between unit cell vectors a,c

gamma angle between unit cell vectors a,b

returns a Material object with the specified properties

xrayutilities.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,c44 independent components of the elastic tensor of a hexagonal material

Returns

6x6 materix with elastic constants

class xrayutilities.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 xrayutilities 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 xrayutilities 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
>
> > **en** x-ray energy eV, if omitted the value from the xrayutilities 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
>
> > **q** momentum transfer in (1/A)
> >
> > **en** xray energy in eV, if omitted the value from the xrayutilities configuration is used
> >
> > **temp** temperature used for Debye-Waller-factor calculation
> >
> > **polarization** either '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
>
> > **en** energy of the x-rays, if omitted the value from the xrayutilities configuration is used
> >
> > **deg** return 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
>
> > **Q** momentum transfer (1/A)
> >
> > **en** x-ray energy (eV), if omitted the value from the xrayutilities configuration is used
>
> Returns
>
> > **deltaTheta** peak 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
>
> > **en** x-ray energy eV, if omitted the value from the xrayutilities configuration is used
>
> Returns
>
> > delta (float)

distances ()
> function to obtain distances of atoms in the crystal up to the unit cell size (largest value of a,b,c is the cut-off)
>
> returns a list of tuples with distance d and number of occurence n [(d1,n1),(d2,n2),...]

Note: Note: if the base of the material is empty the list will be empty

---

idx_refraction (*en='config'*)
> function to calculate the complex index of refraction of a material in the x-ray range
>
> Parameter
>
>> **en** energy of the x-rays, if omitted the value from the xrayutilities configuration is used
>
> Returns
>
>> n (complex)

lam

mu

nu

xrayutilities.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 on (001) substrates
>
> **required input arguments:**
>
>> **submat** substrate material
>>
>> **layermat** bulk material of the layer
>
> **return value:** An instance of Material holding the new pseudomorphically strained material.

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

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

4.5.7 xrayutilities.materials.predefined_materials module

class xrayutilities.materials.predefined_materials.**SiGe** (*x*)
> Bases: xrayutilities.materials.material.CubicAlloy
>
> **lattice_const_AB** (*latA*, *latB*, *x*)
>> method to calculate the lattice parameter of the SiGe alloy with composition Si_{1-x}Ge_x
>
> **x**

4.5.8 Module contents

4.6 xrayutilities.math package

4.6.1 Submodules

4.6.2 xrayutilities.math.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

---

xrayutilities.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,y** data coordinates (do NOT need to be regularly spaced)
  >
  > **data** data set used for fitting (e.g. intensity at the data coords)
  >
  > **start** set of starting parameters for the fit used as first parameter of function fit_function
  >
  > **drange** limits 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_function** function which should be fitted must accept the parameters (x,y,*params)

  Returns

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

xrayutilities.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

  > **xdata** xcoordinates of the data to be fitted
  >
  > **ydata** ycoordinates of the data which should be fit

  keyword parameters:

  > **iparams** initial paramters for the fit (determined automatically if nothing is given
  >
  > **maxit** maximal 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

xrayutilities.math.fit.**multGaussFit**(**args*, ***kwargs*)
  convenience function to keep API stable see multPeakFit for documentation

xrayutilities.math.fit.**multGaussPlot**(**args*, ***kwargs*)
  convenience function to keep API stable see multPeakPlot for documentation

xrayutilities.math.fit.**multPeakFit**(*x*, *data*, *peakpos*, *peakwidth*, *dranges=None*, *peaktype='Gaussian'*)
  function to fit multiple Gaussian/Lorentzian peaks with linear background to a set of data

  Parameters

  > **x** x-coordinate of the data
  >
  > **data** data array with same length as x
  >
  > **peakpos** initial parameters for the peak positions
  >
  > **peakwidth** initial values for the peak width
  >
  > **dranges** list of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as peakpos
  >
  > **peaktype** type of peaks to be used: can be either 'Gaussian' or 'Lorentzian'

  Returns

  > pos,sigma,amp,background

Poslist of peak positions derived by the fit

Sigmalist of peak width derived by the fit

Amplist of amplitudes of the peaks derived by the fit

Backgroundarray of background values at positions x

xrayutilities.math.fit.**multPeakPlot**(*x*, *fpos*, *fwidth*, *famp*, *background*, *dranges=None*, *peaktype='Gaussian'*, *fig='xu_plot'*, *fact=1.0*)

function to plot multiple Gaussian/Lorentz peaks with background values given by an array

Parameters

xx-coordinate of the data

fposlist of positions of the peaks

fwidthlist of width of the peaks

famplist of amplitudes of the peaks

backgroundarray with background values

drangeslist of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as fpos

peaktypetype of peaks to be used: can be either 'Gaussian' or 'Lorentzian'

figmatplotlib figure number or name

factfactor to use as multiplicator in the plot

xrayutilities.math.fit.**peak_fit**(*xdata*, *ydata*, *iparams=*$[\ ]$, *peaktype='Gauss'*, *maxit=200*)

fit function using odr-pack wrapper in scipy similar to :https://github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting for Gauss, Lorentz or Pseudovoigt-functions

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)

peaktypetype of peak to fit ('Gauss','Lorentz','PseudoVoigt')

maxitmaximal iteration number of the fit

Returns

params,sd_params,itlim

the parameters as defined in function Gauss1d/Lorentz1d or PseudoVoigt1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

4.6.3 xrayutilities.math.functions module

module with several common function needed in xray data analysis

xrayutilities.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

> **x** argument of the Debye function (float)

Returns

> **D1(x)** float value of the Debye function

xrayutilities.math.functions.**Gauss1d**(*x*, **p*)

> function to calculate a general one dimensional Gaussian

> Parameters

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

> > **x** coordinate(s) where the function should be evaluated

> Returns

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

> Example

> Calling with a list of parameters needs a call looking as shown below (note the '*') or explicit listing of the parameters: >>> Gauss1d(x,*p) >>> Gauss1d(numpy.linspace(0,10,100), 5, 1, 1e3, 0)

xrayutilities.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

xrayutilities.math.functions.**Gauss1d_der_x**(*x*, **p*)

> function to calculate the derivative of a Gaussian with respect to x

> for parameter description see Gauss1d

xrayutilities.math.functions.**Gauss2d**(*x*, *y*, **p*)

> function to calculate a general two dimensional Gaussian

> Parameters

> > **p** list 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 in radians

> > **x,y** coordinate(s) where the function should be evaluated

> Returns

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

xrayutilities.math.functions.**Gauss3d**(*x*, *y*, *z*, **p*)

> function to calculate a general three dimensional Gaussian

> Parameters

> > **p** list of parameters of the Gauss-function [XCEN,YCEN,ZCEN,SIGMAX,SIGMAY,SIGMAZ,AMP,BACKGROUN SIGMA = FWHM / (2*sqrt(2*log(2)))

> > **x,y,z** coordinate(s) where the function should be evaluated

> Returns

> the value of the Gaussian described by the parameters p at positions (x,y,z)

xrayutilities.math.functions.**Lorentz1d**(*x*, **p*)

> function to calculate a general one dimensional Lorentzian

> Parameters

> > **p** list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

> > **x** coordinate(s) where the function should be evaluated

> Returns

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

`xrayutilities.math.functions.`**`Lorentz1d_der_p`**(*x*, **p*)
> function to calculate the derivative of a Gaussian with respect the parameters p

> for parameter description see Lorentz1d

`xrayutilities.math.functions.`**`Lorentz1d_der_x`**(*x*, **p*)
> function to calculate the derivative of a Gaussian with respect to x

> for parameter description see Lorentz1d

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

> Parameters

> > **p** list of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE]
> > ANGLE = rotation of the X,Y direction of the Lorentzian in radians

> > **x,y** coordinate(s) where the function should be evaluated

> Returns

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

`xrayutilities.math.functions.`**`PseudoVoigt1d`**(*x*, **p*)
> function to calculate a pseudo Voigt function as linear combination of a Gauss and Lorentz peak

> Parameters

> > **p** list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
> > :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

> > **x** coordinate(s) where the function should be evaluated

> Returns

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

`xrayutilities.math.functions.`**`PseudoVoigt1dArea`**(**p*)
> function to calculate the area of a pseudo Voigt function with neglected background

> Parameters

> > **p** list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
> > :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

> Returns

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

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

> Parameters

> > **p** list 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 in radians

> > **x,y** coordinate(s) where the function should be evaluated

> Return

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

`xrayutilities.math.functions.`**`kill_spike`**(*data*, *threshold=2.0*)
>   function to smooth **single** data points which differ from the average of the neighboring data points by more than the threshold factor. Such spikes will be replaced by the mean value of the next neighbors.

>   ---
> | **Warning:** Use this function carefully not to manipulate your data! |
> | --- |
>   ---

>   Parameters
>   >   **data** 1d numpy array with experimental data
>   >
>   >   **threshold** threshold factor to identify strange data points
>
>   Returns
>   >   1d data-array with spikes removed

`xrayutilities.math.functions.`**`smooth`**(*x*, *n*)
>   function to smooth an array of data by averaging N adjacent data points

>   Parameters
>   >   **x** 1D data array
>   >
>   >   **n** number of data points to average
>
>   Returns
>   >   **xsmooth** smoothed array with same length as x

4.6.4 xrayutilities.math.transforms module

class `xrayutilities.math.transforms.`**`AxisToZ`**(*newzaxis*)
>   Bases: `xrayutilities.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.

`xrayutilities.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

`xrayutilities.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 `xrayutilities.math.transforms.`**`CoordinateTransform`**(*v1*, *v2*, *v3*)
>   Bases: `xrayutilities.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 `xrayutilities.math.transforms.`**`Transform`**(*matrix*)
>   Bases: `object`

inverse(**args*)
  performs inverse transformation a vector, matrix or tensor of rank 4

  Parameters

    ***args: object to transform, list or numpy array of shape**(n,) (n,n), (n,n,n,n) where n is the
      rank of the transformation matrix

xrayutilities.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.

xrayutilities.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.

xrayutilities.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.

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

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

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

xrayutilities.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

    vec numpy.array or list of length 3

    axis numpy.array or list of length 3

    ang rotation angle in degree (deg=True) or in rad (deg=False)

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

  Returns

    rotvec rotated vector as numpy.array

  Example

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

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

4.6.5 xrayutilities.math.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.

xrayutilities.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:

    v1 vector as numpy array or list

    v2 vector 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

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

> **required input arguments:**
>
> > **v1**vector as numpy array or list
> >
> > **v2**vector as numpy array or list
>
> **return value:**float value

xrayutilities.math.vector.**VecNorm**(*v*)
    Calculate the norm of a vector.

> **required input arguments:**
>
> > **v**vector as list or numpy array
>
> **return value:**float holding the vector norm

xrayutilities.math.vector.**VecUnit**(*v*)
    Calculate the unit vector of v.

> **required input arguments:**
>
> > **v**vector as list or numpy array
>
> **return value:**numpy array with the unit vector

xrayutilities.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

xrayutilities.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

4.6.6 Module contents

4.7 xrayutilities

4.7.1 xrayutilities package

Subpackages

xrayutilities.analysis package

Submodules

xrayutilities.analysis.line_cuts module

`xrayutilities.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

> > **pos** position of the data points

> > **data** data values

> Returns

> > fwhm value (single float)

`xrayutilities.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

> > **qx** equidistant array of qx momentum transfer

> > **qz** equidistant array of qz momentum transfer

> > **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

> > **omcenter** omega-position at which the omega scan should be extracted

> > **ttcenter** 2theta-position at which the omega scan should be extracted

> > **omrange** range of the omega scan to extract

> > **npoints** number of points of the omega scan

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

> > **qrange** integration range perpendicular to scan direction

> > **Nint** number of subscans used for the integration (optionally)

> > **lam** wavelength for use in the conversion to angular coordinates

> > **relative** determines if absolute or relative omega positions are returned (default: True)

> > **bounds** flag to specify if the scan bounds should be returned (default: False)

> Returns

> > **om,omint** omega 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)
> ```

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

> return reciprocal space boundaries of omega scan

> Parameters

> > **omcenter** omega-position at which the omega scan should be extracted

> **ttcenter** 2theta-position at which the omega scan should be extracted
>
> **omrange** range of the omega scan to extract
>
> **npoints** number of points of the omega scan

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **lam** wavelength for use in the conversion to angular coordinates

Returns

> **qx,qz** reciprocal 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter** qx-position at which the omega scan should be extracted
>
> **qzcenter** qz-position at which the omega scan should be extracted
>
> **omrange** range of the omega scan to extract
>
> **npoints** number of points of the omega scan

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative omega positions are returned (default: True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,omint** omega 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)
```

xrayutilities.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

> **qx** equidistant array of qx momentum transfer

qz equidistant array of qz momentum transfer

intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qzpos position at which the line scan should be extracted

**kwargs: possible keyword arguments:

qrange integration range perpendicular to scan direction

qmin,qmax minimum and maximum value of extracted scan axis

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

Returns

qx,qxint qx 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)
```

xrayutilities.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

qx equidistant array of qx momentum transfer

qz equidistant array of qz momentum transfer

intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxpos position at which the line scan should be extracted

**kwargs: possible keyword arguments:

qrange integration range perpendicular to scan direction

qmin,qmax minimum and maximum value of extracted scan axis

Returns

qz,qzint qz scan coordinates and intensities

Example

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

xrayutilities.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

qx equidistant array of qx momentum transfer

qz equidistant array of qz momentum transfer

intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

qxpos position at which the line scan should be extracted

**kwargs: possible keyword arguments:

angrange integration range in angular direction

> **qmin,qmax**minimum and maximum value of extracted scan axis

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

> **intdir**integration direction 'omega': sample rocking angle (default) '2theta': scattering angle

Returns

> **qz,qzint**qz 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)
```

xrayutilities.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

> **qx**equidistant array of qx momentum transfer

> **qz**equidistant array of qz momentum transfer

> **intensity**2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

> **omcenter**om-position at which the radial scan should be extracted

> **ttcenter**tt-position at which the radial scan should be extracted

> **ttrange**two theta range of the radial scan to extract

> **npoints**number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange**integration range perpendicular to scan direction

> **Nint**number of subscans used for the integration (optionally)

> **lam**wavelength for use in the conversion to angular coordinates

> **relative**determines if absolute or relative two theta positions are returned (default=True)

> **bounds**flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,tt,radint**omega,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)
```

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

return reciprocal space boundaries of radial scan

Parameters

> **omcenter**om-position at which the radial scan should be extracted

> **ttcenter**tt-position at which the radial scan should be extracted

> **ttrange** two theta range of the radial scan to extract
>
> **npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange** integration range perpendicular to scan direction
>
> **lam** wavelength for use in the conversion to angular coordinates

Returns

> **qxrad,qzrad** reciprocal space boundaries of radial scan

Example

>>>

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant array of qz momentum transfer
>
> **intensity** 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)
>
> **qxcenter** qx-position at which the radial scan should be extracted
>
> **qzcenter** qz-position at which the radial scan should be extracted
>
> **ttrange** two theta range of the radial scan to extract
>
> **npoints** number of points of the radial scan

**kwargs: possible keyword arguments:

> **omrange** integration range perpendicular to scan direction
>
> **Nint** number of subscans used for the integration (optionally)
>
> **lam** wavelength for use in the conversion to angular coordinates
>
> **relative** determines if absolute or relative two theta positions are returned (default=True)
>
> **bounds** flag to specify if the scan bounds should be returned (default: False)

Returns

> **om,tt,radint** omega,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)

xrayutilities.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

> **qx** equidistant array of qx momentum transfer
>
> **qz** equidistant 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)
```

xrayutilities.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

```
>>>
```

xrayutilities.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

        qx equidistant array of qx momentum transfer

        qz equidistant array of qz momentum transfer

        intensity 2D array of gridded reciprocal space intensity with shape (qx.size,qz.size)

        qxcenter qx-position at which the 2theta scan should be extracted

        qzcenter qz-position at which the 2theta scan should be extracted

        ttrange two theta range of the scan to extract

        npoints number of points of the radial scan

**kwargs: possible keyword arguments:

        omrange integration range in omega direction

        Nint number of subscans used for the integration (optionally)

        lam wavelength for use in the conversion to angular coordinates

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

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

Returns

        tt,ttint 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

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

xrayutilities.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,y coordinates of the point of interest (float)

        xgrid,ygrid grid coordinates in x and y direction (array)

Returns

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

xrayutilities.analysis.line_cuts3d module

xrayutilities.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

        gridder 3d xrayutilities.Gridder3D object containing the data

        qypos,qzpos position at which the line scan should be extracted

**kwargs: possible keyword arguments:

        qrange integration range perpendicular to scan direction

        qmin,qmax minimum and maximum value of extracted scan axis

Returns

> **qx,qxint** qx scan coordinates and intensities

Example

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

xrayutilities.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

> **gridder** 3d xrayutilities.Gridder3D object containing the data
>
> **qxpos,qzpos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qy,qyint** qy scan coordinates and intensities

Example

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

xrayutilities.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

> **gridder** 3d xrayutilities.Gridder3D object containing the data
>
> **qxpos,qypos** position at which the line scan should be extracted

**kwargs: possible keyword arguments:

> **qrange** integration range perpendicular to scan direction
>
> **qmin,qmax** minimum and maximum value of extracted scan axis

Returns

> **qz,qzint** qz scan coordinates and intensities

Example

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

xrayutilities.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,z** coordinates of the point of interest (float)
>
> **xgrid,ygrid,zgrid** grid coordinates in x,y,z direction (array)

Returns

> **ix,iy,iz** index of the closest gridpoint (lower left) of the point (x,y,z)

xrayutilities.analysis.misc module   miscellaneous functions helpful in the analysis and experiment

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

> Parameter

>> **peak** array which gives hkl for the peak of interest

>> **sur** hkl of the surface

>> **inp** inplane 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])

xrayutilities.analysis.sample_align module   functions to help with experimental alignment during experiments, especially for experiments with linear detectors

xrayutilities.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*, *plotlog=False*, *debug=False*)
> 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

>> **angle1** outer detector arm angle

>> **angle2** inner detector arm angle

>> **ccdimages** images of the ccd taken at the angles given above

>> **detaxis** detector arm rotation axis :default: ['z+','y-']

>> **r_i** primary beam direction [xyz][+-] default 'x+'

>> **Keyword_arguments**

>>> **plot** flag to determine if results and intermediate results should be plotted :default: True

>>> **cut_off** cut off intensity to decide if image is used for the determination or not :default: 0.7 = 70%

>>> **start** sequence 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.

>>> **fix** fix parameters of start (default: (False,False,False,False))

>>> **fig** matplotlib figure used for plotting the error :default: None (creates own figure)

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

>>> **plotlog** flag to specify if the created error plot should be on log-scale

debugflag to specify that you want to see verbose output and saving of images to show if the CEN determination works

xrayutilities.analysis.sample_align.**area_detector_calib_hkl**(*sampleang, angle1, angle2, ccdimages, hkls, experiment, material, detaxis, r_i, plot=True, cut_off=0.1, start=(0, 0, 0, 0, 0, 0, 'config'), fix=(False, False, False, False, False, False, False), fig=None, plotlog=False, debug=False*)

function to calibrate the detector parameters of an area detector it determines the detector tilt possible rotations and offsets in the detector arm angles

in this variant not only scans through the primary beam but also scans at a set of symmetric reflections can be used for the detector parameter determination. for this not only the detector parameters but in addition the sample orientation and wavelength need to be fit. Both images from the primary beam hkl = (0,0,0) and from a symmetric reflection hkl = (h,k,l) need to be given for a successful run.

parameters

sampleangsample rocking angle (needed to align the reflections (same rotation direction as inner detector rotation)) other sample angle are not allowed to be changed during the scans

angle1outer detector arm angle

angle2inner detector arm angle

ccdimagesimages of the ccd taken at the angles given above

hklsarray/list of hkl values for every image

experimentExperiment class object needed to get the UB matrix for the hkl peak treatment

materialmaterial used as reference crystal

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, stilt,stazimuth, wavelength. By default (0,0,0,0,0,0,'config') is used.

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

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

plotlogflag to specify if the created error plot should be on log-scale

debug flag to tell if you want to see debug output of the script (switch this to true only if you can handle it :))

xrayutilities.analysis.sample_align.**fit_bragg_peak**(*om*, *tt*, *psd*, *omalign*, *ttalign*, *exphxrd*, *frange=(0.03, 0.03)*, *peaktype='Gauss'*, *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 (xrayutilities.math.Gauss2d) or Lorentzian (xrayutilities.math.Lorentz2d)

PLEASE ALWAYS CHECK THE RESULT CAREFULLY!

Parameter

> **om,tt** angular coordinates of the measurement (numpy.ndarray) either with size of psd or of psd.shape[0]
>
> **psd** intensity values needed for fitting
>
> **omalign** aligned omega value, used as first guess in the fit
>
> **ttalign** aligned 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
>
> **exphxrd** experiment class used for the conversion between angular and reciprocal space.
>
> **frange** data range used for the fit in both directions (see above for details default:(0.03,0.03) unit: AA^{-1})
>
> **peaktype** can be 'Gauss' or 'Lorentz' to fit either of the two peak shapes
>
> **plot** if 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/Lorentzian) as well as their errors

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

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

parameters

> **angle** array of angles in degree of measured detector spectra
>
> **mca_spectra** corresponding detector spectra :(shape: (len(angle),Nchannels)

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

> **r_i** primary beam direction as vector [xyz][+-]; default: 'y+'
>
> **detaxis** detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

selected options from psd_chdeg:

> **plot** flag to specify if a visualization of the fit should be done
>
> **usetilt** whether to use model considering a detector tilt (deviation angle of the pixel direction from orthogonal to the primary beam) (default: True)

Note: Note: see help of psd_chdeg for more options

returns

pixelwidth (at one meter distance) , center_channel[, detector_tilt]

Note: Note: L/pixelwidth*pi/180 ~= channel/degree, with the sample detector distance L

---

pixelwidth is negative in case the hit channel number decreases upon an increase of the detector angle The function also prints out how a linear detector can be initialized using the results obtained from this calibration. Carefully check the results

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

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.

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

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

xrayutilities.analysis.sample_align.**psd_chdeg**(*angles*, *channels*, *stdev=None*, *usetilt=True*, *plot=True*, *datap='kx'*, *modelline='r–'*, *modeltilt='b-'*, *fignum=None*, *mlabel='fit'*, *mtilt-label='fit w/tilt'*, *dlabel='data'*, *figtitle=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: True)

> **datap** plot format of data points
>
> **modelline** plot format of modelline
>
> **modeltilt** plot format of modeltilt
>
> **fignum** figure number to use for the plot
>
> **mlabel** label of the model w/o tilt to be used in the plot
>
> **mtiltlabel** label of the model with tilt to be used in the plot
>
> **dlabel** label of the data line to be used in the plot
>
> **figtitle** boolean to tell if the figure title should show the fit parameters

Returns

> (pixelwidth,centerch,tilt)
>
> **Pixelwidth** the width of one detector channel @ 1m distance, which is negative in case the hit channel number decreases upon an increase of the detector angle.
>
> **Centerch** center channel of the detector
>
> **Tilt** tilt of the detector from perpendicular to the beam (will be zero in case of usetilt=False)

Note: Note: L/pixelwidth*pi/180 = channel/degree for large detector distance with the sample detector disctance L

---

xrayutilities.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

> **primarybeam** primary beam channel number
>
> **angles** list or numpy.array with angles
>
> **channels** list or numpy.array with corresponding detector channels
>
> **plot** flag to specify if a visualization of the fit is wanted :default: True

Returns

> **omega** angle 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])
```

Module contents xrayutilities.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.

xrayutilities.io package

Submodules

xrayutilities.io.cbf module

class `xrayutilities.io.cbf.`**`CBFDirectory`**(*datapath*, *ext='cbf'*, ***keyargs*)

> Bases: `object`

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

> **`Save2HDF5`**(*h5*, *group=''*, *comp=True*)

>> Saves the data stored in the CBF 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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class `xrayutilities.io.cbf.`**`CBFFile`**(*fname*, *nxkey='X-Binary-Size-Fastest-Dimension'*, *nykey='X-Binary-Size-Second-Dimension'*, *dtkey='DataType'*, *path=None*)

> Bases: `object`

> **`ReadData`**()

>> Read the CCD data into the .data object this function is called by the initialization

> **`Save2HDF5`**(*h5*, *group='/'*, *comp=True*)

>> 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 (default to the root of the file) :comp: activate compression - true by default

xrayutilities.io.desy_tty08 module   class for reading data+header information from tty08 data files

tty08 is system used at beamline P08 at Hasylab Hamburg and creates simple ASCII files to save the data. Information is easily read from the multicolumn data file. the functions below enable also to parse the information of the header

`xrayutilities.io.desy_tty08.`**`gettty08_scan`**(*scanname*, *scannumbers*, **args*)

> function to obtain the angular cooridinates as well as intensity values saved in TTY08 datafiles. Especially usefull for reciprocal space map measurements, and to combine date from several scans

> 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

>> **scanname** name of the scans, for multiple scans this needs to be a template string

>> **scannumber** number 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)

> 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.gettty08_scan('text%05d.dat',36,'omega','gamma')
```

class `xrayutilities.io.desy_tty08.`**`tty08File`**(*filename*, *path=None*, *mcadir=None*)

Bases: `object`

Represents a tty08 data file. The file is read during the Constructor call. This class should work for data stored at beamline P08 using the tty08 acquisition system.

Required constructor arguments:

> **filename** a string with the name of the tty08-file

Optional keyword arguments:

> **mcadir** directory name of MCA files

`Read`()

> Read the data from the file

`ReadMCA`()

xrayutilities.io.edf module

class `xrayutilities.io.edf.`**`EDFDirectory`**(*datapath*, *ext='edf'*, ***keyargs*)

Bases: `object`

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

`Save2HDF5`(*h5*, *group=''*, *comp=True*)

> 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 (defaults to pathname if group is empty string) :comp: activate compression - true by default

class `xrayutilities.io.edf.`**`EDFFile`**(*fname*, *nxkey='Dim_1'*, *nykey='Dim_2'*, *dtkey='DataType'*, *path=''*, *header=True*)

Bases: `object`

`ReadData`()

> Read the CCD data into the .data object this function is called by the initialization

`Save2HDF5`(*h5*, *group='/'*, *comp=True*)

> 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 (default to the root of the file) :comp: activate compression - true by default

xrayutilities.io.helper module convenience functions to open files for various data file reader

these functions should be used in new parsers since they transparently allow to open gzipped and bzipped files

`xrayutilities.io.helper.`**`xu_open`**(*filename*, *mode='rb'*)

> function to open a file no matter if zipped or not. Files with extension '.gz' or '.bz2' are assumed to be compressed and transparently opened to read like usual files.
>
> Parameters
>
> > **filename** filename of the file to open (full including path)
> >
> > **mode** mode in which the file should be opened

Returns

> file handle of the opened file

If the file does not exist an IOError is raised by the open routine, which is not caught within the function

xrayutilities.io.imagereader module

class xrayutilities.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

> > **filename** filename 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 xrayutilities.io.imagereader.**PerkinElmer**(***keyargs*)
> Bases: `xrayutilities.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 xrayutilities.io.imagereader.**RoperCCD**(***keyargs*)
> Bases: `xrayutilities.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.

xrayutilities.io.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 xrayutilities.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 xrayutilities.io.panalytical_xml.**XRDMLMeasurement**(*measurement*)
> Bases: `object`

class to handle scans in a XRDML datafile

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

xrayutilities.io.panalytical_xml.**getxrdml_map**(*filetemplate*, *scannrs=None*, *path='.'*,
roi=None)

parses multiple XRDML file and concatenates the results for parsing the xrayutilities.io.XRDMLFile class is used. The function can be used for parsing maps measured with the PIXCel and point detector.

Parameter

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

> **scannrs**int or list of scan numbers

> **path**common path to the filenames

> **roi**region of interest for the PIXCel detector, for other measurements this i not usefull!

Returns

> **om,tt,psd**as flattened numpy arrays

Example

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

xrayutilities.io.panalytical_xml.**getxrdml_scan**(*filetemplate*, **motors*, ***kwargs*)

parses multiple XRDML file and concatenates the results for parsing the xrayutilities.io.XRDMLFile class is used. The function can be used for parsing arbitrary scans and will return the the motor values of the scan motor and additionally the positions of the motors given by in the "*motors" argument

Parameter

> **filetemplate**template string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames given by the scannrs keyword argument

> ***motors: motor names to return: e.g.: 'Omega','2Theta',...**one can also use abbreviations 'Omega' = 'om' = 'o' '2Theta' = 'tt' = 't' 'Chi' = 'c' 'Phi' = 'p'

> ****kwargs:**

>> **scannrs**int or list of scan numbers

>> **path**common path to the filenames

Returns

> scanmot,mot1,mot2,...,detectorint: as flattened numpy arrays

Example

```
>>> scanmot,om,tt,inte = xrayutilities.io.getxrdml_scan("samplename_1.xrdml",'om','tt',path="
```

xrayutilities.io.radicon module python module for converting radicon data to HDF5

xrayutilities.io.radicon.**hst2hdf5**(*h5*, *hstfile*, *nofchannels*, *h5path='/'*, *hstpath='.'*)

Converts a HST file to an HDF5 file.

Required input arguments:

> **h5**HDF5 object where to store the data

> **hstfile**name of the HST file

> **nofchannels**number of channels

optional input arguments:

> **h5path**Path in the HDF5 file where to store the data

> **hstpath**path where the HST file is located (default is the current working directory)

`xrayutilities.io.radicon.`**`rad2hdf5`**(*h5*, *rdcfile*, *h5path='/'*, *rdcpath='.'*)
> Converts a RDC file to an HDF5 file.

> **Required input arguments:**
>> **h5**HDF5 object where to store the data

>> **rdcfile**name of the RDC file

> **optional input arguments:**
>> **h5path**Path in the HDF5 file where to store the data (default to root)

>> **rdcpath**path where the RDC file is located (default is the current working directory)

`xrayutilities.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_limit**exposure time limit

>> **mca_info**pytables table with the exposure data

>> **mca_array**array with all the MCA spectra

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

xrayutilities.io.rotanode_alignment module   parser for the alignment log file of the rotating anode

class `xrayutilities.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
>>> **pname**peakname for which the alignment should be plotted

xrayutilities.io.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 `xrayutilities.io.seifert.`**`SeifertHeader`**
> Bases: `object`

> **`save_h5_attribs`**(*obj*)

---

class `xrayutilities.io.seifert.`**`SeifertMultiScan`** (*filename*, *m_scan*, *m2*, *path=None*)

> Bases: `object`

> **`dump2hdf5`** (*h5*, *iname='INT'*, *group='/'*)
>
> > 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:**
> >
> > > **h5** a HDF5 file object
> >
> > **optional keyword arguments:**
> >
> > > **iname** name for the intensity matrix
> > >
> > > **group** path to the HDF5 group where to store the data
>
> **`dump2mlab`** (*fname*, **args*)
>
> > Store the data in a matlab file.
>
> **`parse`** ()

class `xrayutilities.io.seifert.`**`SeifertScan`** (*filename*, *path=None*)

> Bases: `object`

> **`dump2h5`** (*h5*, **args*, ***keyargs*)
>
> > Save the data stored in the Seifert ASCII file to a HDF5 file.
> >
> > **required input arguments:**
> >
> > > **h5** HDF5 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:**
> >
> > > **group** HDF5 group object where to store the data.
>
> **`dump2mlab`** (*fname*, **args*)
>
> > Save the data from a Seifert scan to a matlab file.
> >
> > **required input arugments:**
> >
> > > **fname** name 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`** ()

`xrayutilities.io.seifert.`**`getSeifert_map`** (*filetemplate*, *scannrs=None*, *path='.'*, *scant-ype='map'*, *Nchannels=1280*)

> parses multiple Seifert *.nja files and concatenates the results. for parsing the xrayutilities.io.SeifertMultiScan class is used. The function can be used for parsing maps measured with the Meteor1D and point detector.
>
> Parameter
>
> > **filetemplate** template string for the file names, can contain a %d which is replaced by the scan number or be a list of filenames
> >
> > **scannrs** int or list of scan numbers
> >
> > **path** common path to the filenames
> >
> > **scantype** type of datafile: "map": reciprocal space map measured with a regular Seifert job "tsk": MCA spectra measured using the TaskInterpreter

---

> > > **Nchannels** number of channels of the MCA (needed for "tsk" measurements)

> Returns

> > **om,tt,psd** as flattened numpy arrays

> Example

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

xrayutilities.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 _.

xrayutilities.io.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 xrayutilities.io.spec.**SPECCmdLine**(*n*, *prompt*, *cmdl*, *out*)

> Bases: `object`

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

class xrayutilities.io.spec.**SPECFile**(*filename*, *path=''*)

> 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*, *comp=True*)

> > 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:**

> > > **h5f** a HDF5 file object or its filename

> > **optional keyword arguments:**

> > > **comp** activate compression - true by default

> **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 xrayutilities.io.spec.**SPECLog**(*filename*, *prompt*, *path=''*)

> Bases: `object`

> **Parse**()

> **Update**()

class xrayutilities.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 xrayutilities.io.spec.**SPECMCAFile**

> Bases: `xrayutilities.io.spec.SPECMCA`

ReadData()

class xrayutilities.io.spec.**SPECMCAInline**

Bases: [xrayutilities.io.spec.SPECMCA](#)

ReadData()

class xrayutilities.io.spec.**SPECScan**(*name*, *scannr*, *command*, *date*, *time*, *itime*, *col-names*, *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*, *group='/'*, *title=''*, *desc=''*, *optattrs={}*, *comp=True*)

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:

> **h5f** a HDF5 file object or its filename

optional keyword arguments:

> **group** name or group object of the HDF5 group where to store the data
>
> **title** a string with the title for the data, defaults to the name of scan if empty
>
> **desc** a string with the description of the data, defaults to the scan command if empty
>
> **optattrs** a dictionary with optional attributes to store for the data
>
> **comp** activate 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_format** number of columns used to save the data
>
> **mca_channels** number of MCA channels stored
>
> **mca_start** first channel that is stored
>
> **mca_stop** last 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.

Parameters

> ***args: arguments for the plot: first argument is the name of x-value column** the following pairs of arguments are the y-value names and plot styles allowed are 3,5,7,... number of arguments
>
> ****keyargs:**
>
> > **newfig** if True a new figure instance will be created otherwise an existing one will be used

> **logy** if True a semilogy plot will be done

`xrayutilities.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
>
>> **h5f** file object of a HDF5 file opened using pytables or its filename
>>
>> **scans** number 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):**
>>
>>> **samplename** string 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')
>  ```

xrayutilities.io.spectra module   module to handle spectra data

class `xrayutilities.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:
>
>> **filename** a string with the name of the SPECTRA file
>
> Optional keyword arguments:
>
>> **mcatmp** template for the MCA files
>>
>> **mcastart,mcastop** start 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:**

> **h5file**string or HDF5 file object
>
> **name**name of the group where to store the data

> **optional keyword arguments:**
>
> > **group**root group where to store the data
> >
> > **description**string with a description of the scan

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

class `xrayutilities.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 `xrayutilities.io.spectra.`**`SPECTRAFileData`**
> Bases: `object`
>
> **append**(*col*)

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

class `xrayutilities.io.spectra.`**`SPECTRAFileParameters`**
> Bases: `dict`

class `xrayutilities.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:**
> > > **mca**matrix with the MCA data
> > >
> > > **f**filter values along the scan
> >
> > **optional keyword arguments:**
> > > **ff**custome filter factors
> >
> > **return value:**Array with the same shape as mca with the corrected MCA data.
>
> **recarray2hdf5**(*h5g*, *rec*, *name*, *desc*)
> > Save a record array in an HDF5 file. A pytables table object is used to store the data.
> >
> > **required input arguments:**
> > > **h5g**HDF5 group object or path
> > >
> > > **rec**record array
> > >
> > > **name**name of the table in the file
> > >
> > > **desc**description of the table in the file
> >
> > **return value:**
> > > **tab**a HDF5 table object
>
> **set_abs_factors**(*ff*)
> > Set the global absorber factors in the module.

spectra2hdf5 (*dir*, *fname*, *mcatemp*, *name=''*, *desc='SPECTRA data'*)
> Convert SPECTRA scan data to a HDF5 format.

> **required input arguments:**

>> **dir** directory where the scan is stored

>> **fname** name of the SPECTRA data file

>> **mcatemp** template for the MCA file names

> **optional keyword arguments:**

>> **name** optional name under which to save the data if empty the basename of the filename will be used

>> **desc** optional description of the scan

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

> **required input arguments:**

>> **dirname** name of the directory to search

> **return values:** list with filenames

xrayutilities.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

>> **h5f** file object of a HDF5 file opened using pytables

>> **scans** number of the scans of the reciprocal space map (int,tuple or list)

> ***args: arbitrary number of motor names (strings)**

>> **omname** name of the omega motor (or its equivalent)

>> **ttname** name of the two theta motor (or its equivalent)

> ****kwargs (optional):**

>> **mca** name of the mca data (if available) otherwise None (default: "MCA")

>> **samplename** string 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']).

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

> **required input arguments:**

>> **fname** name of the file to read

> **return values: (data,hdr)**

>> **data** numpy record array where the keys are the column names

>> **hdr** a dictionary with header information

`xrayutilities.io.spectra.`**`read_mca`**(*fname*)

> Read a single SPECTRA MCA file.

> **required input arguments:**

>> **fname**name of the file to read

> **return value:**

>> **data**a numpy array witht the MCA data

`xrayutilities.io.spectra.`**`read_mca_dir`**(*dirname*, *filetemp*, *sort=True*)

> Read all MCA files within a directory

`xrayutilities.io.spectra.`**`read_mcas`**(*ftemp*, *cntstart*, *cntstop*)

> Read MCA data from a SPECTRA MCA directory. The filename is passed as a generic

Module contents

xrayutilities.materials package

Submodules

xrayutilities.materials.cif module

class `xrayutilities.materials.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.

xrayutilities.materials.database module   module to handle access to the optical parameters database

class `xrayutilities.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:**

>>> **dbname**name of the database

>>> **dbdesc**a 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:**

>>> **name**a string with the name of the material

> > > **description** a string with a description of the material

> **GetF0** (*q*)
> > Obtain the f0 scattering factor component for a particular momentum transfer q.
> >
> > **required input argument:**
> >
> > > **q** single 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:**
> >
> > > **en** float or numpy array with the energy

> **GetF2** (*en*)
> > Return the imaginary part of the scattering factor for a certain energy en.
> >
> > **required input arguments:**
> >
> > > **en** float 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:**
> >
> > > **parameters** list or numpy array with the fit parameters

> **SetF1** (*en, f1*)
> > Set f1 tabels values for the active material.
> >
> > **required input arguments:**
> >
> > > **en** list or numpy array with energy in (eV)
> > >
> > > **f1** list or numpy array with f1 values

> **SetF2** (*en, f2*)
> > Set f2 tabels values for the active material.
> >
> > **required input arguments:**
> >
> > > **en** list or numpy array with energy in (eV)
> > >
> > > **f2** list 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:**
> >
> > > **name** string with the name of the material

> **SetWeight** (*weight*)
> > Save weight of the element as float
> >
> > **required input argument:**
> >
> > > **weight** atomic standard weight of the element (float)

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

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

---

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

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

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

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

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

xrayutilities.materials.elements module

xrayutilities.materials.lattice module   module handling crystal lattice structures

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

> **f**(*q*, *en='config'*)
> > function to calculate the atomic structure factor F
>
> > Parameter
> >
> > > **q** momentum transfer
> > >
> > > **en** energy for which F should be calculated, if omitted the value from the xrayutilities configuration is used
> >
> > Returns
> >
> > > f (float)

> **f0**(*q*)

> **f1**(*en='config'*)

> **f2**(*en='config'*)

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

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

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

xrayutilities.materials.lattice.**CsClLattice**(*aa*, *ab*, *a*)

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

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

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

> **required input arguments:**
> > **a** lattice parameter

> **return value:** an instance of Lattice class

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

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

xrayutilities.materials.lattice.**GeTeRhombohedral**(*aa*, *ab*, *a*, *ang*, *x=0.237*)

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

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

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

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

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

class xrayutilities.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:**

> > > **eps** a 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 xrayutilities.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

> > > **atom** atom object to be added

> > > **pos** position of the atom

> > > **occ** occupancy (default=1.0)

> > > **b** b-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)

xrayutilities.materials.lattice.**MagnetiteLattice**(*aa*, *ab*, *ac*, *a*, *x=0.255*)

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

xrayutilities.materials.lattice.**NiAsLattice**(*aa*, *ab*, *a*, *c*, *biso=0.0*)

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

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

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

`xrayutilities.materials.lattice.`**`RockSalt_Cubic_Lattice`**(*aa*, *ab*, *a*)

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

`xrayutilities.materials.lattice.`**`TetragonalIndiumLattice`**(*aa*, *a*, *c*)

`xrayutilities.materials.lattice.`**`TetragonalTinLattice`**(*aa*, *a*, *c*)

`xrayutilities.materials.lattice.`**`TrigonalR3mh`**(*aa*, *a*, *c*)

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

`xrayutilities.materials.lattice.`**`ZincBlendeLattice`**(*aa*, *ab*, *a*)

xrayutilities.materials.material module class module implements a certain material

class `xrayutilities.materials.material.`**`Alloy`**(*matA*, *matB*, *x*)

> Bases: `xrayutilities.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
>>
>>> **hkl** Miller Indices
>>>
>>> **sub** substrate material or lattice constant (Instance of Material class or float)
>>>
>>> **exp** Experiment class from which the Transformation object and ndir are needed
>>
>> Returns
>>
>>> **qy,qz** reciprocal space coordinates of the corners of the relaxation triangle
>
> **`lattice_const_AB`**(*latA*, *latB*, *x*)
>
>> method to set the composition of the Alloy. x is the atomic fraction of the component B
>
> **`x`**

`xrayutilities.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

`xrayutilities.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 `xrayutilities.materials.material.`**`CubicAlloy`**(*matA*, *matB*, *x*)

> Bases: `xrayutilities.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_inp** inplane peak position of reflection hkl of the alloy in reciprocal space

> > **q_perp** perpendicular peak position of the reflection hkl of the alloy in reciprocal space
>
> > **hkl** Miller indices of the measured asymmetric reflection
>
> > **sur** Miller 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_perp** perpendicular peak position of the reflection hkl of the alloy in reciprocal space
>
> > **hkl** Miller indices of the measured symmetric reflection (also defines the surface normal
>
> > **inpr** Miller indices of a Bragg peak defining the inplane reference direction
>
> > **asub** substrate lattice constant
>
> > **relax** degree of relaxation (needed to obtain the content from symmetric reciprocal space position)
>
> Returns
>
> > **content** the content of B in the alloy determined from the input variables

xrayutilities.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,c44** independent components of the elastic tensor of cubic materials
>
> Returns
>
> > 6x6 materix with elastic constants

xrayutilities.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,c** unit cell extenstions (Angstrom)
>
> > **alpha** angle between unit cell vectors b,c
>
> > **beta** angle between unit cell vectors a,c
>
> > **gamma** angle between unit cell vectors a,b
>
> returns a Material object with the specified properties

xrayutilities.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,c44 independent components of the elastic tensor of a hexagonal material

Returns

    6x6 materix with elastic constants

class `xrayutilities.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:

            hkl list 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

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

            en energy in eV, if omitted the value from the xrayutilities configuration is used

            temp temperature 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

            q0 vectorial momentum transfer (vectors as list,tuple or numpy array are valid)

            en list, tuple or array of energy values in eV

            temp temperature 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

            q vectorial 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)])

            en0 energy value in eV, if omitted the value from the xrayutilities configuration is used

            temp temperature 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
>
>> **en** x-ray energy eV, if omitted the value from the xrayutilities 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
>
>> **q** momentum transfer in (1/A)
>>
>> **en** xray energy in eV, if omitted the value from the xrayutilities configuration is used
>>
>> **temp** temperature used for Debye-Waller-factor calculation
>>
>> **polarization** either '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
>
>> **en** energy of the x-rays, if omitted the value from the xrayutilities configuration is used
>>
>> **deg** return 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
>
>> **Q** momentum transfer (1/A)
>>
>> **en** x-ray energy (eV), if omitted the value from the xrayutilities configuration is used
>
> Returns
>
>> **deltaTheta** peak 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

> **en** x-ray energy eV, if omitted the value from the xrayutilities configuration is used

> Returns

>> delta (float)

distances()
> function to obtain distances of atoms in the crystal up to the unit cell size (largest value of a,b,c is the cut-off)

> returns a list of tuples with distance d and number of occurence n [(d1,n1),(d2,n2),...]

Note: Note: if the base of the material is empty the list will be empty

---

idx_refraction(*en='config'*)
> function to calculate the complex index of refraction of a material in the x-ray range

> Parameter

>> **en** energy of the x-rays, if omitted the value from the xrayutilities configuration is used

> Returns

>> n (complex)

lam

mu

nu

xrayutilities.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 on (001) substrates

> **required input arguments:**

>> **submat** substrate material

>> **layermat** bulk material of the layer

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

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

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

xrayutilities.materials.predefined_materials module

class xrayutilities.materials.predefined_materials.**SiGe**(*x*)
> Bases: xrayutilities.materials.material.CubicAlloy

> **lattice_const_AB**(*latA*, *latB*, *x*)
>> method to calculate the lattice parameter of the SiGe alloy with composition Si_{1-x}Ge_x

> **x**

Module contents

xrayutilities.math package

Submodules

xrayutilities.math.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

xrayutilities.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,y** data coordinates (do NOT need to be regularly spaced)

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

  > **start** set of starting parameters for the fit used as first parameter of function fit_function

  > **drange** limits 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_function** function which should be fitted must accept the parameters (x,y,*params)

  Returns

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

xrayutilities.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

  > **xdata** xcoordinates of the data to be fitted

  > **ydata** ycoordinates of the data which should be fit

  keyword parameters:

  > **iparams** initial paramters for the fit (determined automatically if nothing is given

  > **maxit** maximal 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

xrayutilities.math.fit.**multGaussFit**(**args*, ***kwargs*)
  convenience function to keep API stable see multPeakFit for documentation

xrayutilities.math.fit.**multGaussPlot**(**args*, ***kwargs*)
  convenience function to keep API stable see multPeakPlot for documentation

xrayutilities.math.fit.**multPeakFit**(*x*, *data*, *peakpos*, *peakwidth*, *dranges=None*, *peaktype='Gaussian'*)
  function to fit multiple Gaussian/Lorentzian peaks with linear background to a set of data

  Parameters

  > **x** x-coordinate of the data

  > **data** data array with same length as x

  > **peakpos** initial parameters for the peak positions

  > **peakwidth** initial values for the peak width

  > **dranges** list of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as peakpos

  > **peaktype** type of peaks to be used: can be either 'Gaussian' or 'Lorentzian'

  Returns

pos,sigma,amp,background

Pos list of peak positions derived by the fit

Sigma list of peak width derived by the fit

Amp list of amplitudes of the peaks derived by the fit

Background array of background values at positions x

xrayutilities.math.fit.**multPeakPlot**(*x*, *fpos*, *fwidth*, *famp*, *background*, *dranges=None*, *peaktype='Gaussian'*, *fig='xu_plot'*, *fact=1.0*)

function to plot multiple Gaussian/Lorentz peaks with background values given by an array

Parameters

x x-coordinate of the data

fpos list of positions of the peaks

fwidth list of width of the peaks

famp list of amplitudes of the peaks

background array with background values

dranges list of tuples with (min,max) value of the data ranges to use. does not need to have the same number of entries as fpos

peaktype type of peaks to be used: can be either 'Gaussian' or 'Lorentzian'

fig matplotlib figure number or name

fact factor to use as multiplicator in the plot

xrayutilities.math.fit.**peak_fit**(*xdata*, *ydata*, *iparams=[ ]*, *peaktype='Gauss'*, *maxit=200*)

fit function using odr-pack wrapper in scipy similar to :https: //github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting for Gauss, Lorentz or Pseudovoigt-functions

Parameters

xdata xcoordinates of the data to be fitted

ydata ycoordinates of the data which should be fit

keyword parameters:

iparams initial paramters for the fit (determined automatically if nothing is given)

peaktype type of peak to fit ('Gauss','Lorentz','PseudoVoigt')

maxit maximal iteration number of the fit

Returns

params,sd_params,itlim

the parameters as defined in function Gauss1d/Lorentz1d or PseudoVoigt1d(x, *param) and their errors of the fit, as well as a boolean flag which is false in the case of a successful fit

xrayutilities.math.functions module module with several common function needed in xray data analysis

xrayutilities.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

>> **x** argument of the Debye function (float)

> Returns

>> **D1(x)** float value of the Debye function

xrayutilities.math.functions.**Gauss1d**(*x*, **p*)

> function to calculate a general one dimensional Gaussian

> Parameters

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

>> **x** coordinate(s) where the function should be evaluated

> Returns

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

> Example

> Calling with a list of parameters needs a call looking as shown below (note the '*') or explicit listing of the parameters: >>> Gauss1d(x,*p) >>> Gauss1d(numpy.linspace(0,10,100), 5, 1, 1e3, 0)

xrayutilities.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

xrayutilities.math.functions.**Gauss1d_der_x**(*x*, **p*)

> function to calculate the derivative of a Gaussian with respect to x

> for parameter description see Gauss1d

xrayutilities.math.functions.**Gauss2d**(*x*, *y*, **p*)

> function to calculate a general two dimensional Gaussian

> Parameters

>> **p** list 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 in radians

>> **x,y** coordinate(s) where the function should be evaluated

> Returns

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

xrayutilities.math.functions.**Gauss3d**(*x*, *y*, *z*, **p*)

> function to calculate a general three dimensional Gaussian

> Parameters

>> **p** list of parameters of the Gauss-function [XCEN,YCEN,ZCEN,SIGMAX,SIGMAY,SIGMAZ,AMP,BACKGROUN SIGMA = FWHM / (2*sqrt(2*log(2)))

>> **x,y,z** coordinate(s) where the function should be evaluated

> Returns

> the value of the Gaussian described by the parameters p at positions (x,y,z)

xrayutilities.math.functions.**Lorentz1d**(*x*, **p*)

> function to calculate a general one dimensional Lorentzian

> Parameters

>> **p** list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND]

> xcoordinate(s) where the function should be evaluated

Returns

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

xrayutilities.math.functions.**Lorentz1d_der_p**(*x*, **p*)

> function to calculate the derivative of a Gaussian with respect the parameters p

for parameter description see Lorentz1d

xrayutilities.math.functions.**Lorentz1d_der_x**(*x*, **p*)

> function to calculate the derivative of a Gaussian with respect to x

for parameter description see Lorentz1d

xrayutilities.math.functions.**Lorentz2d**(*x*, *y*, **p*)

> function to calculate a general two dimensional Lorentzian

Parameters

> **p**list of parameters of the Lorentz-function [XCEN,YCEN,FWHMX,FWHMY,AMP,BACKGROUND,ANGLE]
> ANGLE = rotation of the X,Y direction of the Lorentzian in radians

> **x,y**coordinate(s) where the function should be evaluated

Returns

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

xrayutilities.math.functions.**PseudoVoigt1d**(*x*, **p*)

> function to calculate a pseudo Voigt function as linear combination of a Gauss and Lorentz peak

Parameters

> **p**list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
> :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

> **x**coordinate(s) where the function should be evaluated

Returns

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

xrayutilities.math.functions.**PseudoVoigt1dArea**(**p*)

> function to calculate the area of a pseudo Voigt function with neglected background

Parameters

> **p**list of parameters of the Lorentz-function [XCEN,FWHM,AMP,BACKGROUND,ETA]
> :ETA: 0 ...1 0 means pure Gauss and 1 means pure Lorentz

Returns

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

xrayutilities.math.functions.**TwoGauss2d**(*x*, *y*, **p*)

> function to calculate two general two dimensional Gaussians

Parameters

> **p**list 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 in radians

> **x,y**coordinate(s) where the function should be evaluated

Return

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

4.7. xrayutilities

xrayutilities.math.functions.**kill_spike**(*data*, *threshold=2.0*)
>   function to smooth **single** data points which differ from the average of the neighboring data points by more than the threshold factor. Such spikes will be replaced by the mean value of the next neighbors.

> | **Warning:** Use this function carefully not to manipulate your data! |
> | --- |

>   Parameters
>
>>   **data** 1d numpy array with experimental data
>>
>>   **threshold** threshold factor to identify strange data points
>
>   Returns
>
>>   1d data-array with spikes removed

xrayutilities.math.functions.**smooth**(*x*, *n*)
>   function to smooth an array of data by averaging N adjacent data points

>   Parameters
>
>>   **x** 1D data array
>>
>>   **n** number of data points to average
>
>   Returns
>
>>   **xsmooth** smoothed array with same length as x

xrayutilities.math.transforms module

class xrayutilities.math.transforms.**AxisToZ**(*newzaxis*)
>   Bases: `xrayutilities.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.

xrayutilities.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

xrayutilities.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 xrayutilities.math.transforms.**CoordinateTransform**(*v1*, *v2*, *v3*)
>   Bases: `xrayutilities.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 xrayutilities.math.transforms.**Transform**(*matrix*)
>   Bases: `object`

inverse(**args*)

    performs inverse transformation a vector, matrix or tensor of rank 4

    Parameters

        *args: object to transform, list or numpy array of shape(n,) (n,n), (n,n,n,n) where n is the rank of the transformation matrix

xrayutilities.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.

xrayutilities.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.

xrayutilities.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.

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

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

xrayutilities.math.transforms.**mycross**(*vec*, *mat*)

    function implements the cross-product of a vector with each column of a matrix

xrayutilities.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])
```

xrayutilities.math.transforms.**tensorprod**(*vec1*, *vec2*)

    function implements an elementwise multiplication of two vectors

xrayutilities.math.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.

xrayutilities.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

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

> **required input arguments:**

> > **v1**vector as numpy array or list

> > **v2**vector as numpy array or list

> **return value:**float value

xrayutilities.math.vector.**VecNorm**(*v*)
> Calculate the norm of a vector.

> **required input arguments:**

> > **v**vector as list or numpy array

> **return value:**float holding the vector norm

xrayutilities.math.vector.**VecUnit**(*v*)
> Calculate the unit vector of v.

> **required input arguments:**

> > **v**vector as list or numpy array

> **return value:**numpy array with the unit vector

xrayutilities.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

xrayutilities.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

Module contents

Submodules

xrayutilities.config module

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

xrayutilities.exception module

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

exception `xrayutilities.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)

> **Attibutes**expr – input expression in which the error occurred :msg: – explanation of the error

xrayutilities.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 `xrayutilities.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**

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

>>> **mat**Material 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

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

>>> **dettype**detector 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

>>> **q**list or numpy array with the reciprocal space position

>> **optional keyword arguments:**

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

> **Transform**(*v*)
>> transforms a vector, matrix or tensor of rank 4 (e.g. elasticity tensor) to the coordinate frame of the Experiment class.
>>
>> Parameters
>>
>>> **v** object 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 xrayutilities.experiment.**GID**(*idir*, *ndir*, ***keyargs*)

> Bases: `xrayutilities.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,beta** sample 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**
>>
>>> **delta** giving 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
>>>
>>> **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)
>>>
>>> **wl** x-ray wavelength in angstroem (default: self._wl)
>>>
>>> **deg** flag 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
>>
>>> **Q** a list or numpy array of shape (3) with q-space vector components
>>
>> **optional keyword arguments:**
>>
>>> **trans** True/False apply coordinate transformation on Q
>>>
>>> **deg** True/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_i** incidence angle to surface (at the moment always 0)
>
> **azimuth** sample rotation with respect to the inplane reference direction
>
> **twotheta** scattering angle
>
> **beta** exit angle from surface (at the moment always 0)

class xrayutilities.experiment.**GISAXS**(*idir*, *ndir*, ***keyargs*)
    Bases: xrayutilities.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,beta** sample 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

> **delta** giving 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
>
> **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)
>
> **wl** x-ray wavelength in angstroem (default: self._wl)
>
> **deg** flag 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 xrayutilities.experiment.**HXRD**(*idir*, *ndir*, *geometry='hi_lo'*, ***keyargs*)
    Bases: xrayutilities.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,tt** sample 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

> > **delta** giving 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
> >
> > **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: identity matrix)
> >
> > **wl** x-ray wavelength in angstroem (default: self._wl)
> >
> > **deg** flag 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
>
> > **Q** a list, tuple or numpy array of shape (3) with q-space vector components or 3 separate lists with qx,qy,qz

optional keyword arguments:

> > **trans** True/False apply coordinate transformation on Q (default True)
> >
> > **deg** True/Flase (default True) determines if the angles are returned in radians or degrees
> >
> > **geometry** determines 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]
> > >
> > > **default** self.geometry
> >
> > **refrac** boolean to determine if refraction is taken into account :default: False if True then also a material must be given
> >
> > **mat** Material object; needed to obtain its optical properties for refraction correction, otherwise not used
> >
> > **full_output** boolean to determine if additional output is given to determine scattering angles more accurately in case refraction is set to True :default: False
> >
> > **fi,fd** if 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]
>
> > **omega** incidence angle with respect to surface
> >
> > **chi** sample tilt for the case of non-coplanar geometry
> >
> > **phi** sample azimuth with respect to inplane reference direction
> >
> > **twotheta** scattering angle/detector angle

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

> **psi_i** offset of the incidence beam from the scattering plane due to refraction

> **pdi_d** offset ot the diffracted beam from the scattering plane due to refraction

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

Bases: xrayutilities.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,tt** sample 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

> **delta** giving 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

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

> **wl** x-ray wavelength in angstroem (default: self._wl)

> **deg** flag 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

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

optional keyword arguments:

> **trans** True/False apply coordinate transformation on Q (default True)

> **deg** True/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]

> **omega** sample rocking angle

> **chi** sample tilt

> **phi** sample azimuth

> **twotheta** scattering angle (detector)

class xrayutilities.experiment.**Powder**(*mat*, ***keyargs*)

    Bases: xrayutilities.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 xrayutilities.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)

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

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

sampledis sample displacement vector in same units as the detector distance (default: (0,0,0))

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

detectorDir1 direction of the detector (along the pixel direction 1); e.g. 'z+' means higher pixel numbers at larger z positions

detectorDir2 direction of the detector (along the pixel direction 2); e.g. 'x+'

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

Nch1 number of detector pixels along direction 1

Nch2 number of detector pixels along direction 2

distance distance of center pixel from center of rotation

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

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

detrot angle of the detector rotation around primary beam direction (used to correct misalignments)

tiltazimuth direction of the tilt vector in the detector plane (in degree)

tilt tilt 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 !!

Note: Note: the channel numbers run from 0 .. NchX-1

**kwargs: optional keyword arguments

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

roi region 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
>
> > **detectorDir** direction of the detector (along the pixel array); e.g. 'z+'
> >
> > **cch** center channel, in direction of self.r_i at zero detectorAngles
> >
> > **Nchannel** total number of detector channels
> >
> > **distance** distance of center channel from center of rotation
> >
> > **pixelwidth** width of one pixel (same unit as distance)
> >
> > **chpdeg** channels per degree (only absolute value is relevant) sign determined through detectorDir
> >
> > > !! Either distance and pixelwidth or chpdeg must be given !!
> >
> > **tilt** tilt of the detector axis from the detectorDir (in degree)

Note: Note: the channel numbers run from 0 .. Nchannel-1

> ****kwargs: optional keyword arguments**
>
> > **Nav** number of channels to average to reduce data size (default: 1)
> >
> > **roi** region 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
> >
> > **sAngles** sample circle angles, number of arguments must correspond to len(self.sampleAxis)
> >
> > **dAngles** detector circle angles, number of arguments must correspond to len(self.detectorAxis)
>
> ****kwargs: possible keyword arguments**
>
> > **delta** giving 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
> >
> > **UB** matrix for conversion from (hkl) coordinates to Q of sample used to determine not Q but (hkl) :(default: self.UB)
> >
> > **Nav** number of channels to average to reduce data size (default: self._linear_nav)
> >
> > **roi** region of interest for the detector pixels; e.g. [100,900] (default: self._linear_roi)
> >
> > **wl** x-ray wavelength in angstroem (default: self._wl)
> >
> > **deg** flag to tell if angles are passed as degree (default: True)
> >
> > **sampledis** sample displacement vector in same units as the detector distance (default: (0,0,0))

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

> **sAngles** sample circle angles, number of arguments must correspond to len(self.sampleAxis)

> **dAngles** detector circle angles, number of arguments must correspond to len(self.detectorAxis)

**kwargs: optional keyword arguments

> **delta** giving 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

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

> **wl** x-ray wavelength in angstroem (default: self._wl)

> **deg** flag to tell if angles are passed as degree :(default: True)

> **sampledis** sample displacement vector in relative units of the detector distance (default: (0,0,0))

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

xrayutilities.gridder module

class xrayutilities.gridder.**Gridder**

Bases: object

Clear()

Clear so far gridded data to reuse this instance of the Gridder

KeepData(*bool*)

Normalize(*bool*)

set or unset the normalization flag. Normalization needs to be done to obtain proper gridding but may want to be disabled in certain cases when sequential gridding is performed

> **data**
>> return gridded data (performs normalization if switched on)

class `xrayutilities.gridder.`**`Gridder1D`**(*nx*)

> Bases: `xrayutilities.gridder.Gridder`

> **dataRange**(*min*, *max*, *fixed=True*)
>> define minimum and maximum data range, usually this is deduced from the given data automatically, however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside the range are simply ignored

>> Parameters

>>> **min** minimum value of the gridding range

>>> **max** maximum value of the gridding range

>>> **fixed** flag to turn fixed range gridding on (True (default)) or of (False)

> **xaxis**
>> Returns the xaxis of the gridder the returned values correspond to the center of the data bins used by the numpy.histogram function

`xrayutilities.gridder.`**`axis`**(*min_value*, *max_value*, *n*)

> Compute the a grid axis.

> Parameters

>> **Min_value** axis minimum value

>> **Max_value** axis maximum value

>> **N** number of steps

`xrayutilities.gridder.`**`check_array`**(*a*, *dtype*)

> Check if an array fits the requirements for the C-code and returns it back to the callee. Such arrays must be aligned and C_CONTIGUOUS which means that they have to follow C-ordering.

> Parameters

>> **A** array to check

>> **Dtype** numpy data type

`xrayutilities.gridder.`**`delta`**(*min_value*, *max_value*, *n*)

> Compute the stepsize along an axis of a grid.

> Parameters

>> **Min_value** axis minimum value

>> **Max_value** axis maximum value

>> **N** number of steps

`xrayutilities.gridder.`**`ones`**(**args*)

> Compute ones for matrix generation. The shape is determined by the number of input arguments.

xrayutilities.gridder2d module

class `xrayutilities.gridder2d.`**`Gridder2D`**(*nx*, *ny*)

> Bases: `xrayutilities.gridder.Gridder`

> **SetResolution**(*nx*, *ny*)
>> Reset the resolution of the gridder. In this case the original data stored in the object will be deleted.

>> Parameters

>>> **Nx** number of points in x-direction

> **Ny** number of points in y-direction

dataRange (*(xmin, xmax), (ymin, ymax), fixed=True*)
> define minimum and maximum data range, usually this is deduced from the given data automatically, however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside the range are simply ignored
>
> Parameters
>
> > **xmin,ymin** minimum value of the gridding range in x,y
> >
> > **xmax,ymax** maximum value of the gridding range in x,y
> >
> > **fixed** flag to turn fixed range gridding on (True (default)) or of (False)

xaxis

xmatrix

yaxis

ymatrix

xrayutilities.gridder3d module

class xrayutilities.gridder3d.**Gridder3D** (*nx*, *ny*, *nz*)
> Bases: xrayutilities.gridder.Gridder

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

dataRange (*(xmin, xmax), (ymin, ymax), (zmin, zmax), fixed=True*)
> define minimum and maximum data range, usually this is deduced from the given data automatically, however, for sequential gridding it is usefull to set this before the first call of the gridder. data outside the range are simply ignored
>
> Parameters
>
> > **xmin,ymin,zmin** minimum value of the gridding range in x,y,z
> >
> > **xmax,ymax,zmax** maximum value of the gridding range in x,y,z
> >
> > **fixed** flag to turn fixed range gridding on (True (default)) or of (False)

xaxis

xmatrix

yaxis

ymatrix

zaxis

zmatrix

xrayutilities.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 `xrayutilities.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 xrayutilities.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

`xrayutilities.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))

`xrayutilities.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))

`xrayutilities.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))

xrayutilities.q2ang_fit module

Module provides functions to convert a q-vector from reciprocal space to angular space. a simple implementation uses scipy optimize routines to perform a fit for a arbitrary goniometer.

The user is, however, expected to use the bounds variable to put restrictions to the number of free angles to obtain reproducible results. In general only 3 angles are needed to fit an arbitrary q-vector (2 sample + 1 detector angles or 1 sample + 2 detector). More complicated restrictions can be implemented using the lmfit package. (done upon request!)

The function is based on a fitting routine. For a speicific goniometer also analytic expressions from literature can be used.

```
Q2AngFit(qvec, expclass, bounds=None, ormat=array([[ 1.,  0.,  0.],
[ 0.,  1.,  0.],
[ 0.,  0.,  1.]]), startvalues=None)
```
Functions to convert a q-vector from reciprocal space to angular space. This implementation uses scipy optimize routines to perform a fit for a goniometer with arbitrary number of goniometer angles.

The user *must* use the bounds variable to put restrictions to the number of free angles to obtain reproducible results. In general only 3 angles are needed to fit an arbitrary q-vector (2 sample + 1 detector angles or 1 sample + 2 detector).

Parameters

qvecq-vector which should be transformed to reciprocal space

expclassexperimental class used to define the goniometer for which the angles should be calculated.

keyword arguments(optional):

boundslist of bounds of the goniometer angles. The number of bounds must correspond to the number of goniometer angles in the expclass. Angles can also be fixed by supplying only one value for a particular angle. e.g.: ((low,up),fix,(low2,up2),(low3,up3))

ormatorientation matrix of the sample to be used in the conversion

startvaluesstart values for the fit, which can significantly speed up the conversion. The number of values must correspond to the number of angles in the goniometer of the expclass

Returns

fittedangles,errcodelist of fitted goniometer angles and the errcode of the scipy minimize function. for a successful fit the error code should be X

xrayutilities.utilities module

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

`xrayutilities.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

>> **int** 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)
```

xrayutilities.utilities_noconf module

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

`xrayutilities.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

`xrayutilities.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)
```

`xrayutilities.utilities_noconf.`**`wavelength`**`(`*wl*`)`

> convert common energy names to energies in eV

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

Parameter

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

Returns

> wavelength in Angstrom as float

Module contents

xrayutilities is a Python package for assisting with x-ray diffraction experiments. Its the python package included in *xrayutilities*.

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

> **Authors** Dominik Kriegner <dominik.kriegner@gmail.com> and Eugen Wintersberger <eugen.wintersberger@desy.de>

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

X