

# PyXScat: User Manual

edgar.gutierrez-fernandez@esrf.fr

EDGAR GUTIERREZ-FERNANDEZ<sup>a,b</sup> AND OIER BIKONDOA<sup>a,b\*</sup>

<sup>a</sup>*Department of Physics. University of Warwick. Gibbet Hill Road. Coventry, CV4 7AL UK, and* <sup>b</sup>*XMaS, The UK-CRG Beamline. ESRF - The European Synchrotron. CS40220. F-38043 Grenoble cedex 09. France. E-mail: oier.bikondoa@esrf.fr*

## 1. Introduction

This document is committed to address in detail the implemented tools in PyXScat. While it is not necessary for the user to fully read it, it has been written as a step-by-step workflow from the first searching of files to the most advanced functionalities. We recommend the user to look over every section during the first usage of the program.

## 2. Initialize Data Repository

We call data repository a *pythonic* instance that contains enough information to handle every data file whose name matches with a specific pattern (e.g. \*.edf), within all the sub-directories of a root directory. For that reason, PyXScat needs only a directory address and a file name pattern. All data and metadata information is saved in a .h5 file, that PyXscat creates, handles and updates. There are two ways to initialize a data repository data in PyXScat:

- ***Picking a Root Directory***: the user chooses a root directory with the Picking tool. Then, a first searching will be run, reading all the files inside the sub-directories of the root whose name matches with the file name pattern, modified by the user through the ***Extension*** and ***Wildcards*** tools.

Once the searching is over, the repository contains the addresses of every located file and the metadata contained inside the files. The data/metadata of these files are read through the FabIO module [1].

- ***Importing/Opening an .h5 File***: if the repository and the .h5 file has been already created, it can be imported directly through the picking tool or clicking it from the list of recently opened .h5 files.

It is important to understand that the .h5 file stores only the addresses of data files. Therefore any change in the local path of the files will require the creation of a new .h5 file.

The needed tools to initiate the data repository are contained in the same tab (*Setup tab*), as displayed in Figure 1.

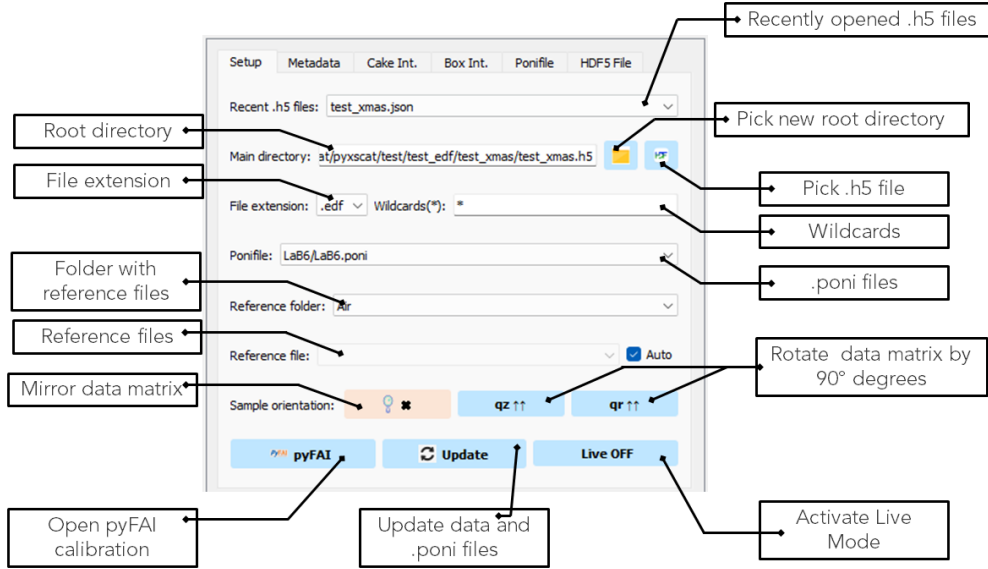


Fig. 1. Setup Tab with the available tools.

### 3. Live data visualization

PyXScat contains live searching engines, so the file repository can be automatically updated if a new data file that matches the pattern pops up in one of the sub-directories. The Live mode can be activated just by clicking on its respective button (Figure 1)

Live mode is currently available only in Linux machines (version 1.0).

### 4. Browsing Folders, Files and Metadata

Once the .h5 has been created or imported, the list of sub directories with data files will be updated under the input tool tab.

- **List of Folders:** a compilation of all detected sub-directories with data files.

By clicking on one sample name, the name of the files and values of metadata are displayed in the table widget.

- **Table of Files-Metadata:** the names of every data file detected in the clicked folder are organized in rows in this table. PyXScat allows to add/remove columns with the metadata associated to the listed Files.
- **Metadata:** PyXScat lists all the detected metadata names of the files contained in the clicked sample, that can be activated to display in the Table widget (Figure ).

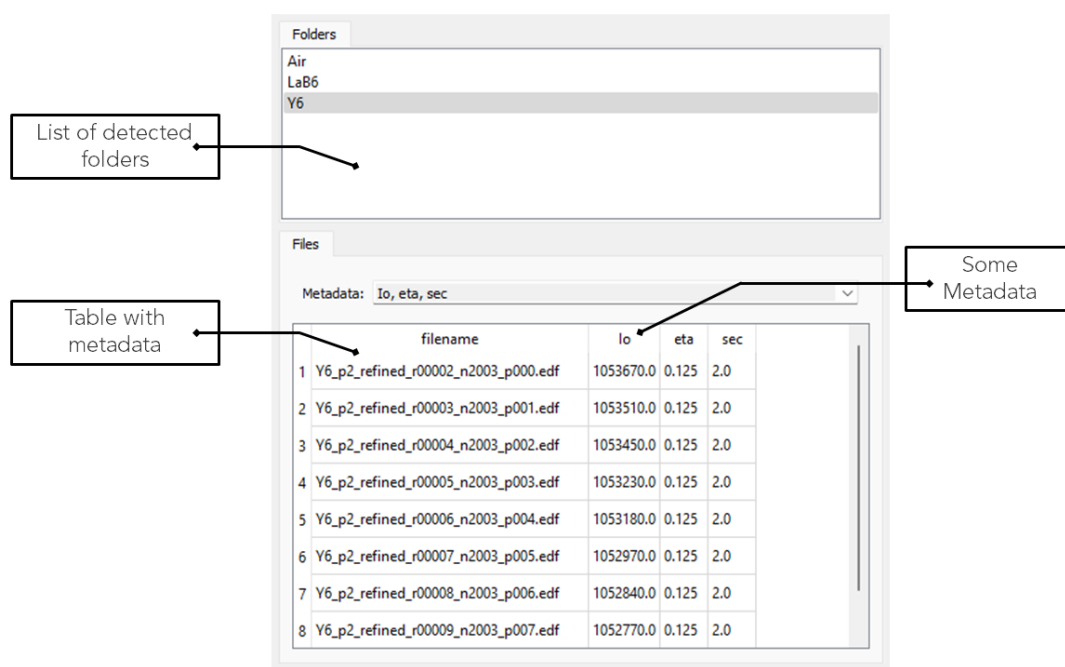


Fig. 2. List of Folders and Table with Metadata.

*Metadata* is every value associated to the collected data that is not the data itself. Metadata includes, for example, time and date values, size of the matrix, motor positions, counters for normalization factors, acquisition time, etc. They are usually saved in the form of *key-value* pairs in the file header.

#### 4.1. Important Metadata Pointers

Metadata is not mandatory to visualize and integrate the 2D maps. However, there are some counters and motors that has to be explicitly declared by the user to use some functionalities of PyXScat:

- **Acquisition time ( $s$ )**: needed to subtract the reference scattering.
- **Normalization factor**: needed to subtract the reference scattering.
- **Incident angle ( $^\circ$ )**: important for grazing-incidence (GI) corrections.
- **Tilt angle ( $^\circ$ )**: extra corrections for GI geometry.

The name of these keys can be modified through the Metadata Keys Tab (Figure 3):

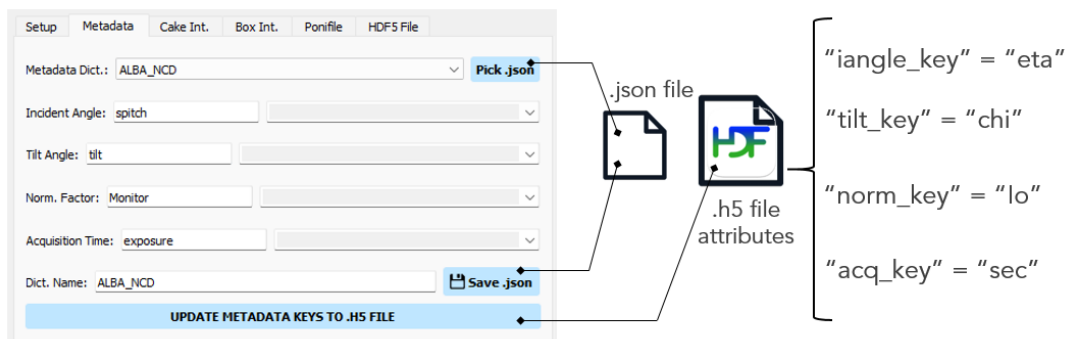


Fig. 3. Metadata Tab to declare the name of important metadata counters.

There should be a name for the set of metadata keys. This value can be the name of a specific beamline, which uses their own metadata keys. It is possible also to save the set of keys as a .json file using the save button. To make the changes effective, the user has to press the button **Update Metadata Keys**.

These keys are allowed to be empty. Without a key for acquisition time, the background subtraction tool won't work in automatic mode, and without keys for incident and tilt angles, they will setup both as 0.0.

## 5. Visualizing 2D Graphs

The 2D graph widget with the plain data array is updated by clicking on the different elements of the Table widget, with no further information to be added (Figure 4.a).

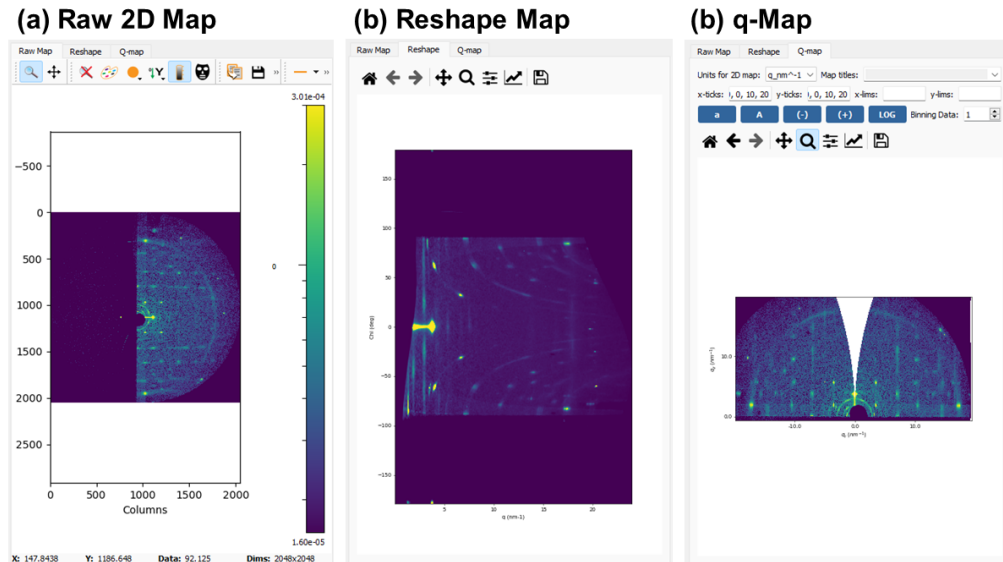


Fig. 4. Metadata Tab to declare the name of important metadata counters.

PyXScat further allows to perform automatically some calculations to plot more physically meaningful 2D maps (Figure 4.b-c). At this point, PyXScat works as an abstraction of pyFAI [2] and pygix [3] modules, the last one based on the former plus grazing-incidence functionalities. To make the shown transformations, PyXScat needs to handle a *.poni*.

### 5.1. *.poni* file

A *.poni* file is a text file with all the instrumentation parameters required to transform the raw pixel map into angular units, whether exit angle or scattering vector modulus. PONI stands for Point-Of-Normal-Incidence: the two metric-coordinates

from the detector origin to the intersection between the detector plane and the line whose vector is the detector plane normal and the point is the sample. Detailed information is in the pyFAI documentation [2]. The information stored in the *.poni* file allows to rearrange the pixels of the intensity array into an azimuth-radius scattering map. This transformation can be visualized in PyXScat clicking on the "Reshape" tab (Figure 5).

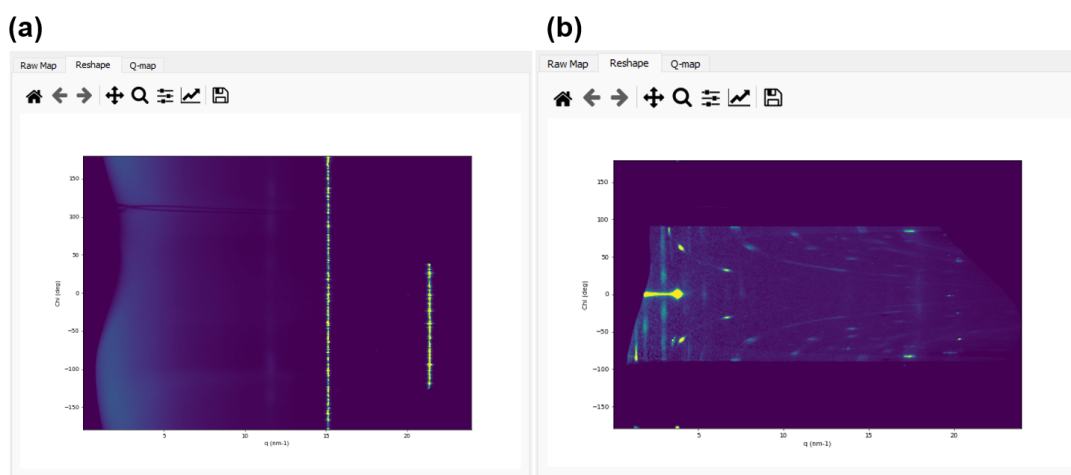


Fig. 5. Reshaped pixel map into a chi(angular)-q map, from a (a) LaB6 powder sample and a (b) organic, thin film sample.

As shown in Figure 5.a, the Debye-Scherrer rings coming from a powder sample are rearranged in the form of rods in an azimuth-radial map. On the other hand (Figure 5.b), if the scattering pattern presents diffraction rods oriented along the vertical axis of the reciprocal space, they will be bended due to the curvature of the scattering sphere.

A *.poni* file is generated after identification of the Debye-Scherrer rings coming from a standard sample, i.e. AgBh or LaB6. The two rings from LaB6 in Figure 5 was used to calibrate a GIWAXS experiment. This calibration step is done through the *pyFAI-calib* graphical interface, which can be accessed through the Setup tab

(Figure 1).

After importing a root directory or an .h5 file, PyXScat searches recursively for .poni files inside the root directory and stores them in the corresponding combobox (Figure 1).

Once the .poni file is activated through one of these methods, we can visualize the calibration parameters in the tab “*Ponifile*” (Figure 6).

Setup Metadata Cake Int. Box Int. **Ponifile**

☐ Modify .poni parameters

Wavelength (m): 1.0332016536100021e-10

Distance (m): 0.22173828837628898

Detector Rayonix SX165 / (2, 2) / (4096, 4096) / (7.9e-05, 7.9e-05)

PONI 1 (m): 0.08997736445825115

PONI 2 (m): 0.08497882142263533

Rots (rads) (1): 0.05020438176140168

Rots (rads) (2): -0.003787516958040375

Rots (rads) (3): 0.0

RETRIEVE UPDATE SAVE

Fig. 6. Tab with .poni file parameters used during a GIWAXS experiment, developed at BM28-XMaS beamline.

These parameters can be modified by the user if required. A new set of parameters can be updated and saved in another file, with a time-date prefix. The original .poni file won't be modified in any case.

The usual way to represent the 2D pattern is after transformation whether to exit angles (vertical angle vs horizontal angle) or, in the case of grazing-incidence geometry, to scattering vector components ( $q_{xy}$  vs  $q_z$ ). PyXScat, via pygix module, allows to do both transformations. This tab (Figure 7) contains several tools to allow the user to modify each stylistic element of the map before saving the figure as a .png file.



- **Units:**  $q(\text{nm}^{-1})$ ,  $q(\text{\AA}^{-1})$ ,  $2\theta(^{\circ})$ ,  $2\theta(\text{rads})$ .
- **Metadata titles:** the output map may contain metadata information as a figure title.
- **Axes ticks:** while the limits of the figure can be configured through the matplotlib toolbar, the ticks are modified through their corresponding line-edits.
- **Log-scaling:** Change the color normalization between linear and logarithmic.
- **Font-size:** two buttons to reduce or increase the fonts of the figure.
- **Scatter size:** PyXScat uses the scatter method to display the transformed map. This means that, upon the size of the figure, it may be empty gaps between the scatter point. The two buttons (-) and (+) reduces and increases the size of the scatter points.
- **Binning parameter:** the 2D arrays collected in synchrotron experiments could be extremely large, and the scatter method to transform the map is a bottleneck in computing speed. The solution is to apply a binning factor to reduce the resolution of the array.

A visual example of how to use the q-map toolbar is shown in Figure 7. By default, the map is binned to 4 (Figure 7.a), so the original size of the array has been reduced by a factor of 16. That's why we see so many empty spaces between scattered points. While we are modifying the style of the map, like the font-size (Figure 7.b), it is time-efficient to use a binning factor. Once we are happy with the style, we can fill the gaps by increasing the size of the point while keeping low resolution (Figure 7.c), but of course, the idea is to use the original resolution for the final figure (Figure 7.d), although it could be demanding.

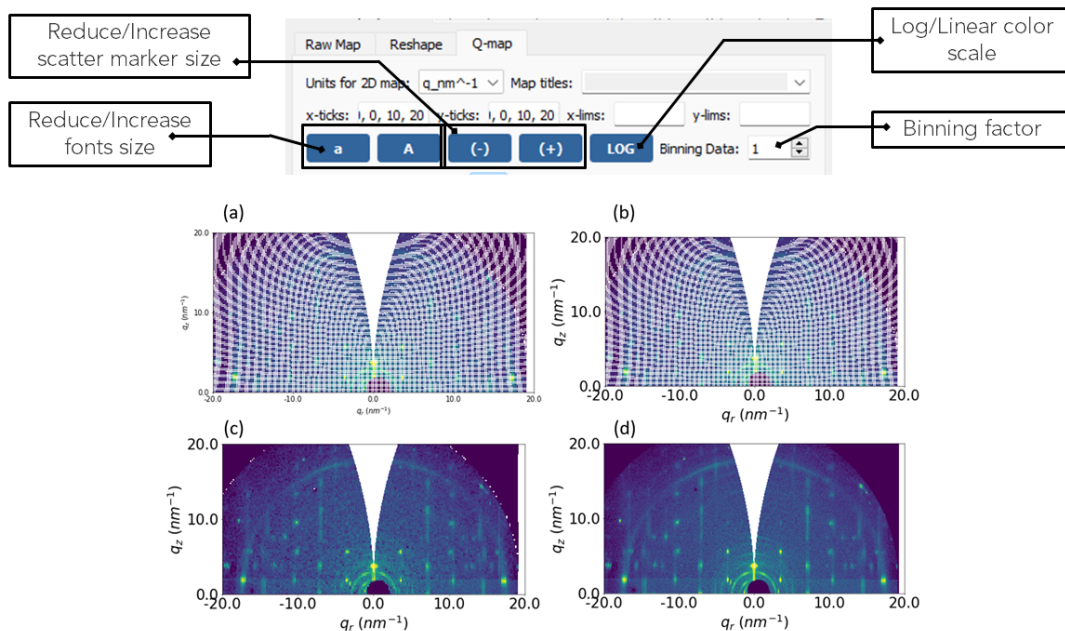


Fig. 7. Up: toolbar to modify the q-map style. Bottom: GIWAXS q-map from an organic thin film: (a) using a binning of 4 and small font-size, (b) after increasing the font-size, (c) after increasing the size of the scatter point, (d) after restoring the original array size (binning=1).

## 6. Pattern Integration

To integrate the pattern into intensity profiles, we need, apart from the .poni file, a set of integration parameters that will determine the region of the pattern to be integrated, according to certain conditions. PyXScat provides two tabs to consult/add/modify the integration parameters: one for "Cake" integrations and the other one for "Box" integrations (Figure 8).

Figure 8 shows two screenshots of software tabs for modifying integration parameters.   
 (a) Cake integration tab: The 'Setup' tab is active. Fields include Name: complete, Suffix: complete, Type: Azimuthal, Units: q\_nm^-1, Radial range: 0.00 to 22.00, Azimuthal range: -180.00 to 180.00, and Azimuthal bins (optional). A list on the right shows 'complete', 'ip', and 'oop'.   
 (b) Box integration tab: The 'Setup' tab is active. Fields include Name: central\_rod, Suffix: central\_rod, Direction: Vertical, Input units: q\_nm^-1, In-plane range: -2.00 to 2.00, Out-of-plane range: 0.00 to 15.00, and Output units: q\_nm^-1. A list on the right shows 'central\_rod'.

Fig. 8. Tabs to modify the integration parameters for (a) Cake integration, and (b) Box integration.

Both Cake and Box integrations are based on the pygix module [3]. A Cake integration consists on slicing the original pattern in a double-arc shape, which is, taking horizontal or vertical slices of the polar reshaped map we showed in Figure 5. A horizontal slide (averaging the angular axis) is called "Azimuthal Integration", and it results on a intensity profile as a function of q modulus:

$$q = \sqrt{q_x^2 + q_y^2 + q_z^2} \quad (1)$$

In contrast, after a "Radial Integration" we average the q-axis so the intensity profile will be a function of azimuthal angle (chi). For both cases, the user has to fill the following parameters into the Cake Integration Tab:

- **Cake Integration Tab:**

- *Name of Integration*: string to identify the set of parameters.
- *Suffix*: short name to identify the set of parameters.
- *Type*: Azimuthal / Radial.
- *Units*:  $q(\text{nm}^{-1})$ ,  $q(\text{\AA}^{-1})$ ,  $2\theta(^{\circ})$ ,  $2\theta(\text{rads})$ .
- *Radial range*: 14 - 22  $\text{nm}^{-1}$  (example).
- *Azimuthal range*: 0 - 90° (example).

- *Azimuth bins*: 90 (example, optional for azimuthal type).

Examples of both Azimuthal and Radial integrations using the same parameters are shown in Figure 9.

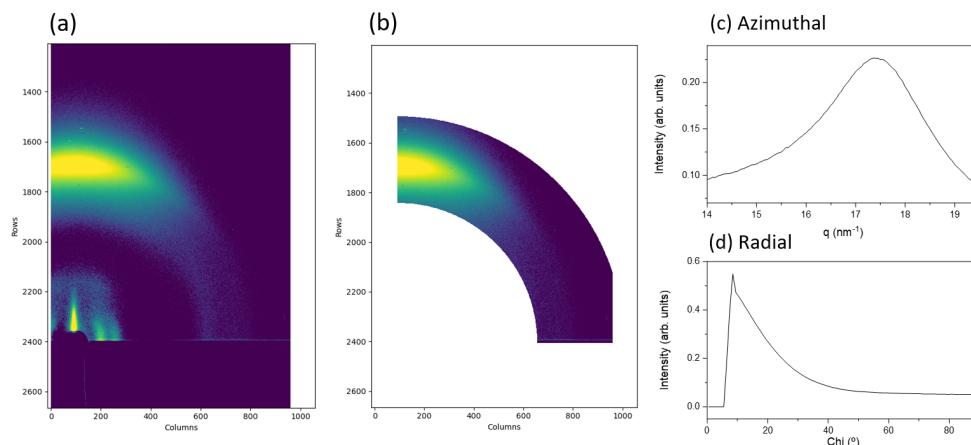


Fig. 9. Example of a Cake Integration, using 14-22  $\text{nm}^{-1}$  as radial range and 0-90° degrees as azimuthal range. (a) Raw GIWAXS pattern of an organic thin film, (b) Masked array with the mentioned integration parameters. (c) Azimuthal type integration, (d) radial type integration.

On the other hand, a "Box Integration" could be useful for GIWAXS measurements. As we anticipated in section 2.4, a  $q_{xy} - q_z$  representation follows a fiber symmetry, which is usually true for thin film samples deposited by spin-coating. The "Box" can be seen as a rectangular slice of the reciprocal space, projected into the scattering sphere. The integration parameters are accessible in the "Box Integration" tab:

• **Box Integration Tab:**

- *Name of Integration*: string to identify the set of parameters.
- *Suffix*: short name to identify the set of parameters.
- *Direction*: Horizontal / Vertical.
- *Input units*:  $q(\text{nm}^{-1})$ ,  $q(\text{\AA}^{-1})$ ,  $2\theta(^{\circ})$ ,  $2\theta(\text{rads})$ .
- *In-plane range*: 3 - 5  $\text{nm}^{-1}$  (example).
- *Out-of-plane range*: 0 - 20  $\text{nm}^{-1}$  (example).

- *Output units:*  $q(\text{nm}^{-1})$ ,  $q(\text{\AA}^{-1})$ ,  $2\theta(^{\circ})$ ,  $2\theta(\text{rads})$ .

Examples of Vertical and Horizontal integrations using the same parameters are shown in Figure 10.

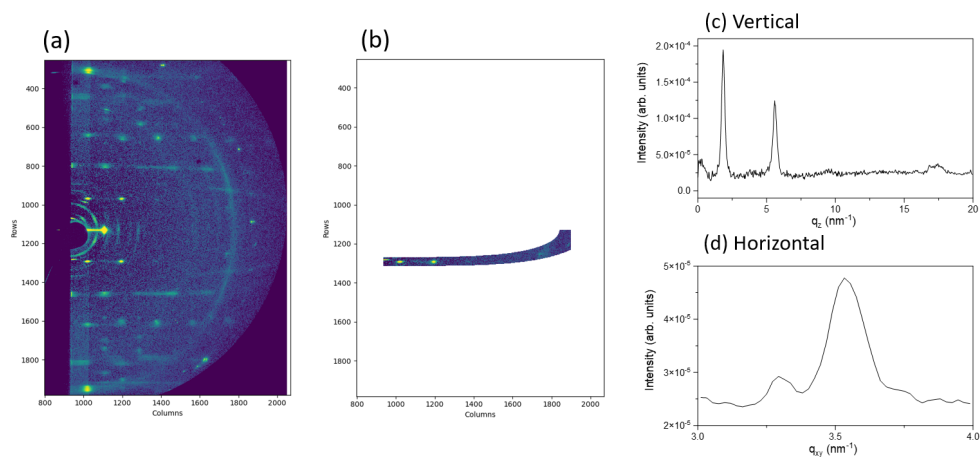


Fig. 10. Example of a Box Integration, using  $3\text{--}5\text{ nm}^{-1}$  as in-plane range and  $0\text{--}20\text{ nm}^{-1}$  as out-of-plane range. (a) Raw GIWAXS pattern of an organic thin film, (b) Masked array with the mentioned integration parameters. (c) Vertical direction integration, (d) horizontal direction integration..

Once the name of the integration is defined, a *.json* file with the name and integration parameters is automatically saved. Every change on these parameters updates the corresponding *.json* file. Then, several integrations protocols can be performed and displayed automatically (Figure 11):

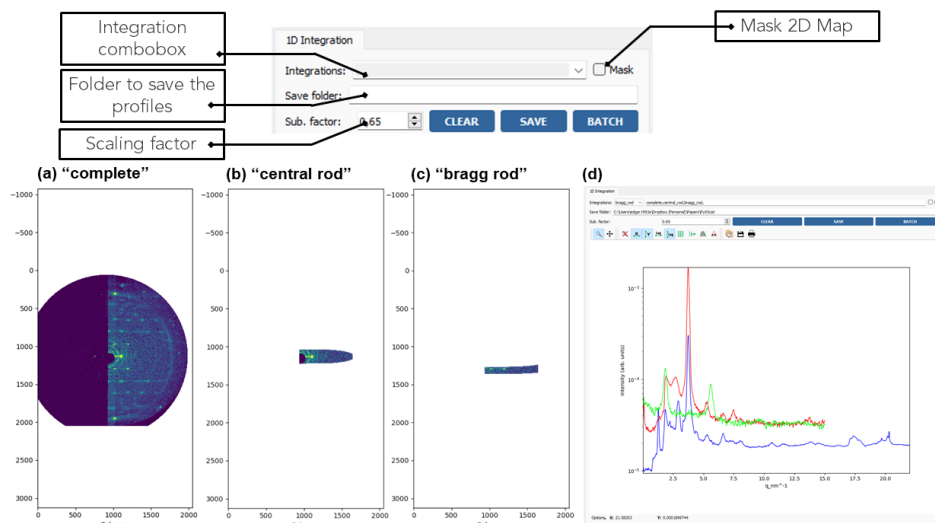


Fig. 11. Upper: toolbar from 1D Integration tab. Bottom: (a-c) Examples of masked patterns by different set of integration parameters. (d) Representation of the three different intensity profile..

The 1D plot widget contains toolbars to add, remove and save intensity profiles:

- **Integration Combobox**: every set of integration parameters saved as .json can be add and remove using these two tools.
- **Mask 2D Map Checkbox**: takes the first integration name from the combobox and masks the 2D pattern according to corresponding integration parameters.
- **Save Folder**: allows the user to write an output path to save the intensity profiles. If empty, the data will be saved in the same folder as the image.
- **Save Data**: saves all the data plotted in the widget with some metadata.
- **Batch Integration**: runs a batch integration and saves the results for all the files contained in the activated sub-directory.
- **Clear**: removes the displayed data.
- **Subtraction Factor**: modify the scaling factor used to subtract a reference pattern. This tool is fully described in the next section.

## 7. Background Subtraction

PyXScat allows to visualize the sample pattern after automatic subtraction of a reference pattern multiplied by an scaling factor. The tool is only performing if the scaling factor is different from default 0.0, and this factor should be below 1, since the sample should absorb more light than a reference sample. In our example, the reference is the scattering from the air, i.e. without any sample (Figure 12).

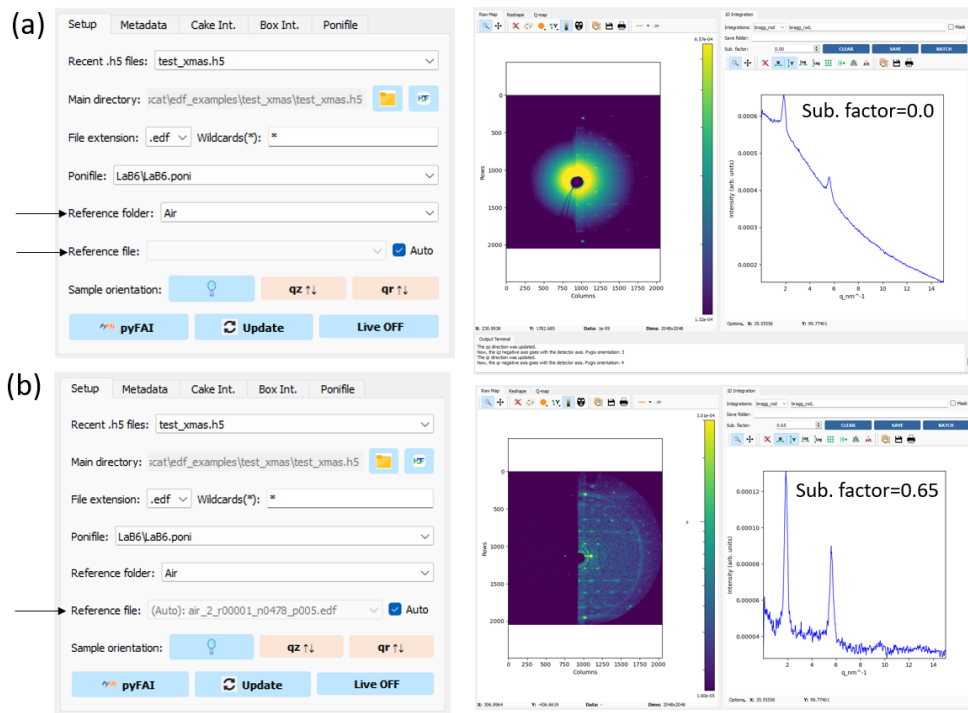


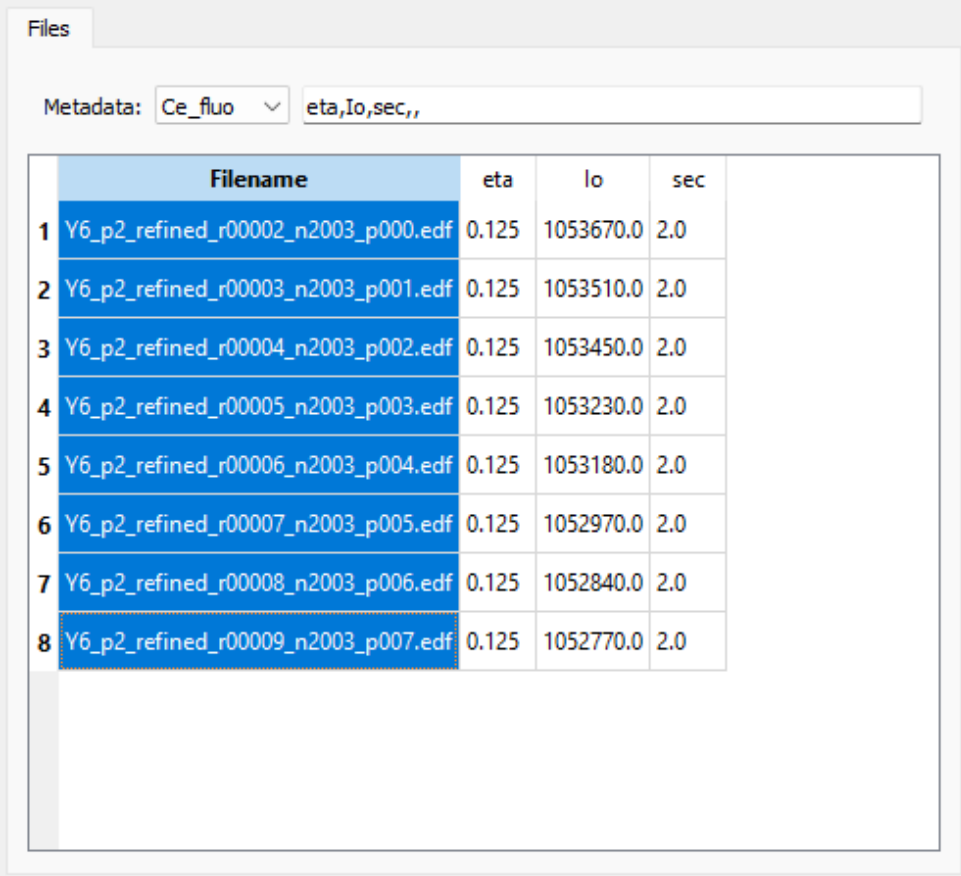
Fig. 12. Example of plotting 2D pattern and integration: (a) without any reference subtraction, (b) after subtracting the air scattering pattern multiplied by a scale factor=0.65.

First, the user has to choose the sub-directory (**Reference Folder Combobox** in Figure 12) where reference patterns are stored. By default, the automatic mode would use the first reference pattern whose acquisition time matches with the one from the sample. That's why PyXScat accepts a metadata key for acquisition time (see section 3.4). If there is no registered key, the user can un-check the **Auto Checkbox** and

choose manually a **Reference File** that's inside the reference sub-directory.

## 8. Frame-Averaging

If the user clicks on multiple items from different files on the Table Widget (Figure 13), PyXScat performs a step of averaging all the arrays belonging to the different files. Consequently, the 2D map, reshape map, integration profiles and q-transformed map would benefit from this averaging step, which, if coming from equivalent measurements, will increase the signal-to-noise ratio.



	Filename	eta	lo	sec
1	Y6_p2_refined_r00002_n2003_p000.edf	0.125	1053670.0	2.0
2	Y6_p2_refined_r00003_n2003_p001.edf	0.125	1053510.0	2.0
3	Y6_p2_refined_r00004_n2003_p002.edf	0.125	1053450.0	2.0
4	Y6_p2_refined_r00005_n2003_p003.edf	0.125	1053230.0	2.0
5	Y6_p2_refined_r00006_n2003_p004.edf	0.125	1053180.0	2.0
6	Y6_p2_refined_r00007_n2003_p005.edf	0.125	1052970.0	2.0
7	Y6_p2_refined_r00008_n2003_p006.edf	0.125	1052840.0	2.0
8	Y6_p2_refined_r00009_n2003_p007.edf	0.125	1052770.0	2.0

Fig. 13. Multiple file selection on the Table Widget of PyXScat.



## 9. Sample/Detector Orientation

Another important parameter to take into account during data analysis and visualization is the different rotation of the sample, but especially the detector. Depending on the physical position of the detector array origin, and its own rotations, the resulting array axis may not match with the sample (reciprocal space) axis. The toolbar "Sample Orientation" of PyXScat allows to rotate the arrays through changing the "sample-orientation" parameter of pygix module. Furthermore, PyXScat incorporates the possibility of adding another mirror transformation, i.e. a (-1) multiplication along the horizontal axis. The idea is shown in Figure 14.

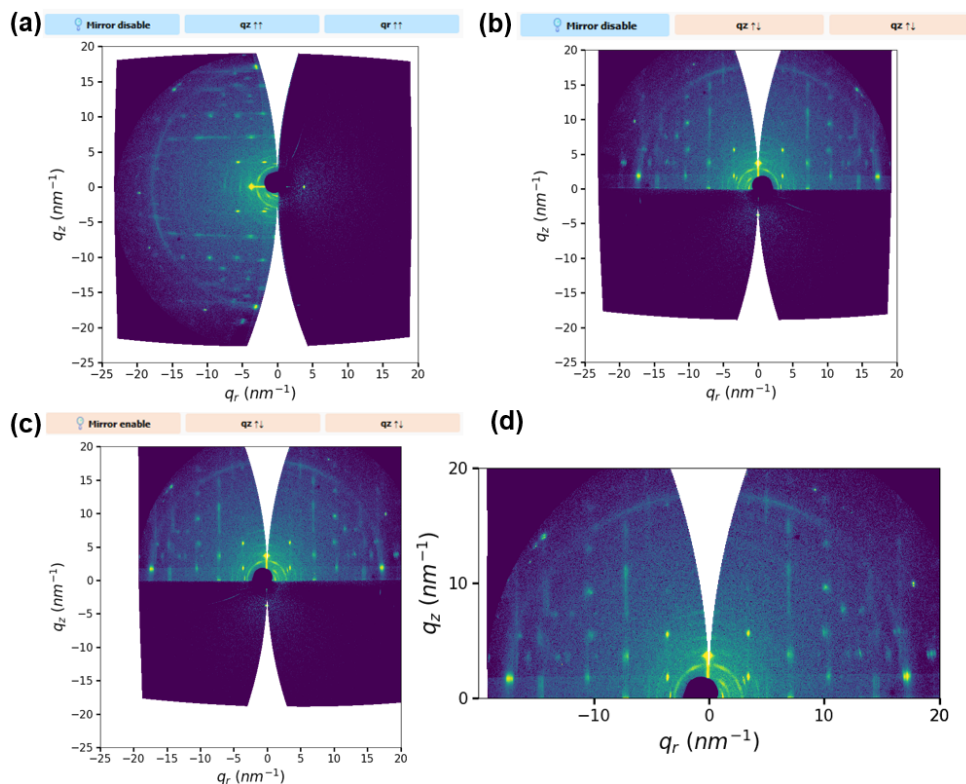


Fig. 14.  $q$ -transformed 2D maps using a (a) wrong and the (b) correct sample orientation; (c) after mirror transformation; (d) after customized figure style parameters..

## References

- [1] Erik B Knudsen, Henning O Sørensen, Jonathan P Wright, Gaeel Goret, and Jérôme Kieffer. FabIO: easy access to two-dimensional X-ray detector images in Python. *Journal of Applied Crystallography*, 46(2):537–539, 2013.
- [2] Jérôme Kieffer and Dimitrios Karkoulis. PyFAI, a versatile library for azimuthal regrouping. In *Journal of Physics: Conference Series*, volume 425, page 202012. IOP Publishing, 2013.
- [3] <https://github.com/tgdane/pygix>.