islatu Documentation

Release 1.0.7

Andrew R. McCluskey

Aug 07, 2023

CONTENTS

`islatu` is an open-source pacakge for the reduction of x-ray reflectometry datasets. Currently, `islatu` is developed at and supports data from Diamond Light Source, however we are happy to work with others to enable data from other sources (including neutron sources).

These webpages include API-level documentation and information about some workflows that can be used for data reduction. There is also documentation on a *command line interface* that can be used to process reflectivity data without any python programming.

CHAPTER

ONE

CONTRIBUTING

As with any coding project, there are many ways to contribue. To report a bug or suggest a feature, open an issue on the github repository. If you would like to contribute code, we would recommend that you first raise an issue before diving into writing code, so we can let you know if we are working on something similar already. To e.g. fix typos in documentation or in the code, or for other minor changes, feel free to make pull requests directly.

CHAPTER

TWO

CONTACT US

If you need to contact the developers about anything, please either raise an issue on the github repository if appropriate, or send an email to richard.brearton@diamond.ac.uk.

CHAPTER

THREE

CONTRIBUTORS

- Richard Brearton
- Andrew R. McCluskey

CHAPTER

FOUR

ACKNOWLEDGEMENTS

We acknowledge the support of the Ada Lovelace Centre – a joint initiative between the Science and Technology Facilities Council (as part of UK Research and Innovation), Diamond Light Source, and the UK Atomic Energy Authority, in the development of this software.

4.1 Installation

`islatu` can be installed from the PyPI package manager with `pip`:

```
pip install islatu
```

Alternatively, the latest development build can be found Github.

4.2 Reduction workflows

A typical data reduction workflow handled by `islatu` is shown here:

4.2.1 I07 reflectometry data

Detailed here are the techniques one can use to reduce reflectometry data acquired on the I07 beamline at diamond using the islatu package. Both the vertical scattering and the double-crystal deflector geometries are discussed. The particular example of reduction of reflectometry data collected from a water-air interface is worked through in detail.

At I07, a reflectometry profile is generally constructed from multiple scans. Each scan has a corresponding nexus file (.nxs) which contains metadata, and an hdf5 file (.h5) containing the images recorded in the scan.

The first task is to instantiate an instance of islatu's Profile class containing the reflectivity profile's data. To do this, we need to download the profile's constituent .h5 and .nxs files, and tell islatu where to find the .nxs files. Provided that your .h5 files are in the same directory as your .nxs files, islatu will be able to load your data.

[45]:
```
from islatu.refl_profile import Profile
from islatu.io import i07_nxs_parser

# On my machine, I have some .nxs and .h5 files in this directory.
data_dir = "../../tests/resources/"
# I have scans 404875-404882 stored in the above directory.
scan_numbers = range(404875, 404882 + 1)
```

(continues on next page)

(continued from previous page)

```
data_files = [f"{data_dir}i07-{i}.nxs" for i in scan_numbers]

# Profile.fromfilenames() expects a list of paths to data files, and a parser
# from islatu.io that it can use to parse the data files.
my_profile = Profile.fromfilenames(data_files, i07_nxs_parser)
```

```
Data file found at ../../tests/resources/excaliburScan404875_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404875_000001.h5
Currently loaded 51 images.
Data file found at ../../tests/resources/excaliburScan404876_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404876_000001.h5
Currently loaded 8 images.
Data file found at ../../tests/resources/excaliburScan404877_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404877_000001.h5
Currently loaded 8 images.
Data file found at ../../tests/resources/excaliburScan404878_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404878_000001.h5
Currently loaded 9 images.
Data file found at ../../tests/resources/excaliburScan404879_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404879_000001.h5
Currently loaded 13 images.
Data file found at ../../tests/resources/excaliburScan404880_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404880_000001.h5
Currently loaded 14 images.
Data file found at ../../tests/resources/excaliburScan404881_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404881_000001.h5
Currently loaded 31 images.
Data file found at ../../tests/resources/excaliburScan404882_000001.h5.
Loading images from file ../../tests/resources/excaliburScan404882_000001.h5
Currently loaded 26 images.
```

When we used the fromfilenames class method above, islatu loaded all of the images taken in the scan. It's now possible to use the profile to get a very crude indication of what our data looks like. This is possible because islatu uses the default signal and default axis specified in the nexus file to guess a profile's reflectivity as a function of Q or .

[46]:
```
# Note that any instantiated islatu profile has the following properties:
print(f"Theta values: {my_profile.theta}")
print(f"Corresponding q-vectors: {my_profile.q_vectors}")
print(f"Reflectivity: {my_profile.reflectivity}")
print(f"Errors on the reflectivity: {my_profile.reflectivity_e}")
```

```
Theta values: [0.045225 0.047505 0.049755 ... 3.525275 3.570413 3.615548]
Corresponding q-vectors: [0.01     0.010504 0.011002 ... 0.779022 0.788984 0.798945]
Reflectivity: [0.793168 0.80557  0.817332 ... 0.040644 0.040322 0.039764]
Errors on the reflectivity: [0.042844 0.043178 0.043492 ... 0.009698 0.00966  0.009593]
```

Let's plot this!

[47]:
```
import plotly.graph_objects as go

# The following two lines are just required for the generation of this document.
import plotly
from plotly.offline import iplot
```

(continues on next page)

(continued from previous page)

```
plotly.offline.init_notebook_mode()

def plot_xrr_curve(islatu_profile: Profile, title: str, log=True):
    """
    Convenience function for plotting simple XRR curves.
    """
    fig = go.Figure().update_layout(title=title,
                                    xaxis_title='Q/Å', yaxis_title='R')
    fig.add_trace(go.Scatter(x=(islatu_profile.q_vectors),
                             y=(islatu_profile.reflectivity), error_y={
        "type": 'data',
        "array": (islatu_profile.reflectivity_e),
        "visible": True},
        name="Islatu"))

    if log:
        fig.update_yaxes(type="log")

    iplot(fig)
```

[48]:
```
title = "Uncorrected profile"
plot_xrr_curve(my_profile, title)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

The above curve is pretty much nonsense, the sole exception being that the x-axis is correct. Before trying to perform any corrections, it is worth understanding more about the profile object that islatu has created for us.

Firstly, just as the real experiment was made up from a series of scans, so is the islatu profile object.

[49]:
```
print(my_profile.scans)
```

```
[<islatu.scan.Scan2D object at 0x137fc7e50>, <islatu.scan.Scan2D object at 0x147767820>,
↪<islatu.scan.Scan2D object at 0x137fde1f0>, <islatu.scan.Scan2D object at 0x147562d30>,
↪ <islatu.scan.Scan2D object at 0x147838be0>, <islatu.scan.Scan2D object at 0x147510430>
↪, <islatu.scan.Scan2D object at 0x1472aedc0>, <islatu.scan.Scan2D object at␣
↪0x147734e50>]
```

Each of these scans has its own q_vectors, reflectivities, and reflectivities_e, just as the overall profile does.

[50]:
```
first_scan = my_profile.scans[0]

print(f"Theta values: {first_scan.theta}")
print(f"Corresponding q-vectors: {first_scan.q_vectors}")
print(f"Reflectivity: {first_scan.reflectivity}")
print(f"Errors on the reflectivity: {first_scan.reflectivity_e}")
```

```
Theta values: [0.045225 0.047505 0.049755 ... 0.15377  0.156033 0.158291]
Corresponding q-vectors: [0.01     0.010504 0.011002 ... 0.034002 0.034502 0.035002]
```

(continues on next page)

(continued from previous page)

```
Reflectivity: [0.940042 0.95474  0.96868  ... 0.018563 0.017533 0.016332]
Errors on the reflectivity: [0.050778 0.051173 0.051545 ... 0.007135 0.006935 0.006693]
```

Scans also have an attribute called metadata. This contains all of the metadata stored in that scan's .nxs file. Islatu uses this object to expose several useful pieces of information, such as background and signal regions of interest, the distance between the detector and the sample, and much more (shown below).

[51]:
```
example_metadata = first_scan.metadata

print([attr for attr in dir(example_metadata) if not attr.startswith('_')])
```

```
['background_regions', 'default_axis', 'default_axis_name', 'default_axis_type',
↪'default_nxdata', 'default_nxdata_name', 'default_signal', 'default_signal_name',
↪'detector', 'detector_distance', 'detector_name', 'entry', 'excalibur_detector',
↪'instrument', 'local_data_path', 'local_path', 'nxfile', 'probe_energy', 'signal_
↪regions', 'src_path', 'transmission']
```

Each of the scans in a profile contain detector images. Let's visualize the first image taken in the first scan.

[52]:
```
# Image data is stored as a numpy array, so we'll want to load numpy to
# manipulate the data.
import numpy as np

first_image = first_scan.images[0]

# Because of how the heatmap will be displayed, to make x pixels lie along the
# x-axis we need to take the transpose of our data.
array_transpose = np.transpose(first_image.array)
# Let's also take the log of our data to make it easier to visualize.
array_log_transpose = np.log(array_transpose+0.1)
```

[53]:
```
# We'll want a simple function for plotting heatmaps.
def plot_heatmap(array_to_plot, title):
    """
    Simple function that we can use to plot heatmaps of detector frames.
    """
    fig = go.Figure().update_layout(title=title,
                                    xaxis_title='x-pixels',
                                    yaxis_title='y-pixels')
    fig.add_trace(go.Heatmap(z=array_to_plot, colorscale='Jet'))
    iplot(fig)
```

[54]:
```
# Now plot the image's array.
plot_heatmap(array_log_transpose, "Uncropped detector frame")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

As we can see, the signal is localized in the detector. There is not much point carrying out corrections on the whole image, so we should crop our images around the signal. While we could use the above image to manually specify a region of interest, this is usually given by the first region of interest specified in GDA for experiments on I07. So we can carry out the cropping rather neatly and automatically as follows.

[55]:
```
from islatu.cropping import crop_to_region

signal_region, = example_metadata.signal_regions
my_profile.crop(crop_to_region, region=signal_region)
```

[56]:
```
# Now lets re-visualize what we have!

array_transpose = np.transpose(first_image.array)
plot_heatmap(array_transpose, "Cropped detector image")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Clearly, the above images will be more efficient to work with. Now it is time to start applying corrections to our reflectivity curve.

The first substantial improvement to the reflectivity and its errors can be made by subtracting background. This replaces the not-so-meaningful default signal stored in the .nxs file with a proper background-subtracted integrated region of interest. Two ways of subtracting background using islatu are shown below.

[57]:
```
%%capture
from islatu.background import roi_subtraction
from islatu.region import Region

# Islatu refers to regions of interest using instances of its Region class. For
# these data, I happen to know that a reasonable estimate of the background can
# be made by looking at counts in the following region.
background_region = Region(x_start=1258, x_end=1308, y_start=206, y_end=224)

# Now, background can be subtracted as follows.
my_profile.bkg_sub(roi_subtraction, list_of_regions=[background_region])

# Typically, during reflectometry experiments the I07 beamline, one specifies a
# number of regions of interest in GDA. While the first region usually
# specifies the signal region, other regions of interest indicate background
# regions. If this was done during your experiment, then background can be
# subtracted without hardcoding a background region by uncommenting the
# following line.

# my_profile.bkg_sub(roi_subtraction,
#     list_of_regions=example_metadata.background_regions)
```

Now let's see how this affected our profile's reflectivity curve.

[58]:
```
title = "Background subtracted profile"
plot_xrr_curve(my_profile, title)
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

The first thing to notice is that the uncertainties on each point have substantially decreased. Now that the background has been subtracted, each data point is much more physically meaningful and uncertainties have been correctly calculated

(replacing the nexus file defaults). These uncertainties might seem shockingly small, but the simple fact of the matter is that statistical errors in synchrotron measurements are tiny!

To continue, lets make this look a little more… connected. The reason the XRR curve is so bumpy at the moment is that the beam attenuation is changing with scan number. Correcting for attenuation variation between scans will help a lot!

The beam attenuation is stored in the .nxs file that we used to load our profile, so islatu already knows about the attenuation as a function of scan number. All we need to do is tell islatu to correct for it, and we're done.

[59]:
```
my_profile.transmission_normalisation()
```

[60]:
```
# Let's visualize the profile now!
plot_xrr_curve(my_profile, "Transmission + bkg corrected")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

This already looks far better. We can flatten the curve above the total external reflection point by applying a footprint correction. Footprint corrections account for the fact that, at small Q, only a small fraction of the beam is incident on the sample surface, which affects reflected intensity. To correct for this, all we need to do is specify the beam FWHM and the sample size.

As mentioned earlier, this is a water sample. In I07, the trough is 20cm long. All distances should be input as SI units.

[61]:
```
beam_FWHM = 100e-6
sample_size = 200e-3

my_profile.footprint_correction(beam_width=beam_FWHM, sample_size=sample_size)
```

[62]:
```
# Now plot the resultant curve!
plot_xrr_curve(my_profile, title="Footprint + transmission + bkg corrected")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Next, we should correct for the variation in the intensity of the incident beam when using I07's double-crystal deflector (DCD) setup. If you didn't use the DCD in your experiment, don't worry – simply ignore this step.

When using the DCD, a normalisation file is acquired. You will need to have downloaded this along with your data. This is usually a .dat file. To carry out the DCD normalisation, islatu will need to generate an interpolator from your .dat file.

[63]:
```
from islatu.corrections import get_interpolator
from islatu.io import i07_dat_to_dict_dataframe

path_to_DCD_dat_file = data_dir + "404863.dat"

itp = get_interpolator(path_to_DCD_dat_file, i07_dat_to_dict_dataframe)
my_profile.qdcd_normalisation(itp)
```

[64]:
```
# Now check out the plot after correcting for variation in the intensity of the
# incident beam in the DCD setup.
```

(continues on next page)

(continued from previous page)

```
plot_xrr_curve(my_profile, "DCD + footprint + transmission + bkg")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

Finally, there are regions of this curve in which data points from different scans are overlappping. Propagating errors correctly, these data can be combined in a continuum of different ways. Islatu can convert this into a smooth reflectivity curve by rebinning your data into linearly (or, optionally, logarithmically) separated bins in q-space, combining data points by an inverse-variance weighted mean (the optimal way of combining data).

[65]:
```
number_of_lin_spaced_qs = 4000
my_profile.rebin(number_of_q_vectors=number_of_lin_spaced_qs)
```

[66]:
```
# Now visualize the fully corrected curve!
plot_xrr_curve(my_profile, "Fully corrected reflectivity profile")
```

Data type cannot be displayed: application/vnd.plotly.v1+json, text/html

4.3 The command line interface

4.3.1 Introduction

This page contains some simple instructions on how to use the process_xrr CLI.

Typing `process_xrr.py -h` should give an overview of how it can be used.

4.3.2 Creating a .yaml file

Before starting, you should create a .yaml file that contains some details pertaining to your experiment.

An example of the .yaml file could be something like

```
instrument: 'i07'
visit:
  # ...you should probably fill in your actual details here!
  local contact: "Richard Brearton"
  user: 'Richard Brearton'
  user affiliation: 'Diamond Light Source, University of Oxford'
  visit id: 'si28707-1/'
  # YYYY-MM-DD
  date: 2021-08-06
setup:
  # (sample_length, sample_width) in m
  # ...where the "length" direction is parallel to the wavevector of the
  # incident light for |q|=0.
  sample size: (200e-3, 10e-3)
  # Beam FWHM in m
```

(continues on next page)

(continued from previous page)

```
  beam width: 100e-6
  # /path/to/normalization/file
  # Outside of diamond, this might look like, for example:
  # /Users/richardbrearton/Documents/Data/si28707-1/404863.dat
  dcd normalisation: /dls/i07/data/2021/si28707-1/404863.dat
crop:
  method: crop
  # Leave kwargs commented to crop to ROI_1, as specified in GDA.
  # Uncomment kwargs to crop to manually set the cropping region.
  # kwargs: {'x_start': 1050, 'x_end': 1150, 'y_start': 190, 'y_end': 211}
background:
  # The most reliable method that one can use to subtract background is
  # roi_subtraction. We strongly recommend that this option is used.
  method: roi_subtraction
rebin:
  # Number of bins to place q-vectors into. These bins are linearly spaced in q
  # by default.
  n qvectors: 5000
```

If scattering without the DCD in an ordinary vertical geometry, just get rid of the dcd normalization field, eg:

```
instrument: 'i07'
visit:
  # ...you should probably fill in your actual details here!
  local contact: "Richard Brearton"
  user: 'Richard Brearton'
  user affiliation: 'Diamond Light Source, University of Oxford'
  visit id: 'si28707-1/'
  # YYYY-MM-DD
  # Islatu is actually also using this to work out where your data is stored
  # under the hood. This should be set to the date the data was recorded, and
  # this will be written to your final .dat file.
  date: 2021-08-06
setup:
  # (sample_length, sample_width) in m
  # ...where the "length" direction is parallel to the wavevector of the
  # incident light for |q|=0.
  sample size: (10e-3, 10e-3)
  # Beam FWHM in m
  beam width: 100e-6
crop:
  method: crop
  # Leave kwargs commented to crop to ROI_1, as specified in GDA.
  # Uncomment kwargs to crop to manually set the cropping region.
  # kwargs: {'x_start': 1050, 'x_end': 1150, 'y_start': 190, 'y_end': 211}
background:
  # The most reliable method that one can use to subtract background is
  # roi_subtraction. We strongly recommend that this option is used.
  method: roi_subtraction
rebin:
  # Number of bins to place q-vectors into. These bins are linearly spaced in q
  # by default.
```

(continues on next page)

(continued from previous page)

```
n qvectors: 5000
```

4.3.3 Processing with process_xrr.py

So, you have your data files stored in /path/…/to/data/ , your .yaml file at /path/…/to/yaml_file.yaml and you want your processed output to live in /path/…/to/xrr_curve.dat . If the directory /path/…/to/data/ contains exclusively the scans used to construct your reflectivity curve, then this reflectivity curve can be constructed using:

process_xrr.py -d /path/…/to/data/ -y /path/…/to/yaml_file.yaml -o /path/…/to/xrr_curve.dat

So, after writing -d you need to tell process_xrr where to find your data, after writing -y you need to tell process_xrr where to find your .yaml file, and after writing -o you need to tell process_xrr where to save your data.

If you don't want to come up with a new name for the final processed output each time, then don't worry: the islatu package will generate a name for your .dat file at a directory of your choosing. To do this, simply give the path to a directory as opposed to a file to the -o option. Explicitly, this would look like:

```
process_xrr.py -d /path/.../to/data/ -y /path/.../to/yaml_file.yaml -o /path/.../to/
output_directory/
```

Then your output XRR curve will live in /path/…/to/output_directory/generated_name.dat . These names are generated from your .yaml file and your scan numbers, so will be unique for different analyses.

When processing in diamond, often reflectivity profiles will be constructed from consecutive scans whose scan numbers vary incrementally from a minimum number to a maximum number. By default, these will all be stored in one big directory. The above examples would only work if your directory contained exclusively the scans from which your profile will be constructed, which is clearly not the case here.

So, if your scan numbers of interest start at lower_number and end at upper_number and are stored in a directory /path/…/to/data/ that contains many other scan numbers, then processing can be carried out using

```
process_xrr.py -d /path/.../to/data/ -y /path/.../to/yaml_file.yaml -o /path/.../to/
output_directory/ -l lower_number -u upper_number
```

Practical example 1

Imagine my account name is xrr12345, so ~ aliases to /home/xrr12345. In my documents folder, I have a folder called ~/Documents/Recipes/ for .yaml recipes. I have another folder called ~/Documents/Data/ for reduced XRR curves. I'm interested in some data collected in 2021 using the DCD setup in the experiment si28979-1, and my data is stored in the experiment's root directory on the diamond filesystem. My DCD normalization .dat file is stored in /dls/i07/data/2021/si28979-1/817213.dat , and my XRR curve is constructed from scans number 817220-817229 inclusive.

To begin, I add the above DCD template .yaml file to my recipes folder and name it DCD_si28979_1.yaml (note that this name is completely up to you and has no practical consequences).

Now that I have a generic .yaml file where I want it, I open it up with my favourite text editor and fill out my personal and experimental details. Most importantly, my DCD normalization line reads

dcd normalisation: /dls/i07/data/2021/si28979-1/817213.dat

If this field is not filled out correctly, the Islatu package will raise an error, but it should be reasonably uncomplicated to work out what went wrong! Now, to produce my corrected XRR curve I write in a terminal:

```
process_xrr.py -d /dls/i07/data/2021/si28979-1/ -y /home/xrr12345/Documents/Recipes/
DCD_si28979_1.yaml -o /home/xrr12345/Documents/Data/ -l 817220 -u 817229
```

Practical example 2

Now, later on in the same experiment you want to process another reflectivity curve, with numbers between 817241 - 817251. But, acquisition of this profile was not so smooth, and scan numbers 817246 and 817249 should not be included in the final XRR profile. In situations like this, where profiles need to be constructed from custom lists of scan numbers, process_xrr can be run as follows:

```
process_xrr.py -d /dls/i07/data/2021/si28979-1/ -y /home/xrr12345/Documents/Recipes/
DCD_si28979_1.yaml -o /home/xrr12345/Documents/Data/ -N 817241 817242 817243 817244
817245 817247 817248 817250 817251
```

4.4 API

4.4.1 islatu.background

Background substraction is a necessary component of reflectometry reduction, where the background scattering is removed from the reflected intensity.

Herein are some functions to enable that for a two-dimensional detector image, as well as simple dataclasses in which we can store some information relating to the background subtraction, and any fitting that we might have carried out.

class islatu.background.**BkgSubInfo**(*bkg: float*, *bkg_e: float*, *bkg_sub_function: Callable*, *fit_info: Optional[FitInfo] = None*)

Bases: object

A simple data class in which we can store information relating to a background subtraction.

bkg: float

bkg_e: float

bkg_sub_function: Callable

fit_info: *FitInfo* = None

class islatu.background.**FitInfo**(*popt: ndarray*, *pcov: ndarray*, *fit_function: Callable*)

Bases: object

A simple dataclass in which we can store data relating to the quality of a fit.

fit_function: Callable

pcov: ndarray

popt: ndarray

islatu.background.**fit_gaussian_1d**(*image: Image*, *params_0=None*, *bounds=None*, *axis=0*)

Fit a one-dimensional Gaussian function with some ordinate offset to an image with uncertainty. This is achieved by averaging in a given axis before performing the fit. Return the results, and index of the offset.

Parameters

- **image** – The islatu image object to fit.
- **params_0** (list, optional) – An initial guess at the parameters. Defaults to values based on the image.
- **bounds** (list of tuple, optional) – Bounds for the fitting. Defaults to values based on the image.

- **axis** (int) – The dimension along which the averaging will be performed.

Returns

Containing:

- **array_like: The results (with uncertainties) for each** of the 6 parameters fit.
- int: The index of the offset.
- **None: As it is not possible to describe the reflected** peak width.

Return type
tuple

islatu.background.**roi_subtraction**(*image*, *list_of_regions: List[*Region*]*)

Carry out background subtraction by taking a series of rectangular regions of interested (ROIs) as being fair Poissonian measurements of the background.

Parameters

- **image** – The islatu.image.Image object from which we should subtract background from.
- **list_of_regions** – A list of instances of the Regions class corresponding to background regions.

islatu.background.**univariate_normal**(*data*, *mean*, *sigma*, *offset*, *factor*)

Produce a univariate normal distribution.

Parameters

- **data** (array_like) – Abscissa data.
- **mean** (float) – Mean (horizontal).
- **sigma** (float) – Variance (horizontal).
- **offset** (float) – Offset from the 0 for the ordinate, this is the background level.
- **factor** (float) – Multiplicative factor for area of normal distribution.

Returns
Ordinate data for univariate normal distribution.

Return type
array_like

4.4.2 islatu.corrections

Reflectometry data must be corrected as a part of reduction. These functions facilitate this, including the footprint and DCD q-variance corrections.

islatu.corrections.**footprint_correction**(*beam_width*, *sample_size*, *theta*)

The factor by which the intensity should be multiplied to account for the scattering geometry, where the beam is Gaussian in shape.

Parameters

- **beam_width** (float) – Width of incident beam, in metres.
- **sample_size** (float) – Width of sample in the dimension of the beam, in metres.

- **theta** (float) – Incident angle, in degrees.

Returns
Array of correction factors.

islatu.corrections.**get_interpolator**(*file_path*, *parser*, *q_axis_name='qdcd_'*, *intensity_axis_name='adc2'*)

Get an interpolator object from scipy, this is useful for the DCD q-normalisation step.

Parameters

- **file_path** (str) – File path to the normalisation file.
- **parser** (callable) – Parser function for the normalisation file.
- **q_axis_name** (str, optional) – Label for the q-value in the normalisation file. Defaults to 'qdcd_'.
- **intensity_axis_name** (str, optional) – Label for the intensity in the normalisation file. Defaults to 'adc2'.

Returns

Containing:

- array_like: Interpolation knots.
- array_like: B-spline coefficients.
- int: Degree of spline.

Return type
tuple

4.4.3 islatu.cropping

Often the detector is a lot larger than the reflected intensity peak, so it makes sense to crop the image to the peak.

islatu.cropping.**crop_to_region**(*array: ndarray*, *region:* Region)

Crops the input array to the input region.

Parameters

- **array** – The array to crop.
- **region** – The instance of Region to crop to.

4.4.4 islatu.data

This module contains both the Data class and the MeasurementBase class. In a reflectometry measurement, the experimental data corresponds to the reflected intensity as a function of scattering vector Q. In a typical diffractometer, Q is a virtual axis, calculated geometrically from various motor positions. The Data class takes care of these conversions, exposing q, theta, intensity, reflectivity, and energy.

The MeasurementBase class defines a simple class that is Data, but that also has metadata.

class islatu.data.**Data**(*intensity*, *intensity_e*, *energy*, *theta=None*, *q_vectors=None*)

Bases: object

The base class of all Islatu objects that contain data.

intensity

A numpy array containing intensities in this dataset.

intensity_e

A numpy array containing the corresponding errors in intensity.

theta

A numpy array containing the probe particle's angle of incidence at each intensity.

q_vectors

A numpy array containing the magnitude of the probe particle's scattering vector for each intensity value.

energy

The energy of the probe particle used to acquire this data. This is necessary to swap between theta and q.

Parameters

- **intensity** – A numpy array of the intensities in this dataset.
- **intensity_e** – The errors on the intensities.
- **energy** – The energy of the probe particle used to acquire this data.
- **theta** – A numpy array containing the probe particle's angle of incidence at each intensity. NOTE: only one of theta/q needs to be provided.
- **q_vectors** – A numpy array containing the magnitude of the probe particle's scattering vector for each intensity value. NOTE: only one of theta/q needs to be provided.

property q_vectors: array

Returns self._q if this instance of Data was generated from q-data. Otherwise, converts from self._theta to q.

property reflectivity: array

Returns the intensity, normalized such that the maximum value of the intensity is equal to 1. To acquire

property reflectivity_e: array

Returns the errors on the intensity, divided by the maximum value of the intensity array.

remove_data_points(*indices*)

Convenience method for the removal of a specific data point by its index.

Parameters
indices – The indices to be removed.

property theta: array

Returns self._theta if this instance of Data was generate from th-data. Otherwise, converts from scattered q to theta.

class islatu.data.**MeasurementBase**(*intensity*, *intensity_e*, *energy*, *metadata*, *theta=None*, *q=None*)

Bases: *Data*

All measurements derive from this class.

Attrs:

metadata:
The metadata relevant to this measurement.

4.4.5 islatu.debug

Islatu's simple Debug class.

class islatu.debug.**Debug**(*logging_level*)

Bases: object

A simple logger.

Attrs:

logging_level:
Current logging level. Higher means more unimportant messages will be shown.

log(*log_string*, *unimportance: int = 1*, ***kwargs*)

Prints to stdout if self.logging_level >= unimportance.

Parameters

- **log_string** – The string to be printed.
- **unimportance** – A measure of unimportance assigned to the printing of this string. Very unimportant messages require a larger logging level to be printed. Defaults to 1.

4.4.6 islatu.image

The two-dimension detector generates images of the reflected intensity. The purpose of the Image class stored in this module is the investigation and manipulation of these images.

class islatu.image.**Image**(*array: ndarray*, *transpose: bool = False*)

Bases: object

This class stores information about the detector images.

file_path

File path for the image.

Type
str

array

The image described as an array.

Type
array_like

array_original

The original value of the image array when it was loaded from disk.

Type
array_like

array_e

The errors on each pixel of the array.

Type
array_like

bkg

The background that was subtracted from the image.

Type
float

bkg_e

The uncertainty on the background.

Type
float

Parameters

- **file_path** (str) – The file path for the image.
- **data** (pandas.DataFrame, optional) – Experimental data about the measurement. Defaults to None.
- **transpose** (bool, optional) – Should the data be rotated by 90 degrees? Defaults to False.

background_subtraction(*background_subtraction_function*, ***kwargs*)

Perform a background subtraction based on some function.

Parameters

- **background_subtraction_function** (callable) – The function to model the data and therefore remove the background.
- ****kwargs** (dict) – The background substraction function keyword arguments.

crop(*crop_function*, ***kwargs*)

Perform an image crop based on some function.

Parameters

- **crop_function** (callable) – The function to crop the data.
- ****kwargs** (dict) – The crop function keyword arguments.

property initial_std_devs

Get the standard deviation values of the original raw image array.

Returns
Standard deviation values of image.

Return type
array_like

property nominal_values

Get the nominal values of the image array.

Returns
Nominal values of image.

Return type
array_like

property shape

Array shape

Returns
The shape of the image.

Return type
tuple of int

sum()

Perform a summation on the image's array.

Returns
A tuple taking the form (summed_intensity, summed_intensity_e).

4.4.7 islatu.io

This module contains:

Parsing functions used to extract information from experimental files.

Classes used to help make parsing more modular. These include the NexusBase class and its children.

class islatu.io.**I07Nexus**(*local_path: str*)

Bases: *NexusBase*

This class extends NexusBase with methods useful for scraping information from nexus files produced at the I07 beamline at Diamond.

property background_regions: List[*Region*]

Returns a list of region objects that define the location of background. Currently we just ignore the zeroth region and call the rest of them background regions.

property default_axis_name: str

Returns the name of the default axis.

property default_axis_type: str

Returns the type of our default axis, either being 'q', 'th' or 'tth'.

property detector_distance

Returns the distance between sample and detector.

property detector_name: str

Returns the name of the detector that we're using. Because life sucks, this is a function of time.

excalibur_04_2022 = 'exr'

excalibur_detector_2021 = 'excroi'

property local_data_path: str

The local path to the data (.h5) file. Note that this isn't in the NexusBase class because it need not be reasonably expected to point at a .h5 file.

Raises
FileNotFoundError if the data file cant be found. –

property probe_energy

Returns the energy of the probe particle parsed from this NexusFile.

property signal_regions: List[*Region*]

Returns a list of region objects that define the location of the signal. Currently there is nothing better to do than assume that this is a list of length 1.

property transmission

Proportional to the fraction of probe particles allowed by an attenuator to strike the sample.

class islatu.io.**NexusBase**(*local_path: str*)

Bases: *Metadata*

This class contains *mostly* beamline agnostic nexus parsing convenience stuff. It's worth noting that this class still makes a series of assumptions about how data is laid out in a nexus file that can be broken. Instead of striving for some impossible perfection, this class is practical in its assumptions of how data is laid out in a .nxs file, and will raise if an assumption is violated. All instrument-specific assumptions that one must inevitably make to extract truly meaningful information from a nexus file are made in children of this class.

Attrs:

file_path:
The local path to the file on the local filesystem.

nxfile:
The object produced by loading the file at file_path with nxload.

property default_axis: ndarray

Returns the nxdata associated with the default axis.

property default_axis_name: str

Returns the name of the default axis.

abstract property default_axis_type: str

Returns what type of default axis we have. Options are 'q', 'th' or 'tth'.

property default_nxdata: ndarray

Returns the default NXdata.

property default_nxdata_name

Returns the name of the default nxdata.

property default_signal: ndarray

The numpy array of intensities pointed to by the signal attribute in the nexus file.

property default_signal_name

Returns the name of the default signal.

property detector

Returns the NXdetector instance stored in this NexusFile.

Raises
ValueError if more than one NXdetector is found. –

property entry: NXentry

Returns this nexusfile's entry.

Raises
ValueError if more than one entry is found. –

property instrument

Returns the NXinstrument instanced stored in this NexusFile.

Raises
ValueError if more than one NXinstrument is found. –

property src_path

The name of this nexus file, as it was recorded when the nexus file was written.

islatu.io.**i07_dat_to_dict_dataframe**(*file_path*)

Parses a .dat file recorded by I07, returning a [now mostly obsolete] tuple containing a metadata dictionary and a pandas dataframe of the data.

Though outdated, this is still a handy way to parse the DCD normalization .dat file.

:param (str): The .dat file to be read.

Returns

Containing:

- dict: The metadata from the .dat file.
- pandas.DataFrame: The data from the .dat file.

Return type
tuple

islatu.io.**i07_nxs_parser**(*file_path: str*)

Parses a .nxs file acquired from the I07 beamline at diamond, returning an instance of Scan2D. This process involves loading the images contained in the .h5 file pointed at by the .nxs file, as well as retrieving the metadata from the .nxs file that is relevant for XRR reduction.

Parameters
file_path – Path to the .nxs file.

Returns
An initialized Scan2D object containing all loaded detector frames, as well as the relevant metadata from the .nxs file.

islatu.io.**load_images_from_h5**(*h5_file_path*, *transpose=False*)

Loads images from a .h5 file.

Parameters

- **h5_file_path** – Path to the h5 file from which we're loading images.
- **transpose** – Should we take the transpose of these images? Defaults to True.

4.4.8 islatu.metadata

This module contains the Metadata class, returned by parser methods in the islatu.io module. This class provides a consistent way to refer to metadata returned by different detectors/instruments, and also contains a dictionary of all of the metadata as scraped from the parsed file.

class islatu.metadata.**Metadata**(*local_path*)

Bases: object

An ABC for classes that store metadata parsed from data files. This defines the properties that must be implemented by parsing classes.

abstract property default_axis: ndarray

Returns a numpy array of data associated with the default axis, where "default axis" should be understood in the NeXus sense to mean the experiment's dependent variable.

abstract property default_axis_name: str

Returns the name of the default axis, as it was recorded in the data file stored at local_path.

abstract property default_axis_type: str

Returns what type of default axis we have. Options are ‘q’, ‘th’ or ‘tth’.

abstract property detector_distance

Returns the distance between sample and detector.

abstract property probe_energy

This must be overridden.

abstract property transmission

Proportional to the fraction of probe particles allowed by an attenuator to strike the sample.

4.4.9 islatu.refl_profile

A profile is a measurement resulting from a scan, or a series of scans. Profiles are the central objects in the islatu library, containing the total reflected intensity as a function of scattering vector data.

class islatu.refl_profile.**Profile**(*data:* Data, *scans: List[*Scan*]*)

Bases: *Data*

The object that is used to store all information relating to a reflectivity profile.

bkg_sub(*bkg_sub_function*, ***kwargs*)

Class method for the bkg_sub() method for each Scan in the list.

Parameters

- **bkg_sub_function** (callable) – Background subtraction function to be used.
- **kwargs** (dict, optional) – Keyword arguments for the background subtraction function. Defaults to None.

concatenate()

Class method for *concatenate()*.

crop(*crop_function*, ***kwargs*)

Calls the Class method for the *crop()* method for each Scan2D in self.scans.

Parameters

- **crop_function** (callable) – Cropping function to be used.
- **kwargs** (dict, optional) – Keyword arguments for the cropping function. Defaults to None.

footprint_correction(*beam_width*, *sample_size*)

Class method for footprint_correction() for each Scan in the list.

Parameters

- **beam_width** (float) – Width of incident beam, in metres.
- **sample_size** (uncertainties.core.Variable) – Width of sample in the dimension of the beam, in metres.
- **theta** (float) – Incident angle, in degrees.

classmethod fromfilenames(*filenames*, *parser*)

Instantiate a profile from a list of scan filenames.

Parameters

- **filenames** (list) – List of files, one for each reflectometry scan. Can have length one.
- **parser** (callable) – Parser function for the reflectometry scan files.

qdcd_normalisation(*itp*)

Class method for qdcd_normalisation() for each Scan in the list.

Parameters
normalisation_file (str) – The .dat file that contains the normalisation data.

rebin(*new_q=None*, *rebin_as='linear'*, *number_of_q_vectors=5000*)

Class method for *islatu.stitching.rebin()*.

Parameters

- **new_q** (array_like) – Array of potential q-values. Defaults to None. If this argument is not specified, then the new q, R values are binned according to rebin_as and number_of_q_vectors.
- **rebin_as** (py:attr:*str*) – String specifying how the data should be rebinned. Options are "linear" and "log". This is only used if the new_q are unspecified.
- **number_of_q_vectors** (int, optional) – The max number of q-vectors to be using initially in the rebinning of the data. Defaults to 400.

subsample_q(*scan_identifier*, *q_min=0*, *q_max=inf*)

For the scan with identifier scan_identifier, delete all data points for which q < q_min or q > q_max.

Parameters

- **scan_identifier** – The scan ID of the scan to be subsampled. This must be a unique substring of the filename from which the scan was taken. For example, if a scan's nexus filename is i07-413244.nxs, then a valid scan_ID would be "413244", as this string will uniquely identify the correct scan from within the profile.
- **q_min** – The smallest acceptable value of q. Defaults to 0 Å.
- **q_max** – The largest acceptable value of q. Defaults to inf Å.

transmission_normalisation()

Perform the transmission correction.

4.4.10 islatu.region

This module defines the Region object, whose instances define regions of interest in images.

class islatu.region.**Region**(*x_start*, *x_end*, *y_start*, *y_end*)

Bases: object

Instances of this class define regions of interest.

classmethod from_dict(*region_dict: dict*)

Instantiates a Region from a dictionary with keys in:
['x', 'y', 'width', 'height'].

This is to help loading dictionarys that are generated by calling json.loads on the NXcollections found in I07 nexus files as of 27/04/2022.

property num_pixels

returns the number of pixels in the region.

property x_length

Returns the length of the region in the x-direction.

property y_length

Returns the length of the region in the y-direction.

4.4.11 islatu.scan

This module contains the Scan and Scan2D classes. A Scan is a measurement and so inherits from MeasurementBase. An instance of Scan contains scan metadata, as well as a suite of methods useful for data correction, uncertainty calculations and the like.

A Scan2D is a Scan whose Data object's intensity values are computed from an image captured by an area detector. Many of Scan's methods are overloaded to make use of the additional information provided by the area detector, and extra image manipulation methods are included in Scan2D.

class islatu.scan.**Scan**(*data:* Data, *metadata:* Metadata)

Bases: *MeasurementBase*

A class used to store reflectometry scans taken with a point detector.

footprint_correction(*beam_width*, *sample_size*)

Class method for *islatu.corrections.footprint_correction()*.

Parameters

- **beam_width** (float) – Width of incident beam, in metres.
- **sample_size** (uncertainties.core.Variable) – Width of sample in the dimension of the beam, in metres.
- **theta** (float) – Incident angle, in degrees.

qdcd_normalisation(*itp*)

Perform normalisation by DCD variance.

Parameters

itp (tuple) – Containing interpolation knots (array_like), B-spline coefficients (array_like), and degree of spline (int).

subsample_q(*q_min=0*, *q_max=inf*)

Delete data points less than q_min and more than q_max.

Parameters

- **q_min** – The minimum q to be included in this scan. Defaults to 0 Å.
- **q_max** – The maximum q to be included in this scan. Defaults to inf Å.

transmission_normalisation()

Perform the transmission correction.

class islatu.scan.**Scan2D**(*data:* Data, *metadata:* Metadata, *images: List[*Image*]*)

Bases: *Scan*

data

The intensity as a function of Q data for this scan.

Type

islatu.data.Data

metadata

This scan's metadata.

Type
islatu.metadata.Metadata

images

The detector images in the given scan.

Type
list of *islatu.image.Image*

bkg_sub(*bkg_sub_function*, ***kwargs*)

Perform background substraction for each image in a Scan.

Parameters

- **bkg_sub_function** (callable) – Background subtraction function to be used.
- **kwargs** (dict, optional) – Keyword arguments for the background subtraction function. Defaults to None.
- **progress** (bool, optional) – Show a progress bar. Requires the tqdm package. Defaults to True.

crop(*crop_function*, ***kwargs*)

Crop every image in images according to crop_function.

Parameters

- **crop_function** (callable) – Cropping function to be used.
- **kwargs** (dict, optional) – Keyword arguments for the cropping function. Defaults to None.
- **progress** (bool, optional) – Show a progress bar. Requires the tqdm package. Defaults to True.

remove_data_points(*indices*)

Convenience method for the removal of specific data points by their indices.

Parameters
indices – The indices to be removed.

4.4.12 islatu.stitching

As reflectometry measurements typically consist of multiple scans at different attenutation, we must stitch these together.

islatu.stitching.**concatenate**(*scan_list: List[*Scan*]*)

Concatenate each of the datasets together.

Parameters
scans – List of reflectometry scans.

Returns

Containing:

- q-values.
- Reflected intensities.

– Errors on reflected intensities.

Return type
`tuple`

islatu.stitching.**rebin**(*q_vectors*, *reflected_intensity*, *new_q=None*, *rebin_as='linear'*, *number_of_q_vectors=5000*)

Rebin the data on a linear or logarithmic q-scale.

Parameters

- **q_vectors** – q - the current q vectors.
- **reflected_intensity** (`tuple`) – (I, I_e) - The current reflected intensities, and their errors.
- **new_q** (`array_like`) – Array of potential q-values. Defaults to `None`. If this argument is not specified, then the new q, R values are binned according to rebin_as and number_of_q_vectors.
- **rebin_as** (py:attr:*str*) – String specifying how the data should be rebinned. Options are "linear" and "log". This is only used if the new_q are unspecified.
- **number_of_q_vectors** (`int`, optional) – The max number of q-vectors to be using initially in the rebinning of the data. Defaults to `400`.

Returns

Containing:

- q: rebinned q-values.
- intensity: rebinned intensities.
- intensity_e: rebinned intensity errors.

Return type
`tuple`

CHAPTER

FIVE

SEARCHING

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

L

M

N

P

Q

R

S

T

U

X

Y