

---

# **xrayutilities Documentation**

***Release 0.99***

**Dominik Kriegner**

**Eugen Wintersberger**

May 26, 2013



# CONTENTS

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Concept of usage . . . . .	4
1.2	Angle calculation using the material classes . . . . .	5
1.3	hello world . . . . .	5
<b>2</b>	<b>xrayutilities Python package</b>	<b>9</b>
<b>3</b>	<b>Installation</b>	<b>11</b>
3.1	Express instructions . . . . .	11
3.2	Detailed instructions . . . . .	11
3.3	Required third party software . . . . .	11
3.4	Building and installing the library and python package . . . . .	12
3.5	Setup of the Python package . . . . .	12
3.6	Notes for installing on Windows . . . . .	12
<b>4</b>	<b>Examples and API-documentation</b>	<b>15</b>
4.1	Examples . . . . .	15
4.2	API-documentation . . . . .	20
4.3	analysis Package . . . . .	66
4.4	io Package . . . . .	78
4.5	materials Package . . . . .	87
4.6	math Package . . . . .	96
4.7	Modules . . . . .	100
<b>5</b>	<b>Indices and tables</b>	<b>149</b>
	<b>Python Module Index</b>	<b>151</b>
	<b>Index</b>	<b>153</b>



If you look for downloading the package go [here](#). Installation instructions you find further down [Installation](#).

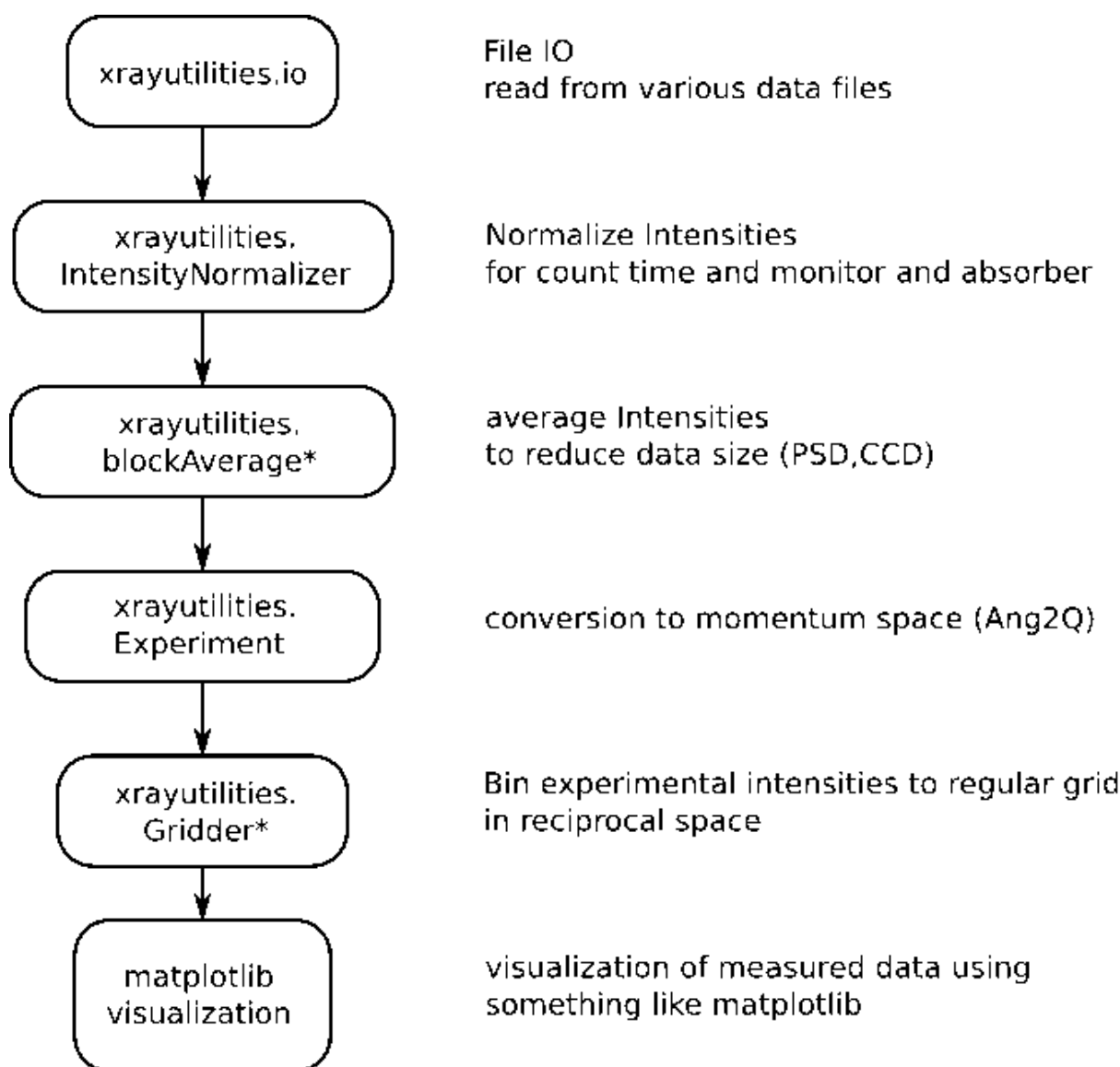


# INTRODUCTION

*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 is especially useful for the reciprocal space conversion of diffraction data taken with linear and area detectors.

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

## 1.1 Concept of usage



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

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

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

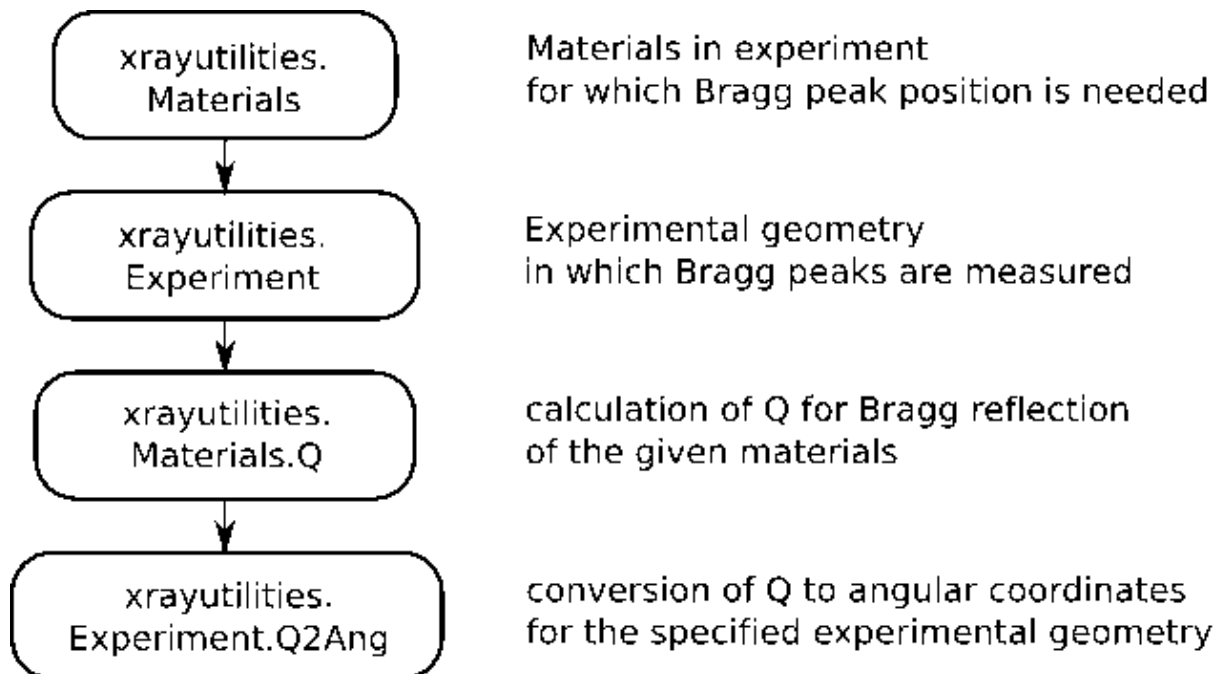
After the conversion to reciprocal space in order to visualize the data it is convenient to transform them to a regular grid in reciprocal space. For this purpose in *xrayutilities* the *gridder Module* is included. For the visualization of



the data in reciprocal space the usage of `matplotlib` is recommended.

A practical example showing the usage is given below.

## 1.2 Angle calculation using the material classes



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

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

## 1.3 hello world

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

```

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

```

14

15

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

```
# global setting for the experiment
sample = "test" # sample name used also as file name for the data file
energy = 8042.5 # x-ray energy in eV
center_ch = 715.9 # center channel of the linear detector
chpdeg = 345.28 # channels per degree of the linear detector
roi=[100,1340] # region of interest of the detector
nchannel = 1500 # number of channels of the detector

# intensity normalizer function responsible for count time and absorber correction
absfun = lambda d: d["detcorr"]/d["psd2"].astype(numpy.float)
normalizer_detcorr = xu.IntensityNormalizer("MCA",mon="Monitor",time="Seconds",absfun=absfun)

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

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

# configure linear detector
# detector direction + parameters need to be given
# mounted along z direction, which corresponds to twotheta
hxr.Ang2Q.init_linear('z-',center_ch,nchannel,chpdeg=chpdeg,roi=roi)

# read spec file and save to HDF5-file
# since reading is much faster from HDF5 once the data are transformed
h5file = os.path.join("data",sample+".h5")
try: s = xu.io.SPECFile(sample+".spec",path="data")
except NameError: s = xu.io.SPECFile(sample+".spec",path="data")
else: s.Update()
s.Save2HDF5(h5file)

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

# read the data from the HDF5 file
#(scan number:36, names of motors in spec file: omega= sample rocking, gamma = twotheta)
[om,tt],MAP = xu.io.get_h5_scan(h5file,36,'omega','gamma')
# normalize the intensity values (absorber and count time corrections)
psdraw = normalizer_detcorr(MAP)
# remove unusable detector channels/regions (no averaging of detector channels)
psd = xu.blockAveragePSD(psdraw, 1, roi=roi)

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

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

# plot the intensity as contour plot using matplotlib
plt.figure()
cf = plt.contourf(gridder.xaxis, gridder.yaxis,INT,100,extend='min')
plt.xlabel(r'$Q_{[11\bar{2}]}$ ($\AA^{-1}$)')
plt.ylabel(r'$Q_{[\bar{1}1\bar{1}]}$ ($\AA^{-1}$)')
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](mailto:dominik.kriegner@gmail.com)> and Eugen Wintersberger <[eu-gen.wintersberger@desy.de](mailto:eu-gen.wintersberger@desy.de)>

for more details see the full *API-documentation*



# INSTALLATION

## 3.1 Express instructions

- install the dependencies (Windows: [pythonxy](#), [SCons](#); Linux/Unix: see below for dependencies).
- download *xrayutilities* from [here](#) or use git to check out the [latest](#) version.
- open a command line and navigate to the downloaded sources and execute:

```
> scons install
```

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

---

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

---

## 3.2 Detailed instructions

**Installing *xrayutilities* is a two steps process**

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

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

## 3.3 Required third party software

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

**The needed dependencies are:**

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

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

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

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

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

## 3.4 Building and installing the library and python package

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

```
>scons
```

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

```
>scons debug=1
```

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

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

The library is installed in `<install path>/lib`. Installation of the Python module is done via the *distutils* package (called by SCons automatically). The `-prefix` option sets the root directory for the installation. If it is omitted the library 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 dependencies) or a pre-packed solution needs to be used. We strongly suggest to use the `python(x,y)` python distribution, which includes already most of the needed dependencies for installing *xrayutilities*.



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

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

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

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



# 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.:

```
1 import tables
2 import xrayutilities as xu
3 import os
4
5 # open spec file or use open SPECfile instance
6 try: s
7 except NameError:
8     s = xu.io.SPECFile("sample_name.spec", path="./specdir")
9
10 # method (1)
11 scan10 = s[9] # Returns a SPECScan class, note 9 because the list starts at 0
12 scan10.ReadData()
13 scan10data = scan10.data
14
```

```
15 # method (2)
16 h5file = os.path.join("h5dir","h5file.h5")
17 s.Save2HDF5(h5file) # save content of SPEC file to HDF5 file
18 # read data from HDF5 file
19 [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 scan10 = s[9] # Returns a SPECScan class
5 scan10.ReadData()
6 scan10data = scan10.data
7
8 # reread data method (2)
9 s.Save2HDF5(h5) # save content of SPEC file to HDF5 file
10 # read data from HDF5 file
11 [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 is 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.

```

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

```

Note that on line 5 the HXRD class is initialized without specifying the energy used in the experiment. It will use the default energy stored in the configuration file, which defaults to CuK  $\alpha_1$ .

One could also call:

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

to specify the energy explicitly. The HXRD class by default describes a four-circle goniometer as described in more detail [here](#).

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

```

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

```

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

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

```

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

```

```
12 peak_width = 0.01 # in q-space
13 resolution = 0.0005 # resolution in q-space
14 In_th, In_int = In_powder.Convolute(resolution, peak_width)
15
16 plt.figure()
17 plt.xlabel(r"2Theta (deg)"); plt.ylabel(r"Intensity")
18 # plot the convoluted signal
19 plt.plot(In_th*2, In_int/In_int.max(), 'k-', label="Indium powder convolution")
20 # 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 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:

```
1 import xrayutilities as xu
2
3 # defining a ZincBlendeLattice with two types of atoms and lattice constant a
4 def ZincBlendeLattice(aa, ab, a):
5     #create lattice base
6     lb = xu.materials.LatticeBase()
7     lb.append(aa, [0, 0, 0])
8     lb.append(aa, [0.5, 0.5, 0])
9     lb.append(aa, [0.5, 0, 0.5])
10    lb.append(aa, [0, 0.5, 0.5])
11    lb.append(ab, [0.25, 0.25, 0.25])
12    lb.append(ab, [0.75, 0.75, 0.25])
```

```

13     lb.append(ab, [0.75,0.25,0.75])
14     lb.append(ab, [0.25,0.75,0.75])
15
16     #create lattice vectors
17     a1 = [a,0,0]
18     a2 = [0,a,0]
19     a3 = [0,0,a]
20
21     l = xu.materials.Lattice(a1,a2,a3,base=lb)
22     return l
23
24     # defining InP, no elastic properties are given,
25     # helper functions exist to create the (6,6) elastic tensor for cubic materials
26     atom_In = xu.materials.elements.In
27     atom_P = xu.materials.elements.P
28     elastictensor = xu.materials.CubicElasticTensor(10.11e+10,5.61e+10,4.56e+10)
29     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:

```

1  import xrayutilities as xu
2  import numpy
3
4  # defining material and experimental setup
5  InAs = xu.materials.InAs
6  energy= 8048 # eV
7
8  # calculate the structure factor for InAs (111) (222) (333)
9  hkllist = [[1,1,1],[2,2,2],[3,3,3]]
10 for hkl in hkllist:
11     qvec = InAs.Q(hkl)
12     F = InAs.StructureFactor(qvec,energy)
13     print (" |F| = %8.3f" %numpy.abs(F))

```

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

```

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.:

```

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)
```

## 4.2 API-documentation

### 4.2.1 xrayutilities 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>

#### 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.

#### 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)

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

#### experiment Module

module helping with planning and analyzing experiments

various classes are provided for

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

**class** 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 identity 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, \*\*kwargs*)

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

**UBmatrix** 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)

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

Returns

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

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

calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

**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.**GID\_ID10B** (*idir*, *ndir*, *\*\*keyargs*)

Bases: xrayutilities.experiment.GID

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

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

**Ang2Q** (*th*, *om*, *delta*, *gamma*, *\*\*kwargs*)

angular to momentum space conversion for a point detector. Also see help GID\_ID10B.Ang2Q for procedures which treat line and area detectors

Parameters

**th,om,delta,gamma**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 th,om,delta,gamma - delta

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

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

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

## Returns

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

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

calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

## Parameters

**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/False (default True) determines if the angles are returned in radians or degrees

## Returns

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

**theta** incidence angle to surface (at the moment always 0)

**omega** sample rotation with respect to the inplane reference direction

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

**gamma** scattering angle

**class** `xrayutilities.experiment.GISAXS` (*idir*, *ndir*, *\*\*kwargs*)

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 ( $\alpha_i$ ,  $2\theta$ ,  $\beta$ ) 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  $\theta$ ,  $\omega$ ,  $\beta$  - 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 angstrom (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, \*\*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)

**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

**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/False (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

**psi\_d**offset of 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)

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

Returns

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

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

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

## Parameters

**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/False (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*)

Convolves 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  $\text{len}(\text{self.sampleAxis}) + \text{len}(\text{detectorAxis})$  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  $\text{len}(\text{self.sampleAxis})$

**dAngles** detector circle angles, number of arguments must correspond to  $\text{len}(\text{self.detectorAxis})$

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

**delta** giving delta angles to correct the given ones for misalignment delta must be a numpy array or list of  $\text{len}(*\text{args})$  used angles are than  $*\text{args} - \text{delta}$

**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 angstrom (default: self.\_wl)

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

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape  $((*) * (\text{self._area\_roi}[1] - \text{self._area\_roi}[0] + 1) * (\text{self._area\_roi}[3] - \text{self._area\_roi}[2] + 1), 3)$  where detectorDir1 is the fastest varying

**detectorAxis**

property handler for \_detectorAxis

Returns

list of detector axis following the syntax  $/[\text{xyz}][+ -]/$

**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** detector rotation around primary beam direction

**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 !!

---

**\*\*kwargs: optional keyword arguments**

**N**avnumber of channels to average to reduce data size (default: [1,1])

**ro**i 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)

**\*\*kwargs: optional keyword arguments**

**N**avnumber of channels to average to reduce data size (default: 1)

**ro**i 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)

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

Returns

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

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

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

Parameters

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

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

**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)

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

Returns

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

**sampleAxis**

property handler for \_sampleAxis

Returns

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

**wavelength**

## gridder Module

**class** xrayutilities.gridder.**Gridder** ( \*\*keyargs)

Bases: object

**KeepData** (bool)

**Normalize** (bool)

**SetChunkSize** (n)

**SetChunkUnit** (u)

**SetThreads** (*n*)

**class** xrayutilities.gridder.**Gridder1D** (*nx*, *\*\*keyargs*)

Bases: xrayutilities.gridder.Gridder

**Clear** ()

**data**

**xaxis**

**class** xrayutilities.gridder.**Gridder2D** (*nx*, *ny*, *\*\*keyargs*)

Bases: xrayutilities.gridder.Gridder

**Clear** ()

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

**data**

**xaxis**

**xmatrix**

**yaxis**

**ymatrix**

**class** xrayutilities.gridder.**Gridder3D** (*nx*, *ny*, *nz*, *\*\*keyargs*)

Bases: xrayutilities.gridder.Gridder2D

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

**zaxis**

**zmatrix**

## libxrayutils Module

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

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

## normalize Module

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

and

provide functions for normalizing intensities for

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

**class** 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,Nav2**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 several 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) )

## 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)
```

## 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

**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

## Subpackages

### analysis Package

`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.

### 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

fw hm 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)

**boundsflag** 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 extracted 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,      tt-  
                                                         center,      om-  
                                                         range,      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)

**boundsflag** 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 extracted 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

**qxequidistant** array of qx momentum transfer

**qzequidistant** 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

**boundsflag** 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

**qxequidistant** array of qx momentum transfer

**qzequidistant** 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 extracted 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)

**boundsflag** 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

**qxequidistant** array of qx momentum transfer

**qzequidistant** 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)

**boundsflag** 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

**qx****tt****qzt**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

&gt;&gt;&gt;

```
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.get_index(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)

### **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

**gridder3d** 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

**gridder3d** 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

**gridder3d** 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.get_index3d(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)

**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])
```

**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), fix=(False, False, False), fig=None, wl=None)`

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

parameters

**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**

**plotflag** 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))

**figmat**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

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

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

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  $\pm$ frangeAA<sup>-1</sup> 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<sup>-1</sup>)

**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) 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:

**r\_iprimary** beam direction as vector [xyz][+-]; default: 'y+'

**detaxis** detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

returns

pixelwidth (at one meter distance) , center\_channel[, detector\_tilt]

---

**Note:** Note:  $L/\text{pixelwidth} \cdot \pi/180 \sim \text{channel/degree}$ , 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*,  
*plot=True*, *omega0=None*)

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

Parameters

**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.

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

**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.

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  $nchannel = center\_ch + chdeg \cdot \tan(\text{angles})$  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)

**dataplot**format of data points

**modellineplot**format of modelline

**modeltiltplot**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:  $L/\text{pixelwidth} \cdot \pi/180 = \text{channel/degree}$  for large detector distance with the sample detector distance 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])
```

## io Package

### edf Module

**class** `xrayutilities.io.edf.EDFDirectory` (*datapath, \*\*keyargs*)

Bases: object

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



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

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

required arguments. :h5: a HDF5 file object

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

**class** xrayutilities.io.edf.**EDFFile** (*fname, \*\*keyargs*)

Bases: object

**ReadData** ()

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

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

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

required arguments. :h5: a HDF5 file object

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

**imagereader Module**

**class** 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.

**panalytical\_xml Module** Panalytical XML ([www.XRDML.com](http://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.getOmPixel` (*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 is not useful!

Returns

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

Example

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

**radicon Module** python module for converting radicon data to HDF5

`xrayutilities.io.radicon.hst2hdf5` (*h5*, *hstfile*, *nofchannels*, *\*\*keyargs*)

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 (named) 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*, *\*\*keyargs*)

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 (named) input arguments:**

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

**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 histograms from 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).

**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

**seifert Module** a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two possibilities 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 is 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, \*args, \*\*keyargs*)

Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument "group" must be used.

**required arguments:**

**h5** a HDF5 file object

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

**optional keyword arguments:**

**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 arguments:**

**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 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 `_`.

**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, **keyargs)
```

Bases: object

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

```
Parse()
```

Parses the file from the starting at last\_offset and adding found scans to the scan list.

```
Save2HDF5(h5f, **keyargs)
```

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

**required arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

**compactivate** compression - true by default

**name** optional name for the file group

```
Update()
```

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

```
class xrayutilities.io.spec.SPECLog(filename, prompt, **keyargs)
```

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, \*\*keyargs*)

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

**input arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

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

**titlea** string with the title for the data

**desca** string with the description of the data

**optattrsa** dictionary with optional attributes to store for the data

**compactivate** compression - true by default

**SetMCAParams** (*mca\_column\_format, mca\_channels, mca\_start, mca\_stop*)

Set the parameters used to save the MCA data to the file. This method calculates the number of lines used to store the MCA data from the number of columns and the

**required input arguments:**

**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.

xrayutilities.io.spec.**geth5\_scan** (*h5f, scans, \*args, \*\*kwargs*)

function to obtain the angular coordinates 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):**

**sample**namestring with the hdf5-group containing the scan data if omitted the first child node of h5f.root will be used

Returns

MAP

or

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

Example

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

**spectra Module** module to handle spectra data

**class** 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 separate 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 `_absorber_factors`. Despite this, costume values can be passed via optional keyword arguments.

**required input arguments:**

**mcamatrix** with the MCA data

**ffilter** values along the scan

**optional keyword arguments:**

**ffcustome** filter factors

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

**recarray2hdf5** (*h5g, rec, name, desc, \*\*keyargs*)

Save a record array in an HDF5 file. A pytables table object is used to store the data.

**required input arguments:**

**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

optional keyword arguments:

**return value:**

**taba** HDF5 table object

**set\_abs\_factors** (*ff*)

Set the global absorber factors in the module.

**spectra2hdf5** (*dir, fname, mcatemp, \*\*keyargs*)

Convert SPECTRA scan data to a HDF5 format.

**required input arguments:**

**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

**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.get_h5_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

**h5file** object of a HDF5 file opened using pytables

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

**\*args: names of the motors (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")

**sample** name string with the hdf5-group containing the scan data if omitted 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 with the MCA data

`xrayutilities.io.spectra.read_mca_dir (dirname, filetemp, **keyargs)`

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

## materials Package

### 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.

**\_create\_database Module** script to create the HDF5 database from the raw data of XOP this file is only needed for administration

**\_create\_database\_alt Module** script to create the HDF5 database from the raw data of XOP this file is only needed for administration

**database Module** module to handle access to the optical parameters database

**class** `xrayutilities.materials.database.DataBase` (*fname*)

Bases: `object`

**Close** ()

Close an opened 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 labels 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 labels 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.

**required 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)
```

## elements Module

**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.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.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.

**required 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)

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

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

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*)

**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 triangle 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*)

**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 symmetric 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 matrix 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 extensions (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 matrix 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)

calculates 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)

calculates 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)

calculates 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 ( $n = 1 + \text{chi\_r0}/2 + i*\text{chi\_i0}/2$  vs.  $n = 1 - \text{delta} + i*\text{beta}$ )

**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/Å)

**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/Å)

**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 ( $n=1-\text{delta}+i*\text{beta}$ )

Parameter

**en**x-ray energy eV, if omitted the value from the xrayutilities configuration is used

Returns

delta (float)

**idx\_refraction** (*en='config'*)

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

Parameter

**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.

**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*)

## math Package

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

`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](https://github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting)

Parameters

**xdata***x*coordinates of the data to be fitted

**ydata***y*coordinates 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

**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](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`

`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.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,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.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.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,YCEN2]

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)

## **ttransforms 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

`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 radians.

`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 radians.

`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 radians.

`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

**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  $v1.v2 = \text{norm}(v1) * \text{norm}(v2) * \cos(\alpha)$

$\alpha = \arccos((v1.v2)/(\text{norm}(v1) * \text{norm}(v2)))$

**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 radians

**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

## 4.3 analysis Package

`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.

### 4.3.1 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, tt-
                                                            center,      om-
                                                            range,      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 subs cans 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 extracted 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 subs cans 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

&gt;&gt;&gt;

```
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 subs cans 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 extracted 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 extracted 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 extracted 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.get_index(x, y, xgrid, ygrid)`

gives the indices of the point x,y in the grid given by xgrid ygrid, 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)

### 4.3.2 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.get_index3d(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)

### 4.3.3 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

**surhkl** of the surface

**inpinplane** reference peak or direction

Returns

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

Example

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

#### 4.3.4 sample\_align Module

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

```
xrayutilities.analysis.sample_align.area_detector_calib(angle1, angle2,
                                                         ccdimages, de-
                                                         taxi, r_i, plot=True,
                                                         cut_off=0.7, start=(0,
                                                         0, 0, 0), fix=(False,
                                                         False, False, False),
                                                         fig=None, wl=None)
```

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

parameters

**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

```
xrayutilities.analysis.sample_align.fit_bragg_peak(om, tt, psd, omalign, ttalign,
                                                    exphrd, frange=(0.03, 0.03),
                                                    plot=True)
```

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

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  $\pm$  frange  $AA^{-1}$  of those values

**exphxr** 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:  $AA^{-1}$ )

**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) 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:

**r\_i** primary beam direction as vector [xyz][+-]; default: 'y+'

**detaxis** detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

## returns

pixelwidth (at one meter distance) , center\_channel[, detector\_tilt]

---

**Note:** Note:  $L/\text{pixelwidth} \cdot \pi/180 \approx \text{channel/degree}$ , 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*,  
*plot=True*, *omega0=None*)

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

## Parameters

**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.

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

**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.

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  $nchannel = center\_ch + chdeg * \tan(\text{angles})$  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

**plotflag** 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:  $L/\text{pixelwidth} \cdot \pi/180 = \text{channel/degree}$  for large detector distance with the sample detector distance  $L$

---

`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])
```

## 4.4 io Package

### 4.4.1 edf Module

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

Bases: object

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

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

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

required arguments. :h5: a HDF5 file object

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

**class** `xrayutilities.io.edf.EDFFile(fname, **keyargs)`

Bases: object

**ReadData** ()

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

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

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

required arguments. :h5: a HDF5 file object

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

## 4.4.2 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.

## 4.4.3 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.**XRDMLEFile** (*fname*)

Bases: object

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

**class** xrayutilities.io.panalytical\_xml.**XRDMLEMeasurement** (*measurement*)

Bases: object

class to handle scans in a XRDML datafile

xrayutilities.io.panalytical\_xml.**getOmPixel** (*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.XRDMLEFile 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 is not useful!

**Returns**

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

**Example**

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

#### 4.4.4 radicon Module

python module for converting radicon data to HDF5

`xrayutilities.io.radicon.hst2hdf5` (*h5, hstfile, nofchannels, \*\*keyargs*)

Converts a HST file to an HDF5 file.

**Required input arguments:**

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

**hstfilename** of the HST file

**nofchannels** number of channels

**optional (named) 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, \*\*keyargs*)

Converts a RDC file to an HDF5 file.

**Required input arguments:**

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

**rdcfilename** of the RDC file

**optional (named) input arguments:**

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

**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 histograms from 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).

## 4.4.5 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

## 4.4.6 seifert Module

a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two possibilities 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\_attris** (*obj*)

**class** xrayutilities.io.seifert.**SeifertMultiScan** (*filename, m\_scan, m2, path=None*)

Bases: object

**dump2hdf5** (*h5, \*args, \*\*keyargs*)

Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument "group" must be used.

**required arguments:**

**h5** a HDF5 file object

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

**optional keyword arguments:**

**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 arguments:**

**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 preceding `_`.

## 4.4.7 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, **keyargs)
```

Bases: object

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

**Parse** ()

Parses the file from the starting at last\_offset and adding found scans to the scan list.

**Save2HDF5** (*h5f, \*\*keyargs*)

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

**required arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

**compactivate** compression - true by default

**name** optional name for the file group

**Update** ()

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

```
class xrayutilities.io.spec.SPECLog (filename, prompt, **keyargs)
```

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*, *\*\*keyargs*)**

Save a SPEC scan to an HDF5 file. The method creates a group with the name of the scan and stores the data there as a table object with name “data”. By default the scan group is created under the root group of the HDF5 file. The title of the scan group is usually 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 *optattr*s keyword argument.

**input arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

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

**title**a string with the title for the data

**desca** string with the description of the data

**optattr**sa dictionary with optional attributes to store for the data

**compact**ivate 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 arguments:**

**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.

**xrayutilities.io.spec.geth5\_scan (*h5f*, *scans*, *\*args*, *\*\*kwargs*)**

function to obtain the angular coordinates 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) :tname: e.g. name of the two theta motor (or its equivalent)

**\*\*kwargs (optional):**

**sample**namestring with the hdf5-group containing the scan data if omitted the first child node of *h5f.root* will be used

**Returns**

MAP

or



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

Example

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

#### 4.4.8 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 separate 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:**

**mcamatrix** with the MCA data

**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, \*\*keyargs*)

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

optional keyword arguments:

**return value:**

**taba** HDF5 table object

**set\_abs\_factors** (*ff*)

Set the global absorber factors in the module.

**spectra2hdf5** (*dir, fname, mcatemp, \*\*keyargs*)

Convert SPECTRA scan data to a HDF5 format.

**required input arguments:**

**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

**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

**h5file** object of a HDF5 file opened using pytables

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

**\*args: names of the motors (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")

**sample** name string with the hdf5-group containing the scan data if omitted 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 with the MCA data

`xrayutilities.io.spectra.read_mca_dir` (*dirname*, *filetemp*, *\*\*keyargs*)

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

## 4.5 materials Package

### 4.5.1 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.

## 4.5.2 `_create_database` Module

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

## 4.5.3 `_create_database_alt` Module

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

## 4.5.4 `database` Module

module to handle access to the optical parameters database

**class** `xrayutilities.materials.database.Database` (*fname*)

Bases: `object`

**Close** ()

Close an opened database file.

**Create** (*dbname*, *dbdesc*)

Creates a new database. If the database file already exists its content is deleted.

**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.

**required 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)
```

## 4.5.5 elements Module

## 4.5.6 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.**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.**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.

**required input arguments:****epsa** 3x3 matrix independent strain components**GetPoint** (\*args)

determine lattice points with indices given in the argument

## Examples

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

or

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

**ReciprocalLattice** ()**UnitCellVolume** ()

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

**class** 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)**bb**-factor of the atom used as  $\exp(-b^2q^2/(4\pi)^2)$  to reduce the intensity of this atom (only used in case of temp=0 in StructureFactor and chi calculation)xrayutilities.materials.lattice.**NaumanniteLattice** (aa, ab, a, b, c)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)

## 4.5.7 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 triangle 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*)

**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`

**ContentBsym** (*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 symmetric 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 matrix 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 extensions (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 matrix 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)

calculates 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)

calculates 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)

calculates 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 ( $n = 1 + \text{chi\_r0}/2 + i*\text{chi\_i0}/2$  vs.  $n = 1 - \text{delta} + i*\text{beta}$ )

**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/Å)

**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/Å)

**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 ( $n=1-\text{delta}+i*\text{beta}$ )

Parameter

**en** x-ray energy eV, if omitted the value from the xrayutilities configuration is used

Returns

delta (float)

**idx\_refraction** (*en='config'*)  
function to calculate the complex index of refraction of a material in the x-ray range

Parameter

**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.

**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_ij2ijk1` (*ij*)

`xrayutilities.materials.material.index_map_ijk12ij` (*i, j*)

## 4.6 math Package

### 4.6.1 fit Module

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

`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](https://github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting)

Parameters

**xdata**coordinates 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

## 4.6.2 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](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

`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.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,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.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.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,YCEN2]

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)

### 4.6.3 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

`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 radians.

`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 radians.

`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 radians.

`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

## 4.6.4 vector Module

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

`xrayutilities.math.vector.VecAngle(v1, v2, deg=False)`

calculate the angle between two vectors. The following formula is used  $v1.v2 = \text{norm}(v1) * \text{norm}(v2) * \cos(\alpha)$

$\alpha = \arccos((v1.v2)/(\text{norm}(v1) * \text{norm}(v2)))$

**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 radians

**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

## 4.7 Modules

### 4.7.1 API-documentation

#### **xrayutilities 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](mailto:dominik.kriegner@gmail.com)> and Eugen Wintersberger <[eu-gen.wintersberger@desy.de](mailto:eu-gen.wintersberger@desy.de)>



## 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.

## 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)

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

## experiment Module

module helping with planning and analyzing experiments

various classes are provided for

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

**class** `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 identity 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*, *\*\*kwargs*)

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/False (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.GID\_ID10B (*idir, ndir, \*\*keyargs*)

Bases: xrayutilities.experiment.GID

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

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

**Ang2Q** (*th, om, delta, gamma, \*\*kwargs*)

angular to momentum space conversion for a point detector. Also see help GID\_ID10B.Ang2Q for procedures which treat line and area detectors

Parameters

**th, om, delta, gamma**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 a numpy array or list of length 4. used angles are theta, omega, delta, gamma - 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 angstrom (default: self.\_wl)**degflag** to tell if angles are passed as degree (default: True)

Returns

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

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

calculate the GID angles needed in the experiment the inplane reference direction defines the direction were the reference direction is parallel to the primary beam (i.e. lattice planes perpendicular to the beam)

Parameters

**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/False (default True) determines if the angles are returned in radians or degrees

Returns

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

**theta**incidence angle to surface (at the moment always 0)

**omega**sample rotation with respect to the inplane reference direction

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

**gamma**scattering angle

**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)

**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, \*\*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

**UBmatrix** 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)

**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

**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 of 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 then 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 angstrom (default: self.\_wl)

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

Returns

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

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

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

Parameters

**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/False (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)

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

Returns

reciprocal space position of all detector pixels in a numpy.ndarray of shape (  $(*) \cdot (\text{self._area\_roi}[1] - \text{self._area\_roi}[0] + 1) \cdot (\text{self._area\_roi}[3] - \text{self._area\_roi}[2] + 1)$  , 3 ) were detectorDir1 is the fastest varying

#### **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**detector rotation around primary beam direction

**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 !!

---

#### **\*\*kwargs: optional keyword arguments**

**N**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)

#### **\*\*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)

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

Returns

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

#### **point** (\*args, \*\*kwargs)

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

Parameters

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

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

**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`)

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

Returns

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

**sampleAxis**

property handler for `_sampleAxis`

Returns

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

**wavelength**

## gridder Module

```
class xrayutilities.gridder.Gridder (**keyargs)
```

Bases: `object`

**KeepData** (*bool*)

**Normalize** (*bool*)

**SetChunkSize** (*n*)

**SetChunkUnit** (*u*)

**SetThreads** (*n*)

```
class xrayutilities.gridder.Gridder1D (nx, **keyargs)
```

Bases: `xrayutilities.gridder.Gridder`

**Clear** ()

**data**

**xaxis**

```
class xrayutilities.gridder.Gridder2D (nx, ny, **keyargs)
```

Bases: `xrayutilities.gridder.Gridder`

**Clear** ()

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

**data**

**xaxis**  
**xmatrix**  
**yaxis**  
**ymatrix**

```
class xrayutilities.gridder.Gridder3D (nx, ny, nz, **keyargs)
    Bases: xrayutilities.gridder.Gridder2D
    SetResolution (nx, ny, nz)
    zaxis
    zmatrix
```

### libxrayutils Module

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

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

### normalize Module

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

and

provide functions for normalizing intensities for

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

```
class 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,Nav2**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 several 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)` )

## 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<sup>^(abslow)</sup> will be taken as lower boundary

**abshigh**10<sup>^(abshigh)</sup> will be taken as higher boundary

**Returns**

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

**Example**

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

**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**

**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 wavelength 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

## Subpackages

**analysis Package** 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.

### 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

fhwm 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, tt-  
center, om-  
range, 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 extracted 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)

**qxposition** 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 extracted 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 extracted 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

**qx,qt,qtz**reciprocal space boundaries of 2theta scan (bounds=False)

**tt,ttint,(qxb,qzb)**2theta scan coordinates and intensities + reciprocal space bounds of the extracted 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 extracted 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.get_index(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)

**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

**gridder3d** 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

**gridder3d** 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

**gridder3d** 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.get_index3d(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)

**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])
```

**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, de-
                                                         taxis, r_i, plot=True,
                                                         cut_off=0.7, start=(0,
                                                         0, 0, 0), fix=(False,
                                                         False, False),
                                                         fig=None, wl=None)
```

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

parameters

**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**

**plotflag** 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))

**figmat**plotlib 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

`xrayutilities.analysis.sample_align.fit_bragg_peak` (*om, tt, psd, omalign, ttalign, exphrd, frange=(0.03, 0.03), plot=True*)

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

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<sup>-1</sup> of those values

**exphrd**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<sup>-1</sup>)

**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) 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:

**r\_i**primary beam direction as vector [xyz][+]; default: 'y+'

**detaxis**detector arm rotation axis [xyz][+-] e.g. 'x+'; default: 'x+'

returns

pixelwidth (at one meter distance) , center\_channel[, detector\_tilt]

---

**Note:** Note:  $L/\text{pixelwidth} \cdot \pi/180 \sim \text{channel/degree}$ , 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*,  
*plot=True*, *omega0=None*)

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

Parameters

**phiazimuths** in which the reflection was aligned (deg)

**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.

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

**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.

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  $nchannel = center\_ch + chdeg \cdot \tan(\text{angles})$  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)



**dataplot** format of data points

**modellineplot** format of modelline

**modeltiltplot** 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:  $L/\text{pixelwidth} \cdot \pi/180 = \text{channel/degree}$  for large detector distance with the sample detector distance  $L$

---

`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])
```

## io Package

### edf Module

**class** `xrayutilities.io.edf.EDFDirectory` (*datapath, \*\*keyargs*)

Bases: `object`

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

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

`Save2HDF5(h5, **keyargs)`: Saves the data stored in the EDF files in the specified directory in a HDF5 file as a HDF5 arrays in a subgroup. By default the data is stored in a group given by the foldername -

this can be changed by passing the name of a target group or a path to the target group via the “group” keyword argument.

required arguments. :h5: a HDF5 file object

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

```
class xrayutilities.io.edf.EDFFile (fname, **keyargs)
```

Bases: object

**ReadData** ()

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

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

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

required arguments. :h5: a HDF5 file object

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

### imagereader Module

```
class 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.

**panalytical\_xml Module** Panalytical XML ([www.XRDML.com](http://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.getOmPixel` (*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 is not useful!

Returns

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

Example

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

**radicon Module** python module for converting radicon data to HDF5

`xrayutilities.io.radicon.hst2hdf5` (*h5*, *hstfile*, *nofchannels*, *\*\*keyargs*)

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 (named) 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*, *\*\*keyargs*)

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 (named) input arguments:**

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

**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 histograms from 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).

**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

**seifert Module** a set of routines to convert Seifert ASCII files to HDF5 in fact there exist two possibilities 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\_attr** (*obj*)

**class** `xrayutilities.io.seifert.SeifertMultiScan` (*filename, m\_scan, m2, path=None*)

Bases: object

**dump2hdf5** (*h5, \*args, \*\*keyargs*)

Saves the content of a multi-scan file to a HDF5 file. By default the data is stored in the root group of the file. To save data somewhere else the keyword argument “group” must be used.

**required arguments:**

**h5**a HDF5 file object

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

**optional keyword arguments:**

**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 arguments:**

**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 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 preceding `_`.

**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** (*h5f, \*\*keyargs*)

**class** `xrayutilities.io.spec.SPECFile(filename, **keyargs)`

Bases: `object`

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

**Parse()**

Parses the file from the starting at `last_offset` and adding found scans to the scan list.

**Save2HDF5** (*h5f, \*\*keyargs*)

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

**required arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

**compactivate** compression - true by default

**name** optional name for the file group

**Update()**

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

**class** `xrayutilities.io.spec.SPECLog(filename, prompt, **keyargs)`

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, \*\*keyargs*)

Save a SPEC scan to an HDF5 file. The method creates a group with the name of the scan and stores the data there as a table object with name "data". By default the scan group is created under the root group of the HDF5 file. The title of the scan group is usually 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 *optattr*s keyword argument.

**input arguments:**

**h5fa** HDF5 file object or its filename

**optional keyword arguments:**

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

**title**a string with the title for the data

**desca** string with the description of the data

**optattr**s a dictionary with optional attributes to store for the data

**compact** 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 arguments:**

**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.

**xrayutilities.io.spec.geth5\_scan** (*h5f, scans, \*args, \*\*kwargs*)

function to obtain the angular coordinates 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):**

**sample**namestring with the hdf5-group containing the scan data if omitted the first child node of h5f.root will be used

Returns

MAP

or

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

Example

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

**spectra Module** module to handle spectra data

**class** 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 separate 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 `_absorber_factors`. Despite this, costume values can be passed via optional keyword arguments.

**required input arguments:**

**mcamatrix** with the MCA data

**ffilter** values along the scan

**optional keyword arguments:**

**ffcustome** filter factors

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

**recarray2hdf5** (*h5g, rec, name, desc, \*\*keyargs*)

Save a record array in an HDF5 file. A pytables table object is used to store the data.

**required input arguments:**

**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

optional keyword arguments:

**return value:**

**taba** HDF5 table object

**set\_abs\_factors** (*ff*)

Set the global absorber factors in the module.

**spectra2hdf5** (*dir, fname, mcatemp, \*\*keyargs*)

Convert SPECTRA scan data to a HDF5 format.

**required input arguments:**

**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

**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.get_h5_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

**h5file** object of a HDF5 file opened using pytables

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

**\*args: names of the motors (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")

**sample** name string with the hdf5-group containing the scan data if omitted 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 with the MCA data

`xrayutilities.io.spectra.read_mca_dir (dirname, filetemp, **keyargs)`

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

## materials Package

### 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.

**\_create\_database Module** script to create the HDF5 database from the raw data of XOP this file is only needed for administration

**\_create\_database\_alt Module** script to create the HDF5 database from the raw data of XOP this file is only needed for administration

**database Module** module to handle access to the optical parameters database

**class** `xrayutilities.materials.database.DataBase (fname)`

Bases: `object`

**Close ()**

Close an opened 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 labels 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 labels 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.

**required 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)
```

## elements Module

**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.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.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:**

**epsa** 3x3 matrix independent strain components

**GetPoint** (*\*args*)

determine lattice points with indices given in the argument

Examples

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

or

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

**ReciprocalLattice** ()

**UnitCellVolume** ()

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

**class** 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)

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

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

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*)

**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 triangle 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*)

**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 symmetric 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 matrix 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 extensions (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 matrix 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*)

calculates 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*)

calculates 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*)

calculates 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 ( $n = 1 + \text{chi\_r0}/2 + i*\text{chi\_i0}/2$  vs.  $n = 1 - \text{delta} + i*\text{beta}$ )

**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/Å)

**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/Å)

**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 ( $n=1-\text{delta}+i*\text{beta}$ )

Parameter

**en**x-ray energy eV, if omitted the value from the xrayutilities configuration is used

Returns

delta (float)

**idx\_refraction** (*en='config'*)

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

Parameter

**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.

**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*)

## math Package

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

`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](https://github.com/tiagopereira/python_tips/wiki/Scipy%3A-curve-fitting)

Parameters

**xdata** coordinates of the data to be fitted

**ydata** coordinates 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

**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](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**

`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.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,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.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.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,YCEN2,AMP2,ANGLE2]  
 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)

## 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

`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 radians.

`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 radians.

`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 radians.

`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

**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  $v1.v2 = \text{norm}(v1) * \text{norm}(v2) * \cos(\alpha)$

$\alpha = \arccos((v1.v2)/(\text{norm}(v1) * \text{norm}(v2)))$

**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 radians

**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





# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*



# PYTHON MODULE INDEX

## X

- `xrayutilities`, 100
- `xrayutilities.analysis`, 114
- `xrayutilities.analysis.line_cuts`, 114
- `xrayutilities.analysis.line_cuts3d`, 121
- `xrayutilities.analysis.misc`, 122
- `xrayutilities.analysis.sample_align`, 122
- `xrayutilities.config`, 101
- `xrayutilities.exception`, 101
- `xrayutilities.experiment`, 101
- `xrayutilities.gridder`, 110
- `xrayutilities.io`, 125
- `xrayutilities.io.edf`, 125
- `xrayutilities.io.imagereader`, 126
- `xrayutilities.io.panalytical_xml`, 127
- `xrayutilities.io.radicon`, 127
- `xrayutilities.io.rotanode_alignment`, 128
- `xrayutilities.io.seifert`, 128
- `xrayutilities.io.spec`, 130
- `xrayutilities.io.spectra`, 132
- `xrayutilities.libxrayutils`, 111
- `xrayutilities.materials`, 135
- `xrayutilities.materials._create_database`, 135
- `xrayutilities.materials._create_database_alt`, 135
- `xrayutilities.materials.cif`, 135
- `xrayutilities.materials.database`, 135
- `xrayutilities.materials.elements`, 137
- `xrayutilities.materials.lattice`, 137
- `xrayutilities.materials.material`, 139
- `xrayutilities.math`, 143
- `xrayutilities.math.fit`, 143
- `xrayutilities.math.functions`, 144
- `xrayutilities.math.transforms`, 145
- `xrayutilities.math.vector`, 146
- `xrayutilities.normalize`, 111
- `xrayutilities.utilities`, 112
- `xrayutilities.utilities_noconf`, 113



# INDEX

## A

a1 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
a2 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
a3 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
abs\_corr() (xrayutilities.io.spectra.Spectra method), 52, 86, 133  
absfun (xrayutilities.normalize.IntensityNormalizer attribute), 30, 111  
add\_f0\_from\_intertab() (in module xrayutilities.materials.database), 55, 89, 136  
add\_f0\_from\_xop() (in module xrayutilities.materials.database), 55, 89, 136  
add\_f1f2\_from\_ascii\_file() (in module xrayutilities.materials.database), 55, 89, 136  
add\_f1f2\_from\_henkedb() (in module xrayutilities.materials.database), 55, 89, 136  
add\_f1f2\_from\_kissel() (in module xrayutilities.materials.database), 55, 89, 136  
add\_mass\_from\_NIST() (in module xrayutilities.materials.database), 56, 89, 137  
Alloy (class in xrayutilities.materials.material), 58, 91, 139  
Ang2HKL() (xrayutilities.experiment.Experiment method), 20, 101  
Ang2Q() (xrayutilities.experiment.GID method), 21, 102  
Ang2Q() (xrayutilities.experiment.GID\_ID10B method), 22, 103  
Ang2Q() (xrayutilities.experiment.GISAXS method), 23, 104  
Ang2Q() (xrayutilities.experiment.HXRD method), 24, 104  
Ang2Q() (xrayutilities.experiment.NonCOP method), 25, 106  
append() (xrayutilities.io.spectra.SPECTRAFileData method), 52, 85, 133  
append() (xrayutilities.materials.lattice.LatticeBase method), 57, 91, 138  
ApplyStrain() (xrayutilities.materials.lattice.Lattice method), 57, 90, 138  
ApplyStrain() (xrayutilities.materials.material.Material method), 59, 93, 140

area() (xrayutilities.experiment.QConversion method), 26, 107  
area\_detector\_calib() (in module xrayutilities.analysis.sample\_align), 41, 75, 122  
Atom (class in xrayutilities.materials.lattice), 56, 90, 137  
avmon (xrayutilities.normalize.IntensityNormalizer attribute), 30, 111  
AxisToZ (class in xrayutilities.math.transforms), 64, 98, 145

## B

B (xrayutilities.materials.material.Material attribute), 59, 93, 140  
b1 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
b2 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
b3 (xrayutilities.materials.material.Material attribute), 60, 94, 141  
BaddeleyiteLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
BCCLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
BCTLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
beta() (xrayutilities.materials.material.Material method), 60, 94, 141  
blockAverage1D() (in module xrayutilities.normalize), 31, 111  
blockAverage2D() (in module xrayutilities.normalize), 31, 112  
blockAveragePSD() (in module xrayutilities.normalize), 31, 112

## C

chi0() (xrayutilities.materials.material.Material method), 61, 95, 142  
chih() (xrayutilities.materials.material.Material method), 61, 95, 142  
CIFFile (class in xrayutilities.materials.cif), 54, 87, 135  
Cij2Cijkl() (in module xrayutilities.materials.material), 58, 92, 139  
Cij2Cijkl() (in module xrayutilities.math.transforms), 64, 98, 145

- Cijkl2Cij() (in module xrayutilities.materials.material), 58, 92, 139
- Cijkl2Cij() (in module xrayutilities.math.transforms), 64, 98, 145
- Clear() (xrayutilities.gridder.Gridder1D method), 30, 110
- Clear() (xrayutilities.gridder.Gridder2D method), 30, 110
- ClearData() (xrayutilities.io.spec.SPECSScan method), 50, 83, 131
- Close() (xrayutilities.materials.database.DataBase method), 54, 88, 135
- ContentBasym() (xrayutilities.materials.material.CubicAlloy method), 58, 92, 139
- ContentBsym() (xrayutilities.materials.material.CubicAlloy method), 58, 92, 139
- Convolute() (xrayutilities.experiment.Powder method), 26, 107
- CoordinateTransform (class in xrayutilities.math.transforms), 64, 98, 145
- Create() (xrayutilities.materials.database.DataBase method), 54, 88, 135
- CreateMaterial() (xrayutilities.materials.database.DataBase method), 54, 88, 135
- critical\_angle() (xrayutilities.materials.material.Material method), 61, 95, 142
- CubicAlloy (class in xrayutilities.materials.material), 58, 92, 139
- CubicElasticTensor() (in module xrayutilities.materials.material), 59, 93, 140
- CubicFm3mBaF2() (in module xrayutilities.materials.lattice), 56, 90, 137
- CubicLattice() (in module xrayutilities.materials.lattice), 56, 90, 137
- CuMnAsLattice() (in module xrayutilities.materials.lattice), 56, 90, 137
- ## D
- darkfield (xrayutilities.normalize.IntensityNormalizer attribute), 30, 111
- data (xrayutilities.gridder.Gridder1D attribute), 30, 110
- data (xrayutilities.gridder.Gridder2D attribute), 30, 110
- DataBase (class in xrayutilities.materials.database), 54, 88, 135
- Debye1() (in module xrayutilities.math.functions), 63, 97, 144
- delta() (xrayutilities.materials.material.Material method), 61, 95, 142
- det (xrayutilities.normalize.IntensityNormalizer attribute), 31, 111
- detectorAxis (xrayutilities.experiment.QConversion attribute), 27, 108
- DiamondLattice() (in module xrayutilities.materials.lattice), 56, 90, 137
- dTheta() (xrayutilities.materials.material.Material method), 61, 95, 142
- dump2h5() (xrayutilities.io.seifert.SeifertScan method), 48, 81, 129
- dump2hdf5() (xrayutilities.io.seifert.SeifertMultiScan method), 47, 81, 128
- dump2mlab() (xrayutilities.io.seifert.SeifertMultiScan method), 48, 81, 129
- dump2mlab() (xrayutilities.io.seifert.SeifertScan method), 48, 82, 129
- ## E
- EDFDirectory (class in xrayutilities.io.edf), 44, 78, 125
- EDFFile (class in xrayutilities.io.edf), 45, 78, 126
- energy (xrayutilities.experiment.Experiment attribute), 21, 102
- energy (xrayutilities.experiment.QConversion attribute), 27, 108
- energy() (in module xrayutilities.utilities\_noconf), 32, 113
- Experiment (class in xrayutilities.experiment), 20, 101
- ## F
- f() (xrayutilities.materials.lattice.Atom method), 56, 90, 137
- f0() (xrayutilities.materials.lattice.Atom method), 56, 90, 137
- f1() (xrayutilities.materials.lattice.Atom method), 56, 90, 137
- f2() (xrayutilities.materials.lattice.Atom method), 56, 90, 137
- FCCLattice() (in module xrayutilities.materials.lattice), 56, 90, 137
- fit\_bragg\_peak() (in module xrayutilities.analysis.sample\_align), 42, 75, 123
- fit\_peak2d() (in module xrayutilities.math.fit), 62, 96, 143
- flatfield (xrayutilities.normalize.IntensityNormalizer attribute), 31, 111
- fwhm\_exp() (in module xrayutilities.analysis.line\_cuts), 33, 66, 114
- ## G
- Gauss1d() (in module xrayutilities.math.functions), 63, 97, 144
- Gauss1d\_der\_p() (in module xrayutilities.math.functions), 63, 97, 144
- Gauss1d\_der\_x() (in module xrayutilities.math.functions), 63, 97, 144
- Gauss2d() (in module xrayutilities.math.functions), 63, 97, 144
- gauss\_fit() (in module xrayutilities.math.fit), 62, 96, 143
- GeneralPrimitiveLattice() (in module xrayutilities.materials.lattice), 56, 90, 137
- GeneralUC() (in module xrayutilities.materials.material), 59, 93, 140

get() (xrayutilities.io.rotanode\_alignment.RA\_Alignment method), 47, 81, 128  
 get\_omega\_scan\_ang() (in module xrayutilities.analysis.line\_cuts), 33, 66, 114  
 get\_omega\_scan\_bounds\_ang() (in module xrayutilities.analysis.line\_cuts), 34, 67, 114  
 get\_omega\_scan\_q() (in module xrayutilities.analysis.line\_cuts), 34, 68, 115  
 get\_qx\_scan() (in module xrayutilities.analysis.line\_cuts), 35, 68, 116  
 get\_qx\_scan3d() (in module xrayutilities.analysis.line\_cuts3d), 40, 73, 121  
 get\_qy\_scan3d() (in module xrayutilities.analysis.line\_cuts3d), 40, 73, 121  
 get\_qz\_scan() (in module xrayutilities.analysis.line\_cuts), 35, 69, 116  
 get\_qz\_scan3d() (in module xrayutilities.analysis.line\_cuts3d), 40, 74, 121  
 get\_qz\_scan\_int() (in module xrayutilities.analysis.line\_cuts), 36, 69, 116  
 get\_radial\_scan\_ang() (in module xrayutilities.analysis.line\_cuts), 36, 70, 117  
 get\_radial\_scan\_bounds\_ang() (in module xrayutilities.analysis.line\_cuts), 37, 70, 118  
 get\_radial\_scan\_q() (in module xrayutilities.analysis.line\_cuts), 37, 71, 118  
 get\_spectra\_files() (in module xrayutilities.io.spectra), 53, 86, 134  
 get\_ttheta\_scan\_ang() (in module xrayutilities.analysis.line\_cuts), 38, 71, 119  
 get\_ttheta\_scan\_bounds\_ang() (in module xrayutilities.analysis.line\_cuts), 38, 72, 119  
 get\_ttheta\_scan\_q() (in module xrayutilities.analysis.line\_cuts), 39, 72, 120  
 getangles() (in module xrayutilities.analysis.misc), 41, 74, 122  
 GetF0() (xrayutilities.materials.database.DataBase method), 54, 88, 135  
 GetF1() (xrayutilities.materials.database.DataBase method), 54, 88, 135  
 GetF2() (xrayutilities.materials.database.DataBase method), 55, 88, 136  
 geth5\_scan() (in module xrayutilities.io.spec), 50, 84, 131  
 geth5\_spectra\_map() (in module xrayutilities.io.spectra), 53, 86, 134  
 getindex() (in module xrayutilities.analysis.line\_cuts), 39, 73, 120  
 getindex3d() (in module xrayutilities.analysis.line\_cuts3d), 41, 74, 122  
 GetMismatch() (xrayutilities.materials.material.Material method), 59, 93, 140  
 getOmPixel() (in module xrayutilities.io.panalytical\_xml), 46, 79, 127  
 GetPoint() (xrayutilities.materials.lattice.Lattice method), 57, 91, 138  
 getSeifert\_map() (in module xrayutilities.io.seifert), 48, 82, 129  
 getSyntax() (in module xrayutilities.math.vector), 66, 100, 147  
 getVector() (in module xrayutilities.math.vector), 66, 100, 147  
 getxrddl\_map() (in module xrayutilities.io.panalytical\_xml), 46, 79, 127  
 GID (class in xrayutilities.experiment), 21, 102  
 GID\_ID10B (class in xrayutilities.experiment), 22, 103  
 GISAXS (class in xrayutilities.experiment), 23, 104  
 Gridder (class in xrayutilities.gridder), 29, 110  
 Gridder1D (class in xrayutilities.gridder), 30, 110  
 Gridder2D (class in xrayutilities.gridder), 30, 110  
 Gridder3D (class in xrayutilities.gridder), 30, 111

## H

HCPLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
 Hexagonal3CLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
 Hexagonal4HLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
 Hexagonal6HLattice() (in module xrayutilities.materials.lattice), 56, 90, 137  
 HexagonalElasticTensor() (in module xrayutilities.materials.material), 59, 93, 140  
 hst2hdf5() (in module xrayutilities.io.radicon), 46, 80, 127  
 HXRD (class in xrayutilities.experiment), 23, 104

## I

idx\_refraction() (xrayutilities.materials.material.Material method), 61, 95, 142  
 ImageReader (class in xrayutilities.io.imagereader), 45, 79, 126  
 index\_map\_ij2ijkl() (in module xrayutilities.materials.material), 62, 96, 143  
 index\_map\_ij2ijkl() (in module xrayutilities.math.transforms), 65, 99, 146  
 index\_map\_ijkl2ij() (in module xrayutilities.materials.material), 62, 96, 143  
 index\_map\_ijkl2ij() (in module xrayutilities.math.transforms), 65, 99, 146  
 init\_area() (xrayutilities.experiment.QConversion method), 27, 108  
 init\_linear() (xrayutilities.experiment.QConversion method), 28, 108  
 init\_material\_db() (in module xrayutilities.materials.database), 56, 89, 137  
 InputError, 20, 101  
 IntensityNormalizer (class in xrayutilities.normalize), 30, 111

## K

KeepData() (xrayutilities.gridder.Gridder method), 29, 110

keys() (xrayutilities.io.rotanode\_alignment.RA\_Alignment method), 47, 81, 128

## L

lam (xrayutilities.materials.material.Material attribute), 62, 95, 143

lam2en() (in module xrayutilities.utilities\_noconf), 32, 113

Lattice (class in xrayutilities.materials.lattice), 56, 90, 137

Lattice() (xrayutilities.materials.cif.CIFFile method), 54, 88, 135

lattice\_const\_AB() (xrayutilities.materials.material.Alloy method), 58, 92, 139

LatticeBase (class in xrayutilities.materials.lattice), 57, 91, 138

linear() (xrayutilities.experiment.QConversion method), 28, 109

linear\_detector\_calib() (in module xrayutilities.analysis.sample\_align), 42, 76, 123

Lorentz1d() (in module xrayutilities.math.functions), 63, 97, 144

Lorentz2d() (in module xrayutilities.math.functions), 63, 97, 144

## M

maplog() (in module xrayutilities.utilities), 32, 112

Material (class in xrayutilities.materials.material), 59, 93, 140

miscut\_calc() (in module xrayutilities.analysis.sample\_align), 43, 76, 124

mon (xrayutilities.normalize.IntensityNormalizer attribute), 31, 111

mu (xrayutilities.materials.material.Material attribute), 62, 95, 143

mycross() (in module xrayutilities.math.transforms), 65, 99, 146

## N

NaumanniteLattice() (in module xrayutilities.materials.lattice), 57, 91, 138

NonCOP (class in xrayutilities.experiment), 25, 106

Normalize() (xrayutilities.gridder.Gridder method), 29, 110

nu (xrayutilities.materials.material.Material attribute), 62, 95, 143

## O

Open() (xrayutilities.materials.database.DataBase method), 55, 89, 136

## P

Parse() (xrayutilities.io.rotanode\_alignment.RA\_Alignment method), 47, 81, 128

parse() (xrayutilities.io.seifert.SeifertMultiScan method), 48, 81, 129

parse() (xrayutilities.io.seifert.SeifertScan method), 48, 82, 129

Parse() (xrayutilities.io.spec.SPECFile method), 49, 83, 130

Parse() (xrayutilities.io.spec.SPECLog method), 49, 83, 130

Parse() (xrayutilities.materials.cif.CIFFile method), 54, 88, 135

PerkinElmer (class in xrayutilities.io.imagereader), 45, 79, 126

PerovskiteTypeRhombohedral() (in module xrayutilities.materials.lattice), 57, 91, 138

plot() (xrayutilities.io.rotanode\_alignment.RA\_Alignment method), 47, 81, 128

plot() (xrayutilities.io.spec.SPECScan method), 50, 84, 131

point() (xrayutilities.experiment.QConversion method), 29, 109

Powder (class in xrayutilities.experiment), 26, 106

PowderIntensity() (xrayutilities.experiment.Powder method), 26, 107

psd\_chdeg() (in module xrayutilities.analysis.sample\_align), 43, 77, 124

psd\_refl\_align() (in module xrayutilities.analysis.sample\_align), 44, 78, 125

PseudomorphicMaterial() (in module xrayutilities.materials.material), 62, 96, 143

## Q

Q() (xrayutilities.materials.material.Material method), 60, 93, 141

Q2Ang() (xrayutilities.experiment.Experiment method), 21, 101

Q2Ang() (xrayutilities.experiment.GID method), 22, 102

Q2Ang() (xrayutilities.experiment.GID\_ID10B method), 23, 103

Q2Ang() (xrayutilities.experiment.GISAXS method), 23, 104

Q2Ang() (xrayutilities.experiment.HXRD method), 24, 105

Q2Ang() (xrayutilities.experiment.NonCOP method), 25, 106

Q2Ang() (xrayutilities.experiment.Powder method), 26, 107

QConversion (class in xrayutilities.experiment), 26, 107

QuartzLattice() (in module xrayutilities.materials.lattice), 57, 91, 138

## R

RA\_Alignment (class in xrayutilities.io.rotanode\_alignment), 47, 81, 128

rad2hdf5() (in module xrayutilities.io.radicon), 46, 80, 127

Read() (xrayutilities.io.spectra.SPECTRAFile method), 51, 85, 132



- read\_data() (in module xrayutilities.io.spectra), 53, 87, 134
- read\_mca() (in module xrayutilities.io.spectra), 53, 87, 134
- read\_mca\_dir() (in module xrayutilities.io.spectra), 53, 87, 134
- read\_mcas() (in module xrayutilities.io.spectra), 53, 87, 134
- ReadData() (xrayutilities.io.edf.EDFFile method), 45, 78, 126
- ReadData() (xrayutilities.io.spec.SPECMCAFile method), 49, 83, 130
- ReadData() (xrayutilities.io.spec.SPECMCAInline method), 49, 83, 130
- ReadData() (xrayutilities.io.spec.SPECSScan method), 50, 83, 131
- readImage() (xrayutilities.io.imagereader.ImageReader method), 45, 79, 126
- ReadMCA() (xrayutilities.io.spectra.SPECTRAFile method), 51, 85, 132
- recarray2hdf5() (xrayutilities.io.spectra.Spectra method), 52, 86, 133
- ReciprocalLattice() (xrayutilities.materials.lattice.Lattice method), 57, 91, 138
- RelaxationTriangle() (xrayutilities.materials.material.Alloy method), 58, 91, 139
- repair\_key() (in module xrayutilities.io.seifert), 49, 82, 130
- RockSalt\_Cubic\_Lattice() (in module xrayutilities.materials.lattice), 57, 91, 138
- RockSaltLattice() (in module xrayutilities.materials.lattice), 57, 91, 138
- RoperCCD (class in xrayutilities.io.imagereader), 45, 79, 126
- rotarb() (in module xrayutilities.math.transforms), 65, 99, 146
- RutileLattice() (in module xrayutilities.materials.lattice), 57, 91, 138
- S**
- sampleAxis (xrayutilities.experiment.QConversion attribute), 29, 110
- Save2HDF5() (xrayutilities.io.edf.EDFDirectory method), 44, 78, 125
- Save2HDF5() (xrayutilities.io.edf.EDFFile method), 45, 78, 126
- Save2HDF5() (xrayutilities.io.spec.SPECCmdLine method), 49, 83, 130
- Save2HDF5() (xrayutilities.io.spec.SPECFile method), 49, 83, 130
- Save2HDF5() (xrayutilities.io.spec.SPECSScan method), 50, 84, 131
- Save2HDF5() (xrayutilities.io.spectra.SPECTRAFile method), 51, 85, 132
- save\_h5\_attrs() (xrayutilities.io.seifert.SeifertHeader method), 47, 81, 128
- SeifertHeader (class in xrayutilities.io.seifert), 47, 81, 128
- SeifertMultiScan (class in xrayutilities.io.seifert), 47, 81, 128
- SeifertScan (class in xrayutilities.io.seifert), 48, 81, 129
- selectst() (in module xrayutilities.io.radicon), 47, 80, 128
- set\_abs\_factors() (xrayutilities.io.spectra.Spectra method), 52, 86, 133
- SetChunkSize() (xrayutilities.gridder.Gridder method), 29, 110
- SetChunkUnit() (xrayutilities.gridder.Gridder method), 29, 110
- SetF0() (xrayutilities.materials.database.DataBase method), 55, 89, 136
- SetF1() (xrayutilities.materials.database.DataBase method), 55, 89, 136
- SetF2() (xrayutilities.materials.database.DataBase method), 55, 89, 136
- SetMaterial() (xrayutilities.materials.database.DataBase method), 55, 89, 136
- SetMCAParams() (xrayutilities.io.spec.SPECSScan method), 50, 84, 131
- SetResolution() (xrayutilities.gridder.Gridder2D method), 30, 110
- SetResolution() (xrayutilities.gridder.Gridder3D method), 30, 111
- SetThreads() (xrayutilities.gridder.Gridder method), 29, 110
- SetWeight() (xrayutilities.materials.database.DataBase method), 55, 89, 136
- SPECCmdLine (class in xrayutilities.io.spec), 49, 83, 130
- SPECFile (class in xrayutilities.io.spec), 49, 83, 130
- SPECLog (class in xrayutilities.io.spec), 49, 83, 130
- SPECMCA (class in xrayutilities.io.spec), 49, 83, 130
- SPECMCAFile (class in xrayutilities.io.spec), 49, 83, 130
- SPECMCAInline (class in xrayutilities.io.spec), 49, 83, 130
- SPECSScan (class in xrayutilities.io.spec), 50, 83, 131
- Spectra (class in xrayutilities.io.spectra), 52, 86, 133
- spectra2hdf5() (xrayutilities.io.spectra.Spectra method), 52, 86, 133
- SPECTRAFile (class in xrayutilities.io.spectra), 51, 85, 132
- SPECTRAFileComments (class in xrayutilities.io.spectra), 51, 85, 132
- SPECTRAFileData (class in xrayutilities.io.spectra), 52, 85, 133
- SPECTRAFileDataColumn (class in xrayutilities.io.spectra), 52, 85, 133
- SPECTRAFileParameters (class in xrayutilities.io.spectra), 52, 85, 133
- StructureFactor() (xrayutilities.materials.material.Material method), 60, 94, 141

StructureFactorForEnergy() (xrayutilities.materials.material.Material method), 60, 94, 141

StructureFactorForQ() (xrayutilities.materials.material.Material method), 60, 94, 141

SymStruct() (xrayutilities.materials.cif.CIFFile method), 54, 88, 135

## T

tensorprod() (in module xrayutilities.math.transforms), 65, 99, 146

TetragonalIndiumLattice() (in module xrayutilities.materials.lattice), 57, 91, 138

TetragonalTinLattice() (in module xrayutilities.materials.lattice), 57, 91, 138

TiltAngle() (xrayutilities.experiment.Experiment method), 21, 101

time (xrayutilities.normalize.IntensityNormalizer attribute), 31, 111

Transform (class in xrayutilities.math.transforms), 64, 98, 145

Transform() (xrayutilities.experiment.Experiment method), 21, 102

TrigonalR3mh() (in module xrayutilities.materials.lattice), 57, 91, 138

TwoGauss2d() (in module xrayutilities.math.functions), 64, 98, 145

## U

UB (xrayutilities.experiment.QConversion attribute), 26, 107

UnitCellVolume() (xrayutilities.materials.lattice.Lattice method), 57, 91, 138

Update() (xrayutilities.io.spec.SPECFile method), 49, 83, 130

Update() (xrayutilities.io.spec.SPECLog method), 49, 83, 130

## V

VecAngle() (in module xrayutilities.math.vector), 65, 99, 146

VecDot() (in module xrayutilities.math.vector), 65, 99, 146

VecNorm() (in module xrayutilities.math.vector), 66, 100, 147

VecUnit() (in module xrayutilities.math.vector), 66, 100, 147

## W

wavelength (xrayutilities.experiment.Experiment attribute), 21, 102

wavelength (xrayutilities.experiment.QConversion attribute), 29, 110

wavelength() (in module xrayutilities.utilities\_noconf), 32, 113

WurtziteLattice() (in module xrayutilities.materials.lattice), 57, 91, 138

## X

x (xrayutilities.materials.material.Alloy attribute), 58, 92, 139

xaxis (xrayutilities.gridder.Gridder1D attribute), 30, 110

xaxis (xrayutilities.gridder.Gridder2D attribute), 30, 110

xmatrix (xrayutilities.gridder.Gridder2D attribute), 30, 111

xrayutilities (module), 9, 20, 100

xrayutilities.analysis (module), 33, 66, 114

xrayutilities.analysis.line\_cuts (module), 33, 66, 114

xrayutilities.analysis.line\_cuts3d (module), 40, 73, 121

xrayutilities.analysis.misc (module), 41, 74, 122

xrayutilities.analysis.sample\_align (module), 41, 75, 122

xrayutilities.config (module), 20, 101

xrayutilities.exception (module), 20, 101

xrayutilities.experiment (module), 20, 101

xrayutilities.gridder (module), 29, 110

xrayutilities.io (module), 44, 78, 125

xrayutilities.io.edf (module), 44, 78, 125

xrayutilities.io.imagereader (module), 45, 79, 126

xrayutilities.io.panalytical\_xml (module), 46, 79, 127

xrayutilities.io.radicon (module), 46, 80, 127

xrayutilities.io.rotanode\_alignment (module), 47, 81, 128

xrayutilities.io.seifert (module), 47, 81, 128

xrayutilities.io.spec (module), 49, 82, 130

xrayutilities.io.spectra (module), 51, 85, 132

xrayutilities.libxrayutils (module), 30, 111

xrayutilities.materials (module), 54, 87, 135

xrayutilities.materials.\_create\_database (module), 54, 88, 135

xrayutilities.materials.\_create\_database\_alt (module), 54, 88, 135

xrayutilities.materials.cif (module), 54, 87, 135

xrayutilities.materials.database (module), 54, 88, 135

xrayutilities.materials.elements (module), 56, 90, 137

xrayutilities.materials.lattice (module), 56, 90, 137

xrayutilities.materials.material (module), 58, 91, 139

xrayutilities.math (module), 62, 96, 143

xrayutilities.math.fit (module), 62, 96, 143

xrayutilities.math.functions (module), 63, 97, 144

xrayutilities.math.transforms (module), 64, 98, 145

xrayutilities.math.vector (module), 65, 99, 146

xrayutilities.normalize (module), 30, 111

xrayutilities.utilities (module), 32, 112

xrayutilities.utilities\_noconf (module), 32, 113

XRDMLEFile (class in xrayutilities.io.panalytical\_xml), 46, 79, 127

XRDMLEMeasurement (class in xrayutilities.io.panalytical\_xml), 46, 79, 127

XRotation() (in module xrayutilities.math.transforms), 64, 98, 145

## Y

yaxis (`xrayutilities.gridder.Gridder2D` attribute), [30](#), [111](#)

ymatrix (`xrayutilities.gridder.Gridder2D` attribute), [30](#), [111](#)

YRotation() (in module `xrayutilities.math.transforms`), [64](#), [99](#), [145](#)

## Z

zaxis (`xrayutilities.gridder.Gridder3D` attribute), [30](#), [111](#)

ZincBlendeLattice() (in module `xrayutilities.materials.lattice`), [57](#), [91](#), [138](#)

zmatrix (`xrayutilities.gridder.Gridder3D` attribute), [30](#), [111](#)

ZRotation() (in module `xrayutilities.math.transforms`), [65](#), [99](#), [146](#)