ESRF | The European Synchrotron

- BioSAXS beamline
- FreeSAS software
- PyFAI software

Jérôme Kieffer
Algorithm & Data Analysis
Experiment Division

Beamline dedicated to macromolecular small angle scattering experiments

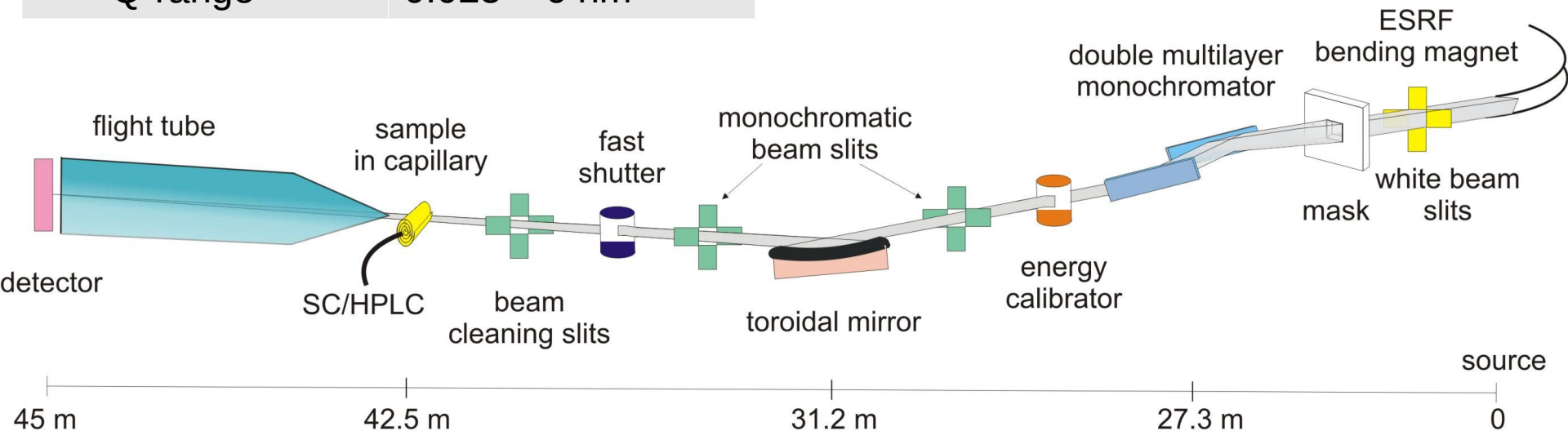
On behalf of the BM29 team:
Mark Tully
Petra Pernot
Hayden Fisher

& Software support:
Antonia Beteva
Marcus Oscarsson
Jean-Baptiste Florial

BioSAXS Beamline @ESRF

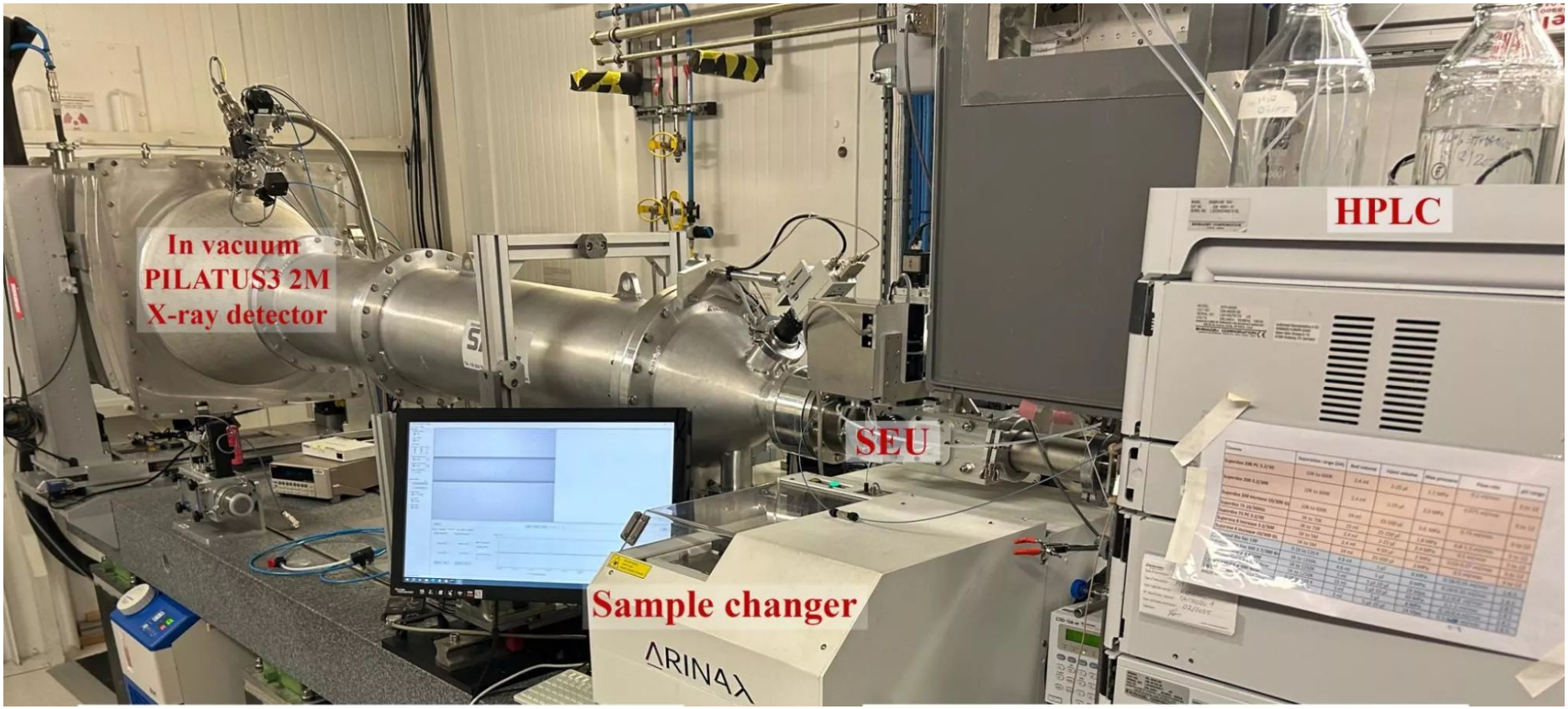
Source	2-pole wiggler
Energy	7-15 keV
Monochromator	Double multilayer
Bandwidth	1%
Beam size	200 µm x 100 µm
Flux	10^{13} ph/s

Detector	Pilatus3 2M
• Pixel size	172 µm
• Distance	2.83 m
• Opening angle	<4°
• Q-range	0.025 – 6 nm^{-1}



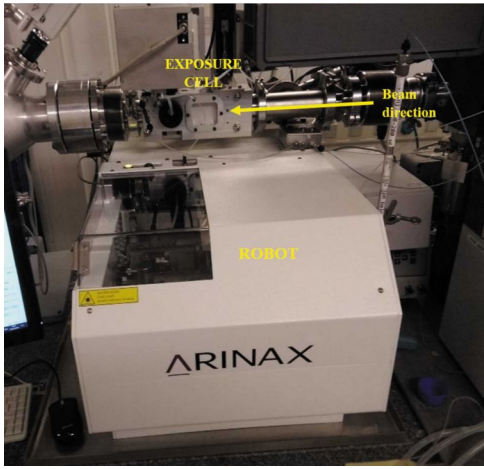
Tully & al (2023). BioSAXS @ ESRF-EBS
https://doi.org/10.1107/S1600577522011286

BM29: Experimental hutch



BioSAXS Beamline: Sample stage

- **Robotic sample changer:**
  - Programmable Buffer-Sample-Buffer sequence
  - Based on 96-well plates with automatic pumping (5 µL)
  - Flow-through the cell to limit radiation damage
  - Programmable temperature (5-60°C)

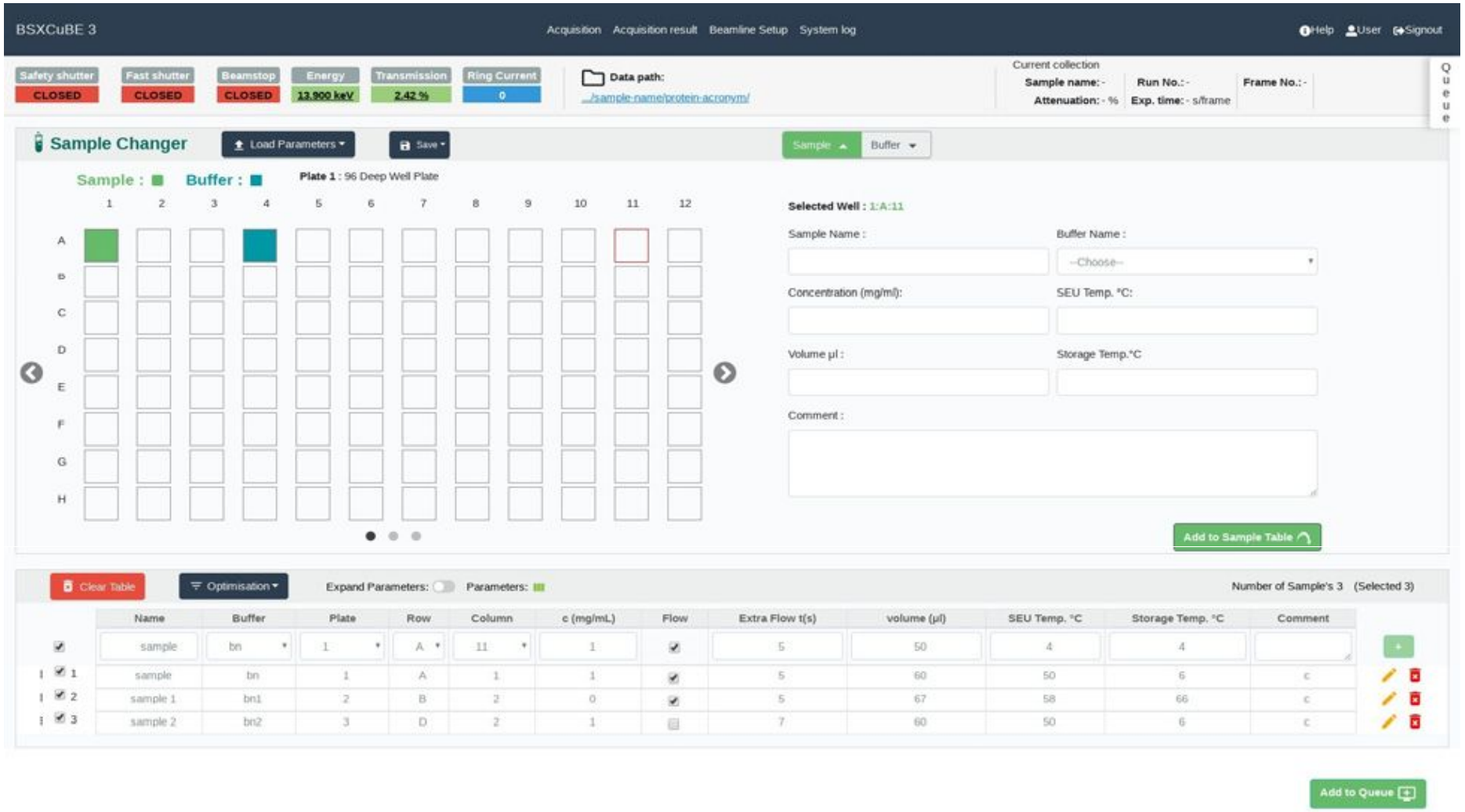


- **Size exclusion chromatography (HPLC)**
  - Separate the different oligomeric states of a macromolecule
  - Follow-up by UV spectrometry

- **Microfluidic setup**
  - Automatic dilution series programming
  - Lower the macromolecule consumption

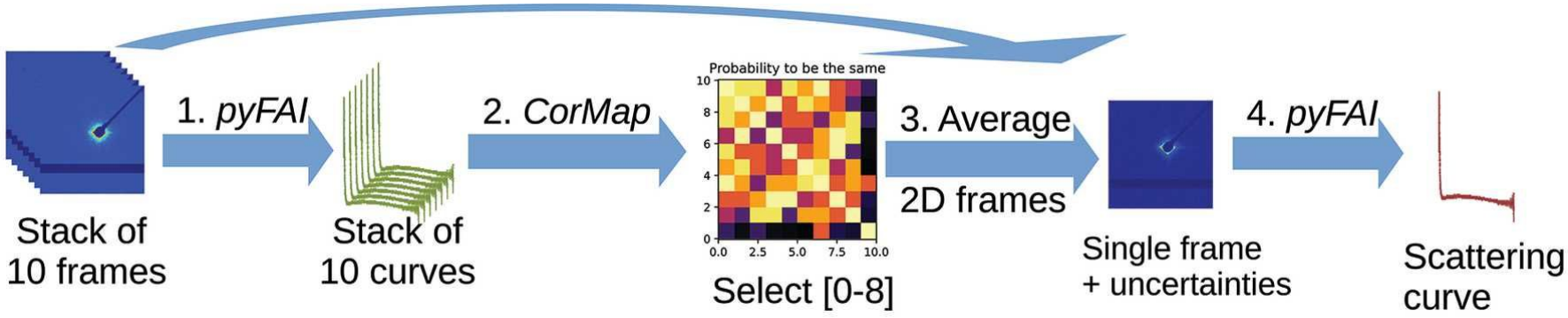
Tully & al (2023). BioSAXS @ ESRF-EBS
https://doi.org/10.1107/S1600577522011286

Software stack at BM29: BSXCube v.3

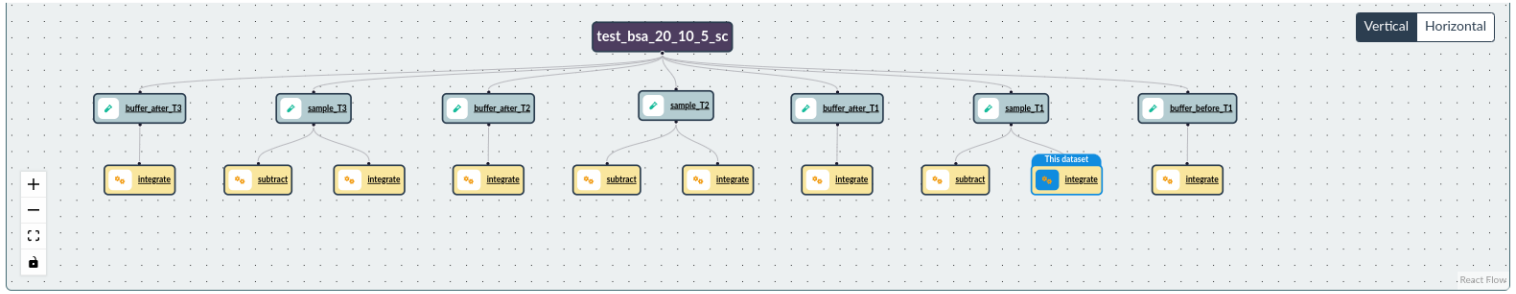


Oskarsson et al. ICALEPCS2019
doi:10.18429/JACoW-ICALEPCS2019-WEPHA115

Data Analysis: Multi-frame integration



Data portal: https://data.esrf.fr

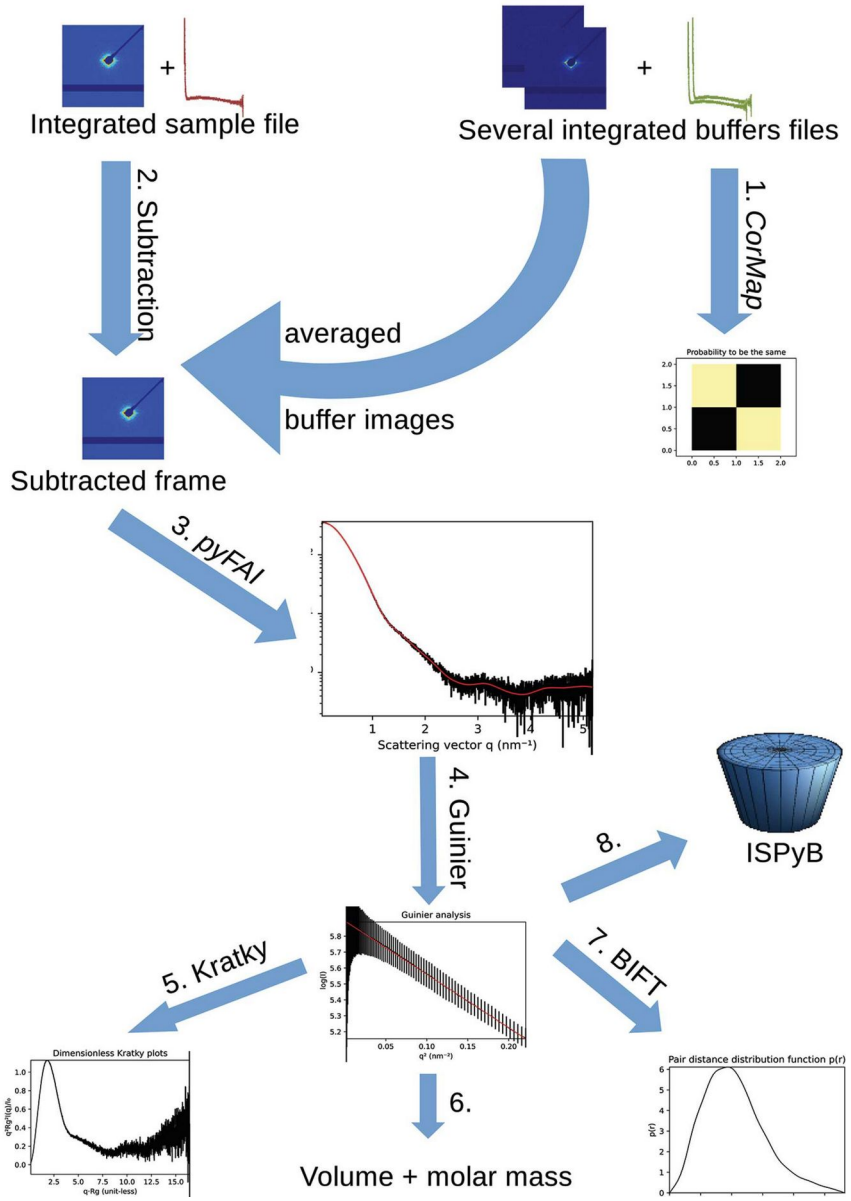


Name integrate
Location test_bsa_20_10_5_sc_sample_T1/processed/integrate
Sample test_bsa_20_10_5_sc
Processed on 27/04/2024 09:53:39
Elapsed time 00:00:00

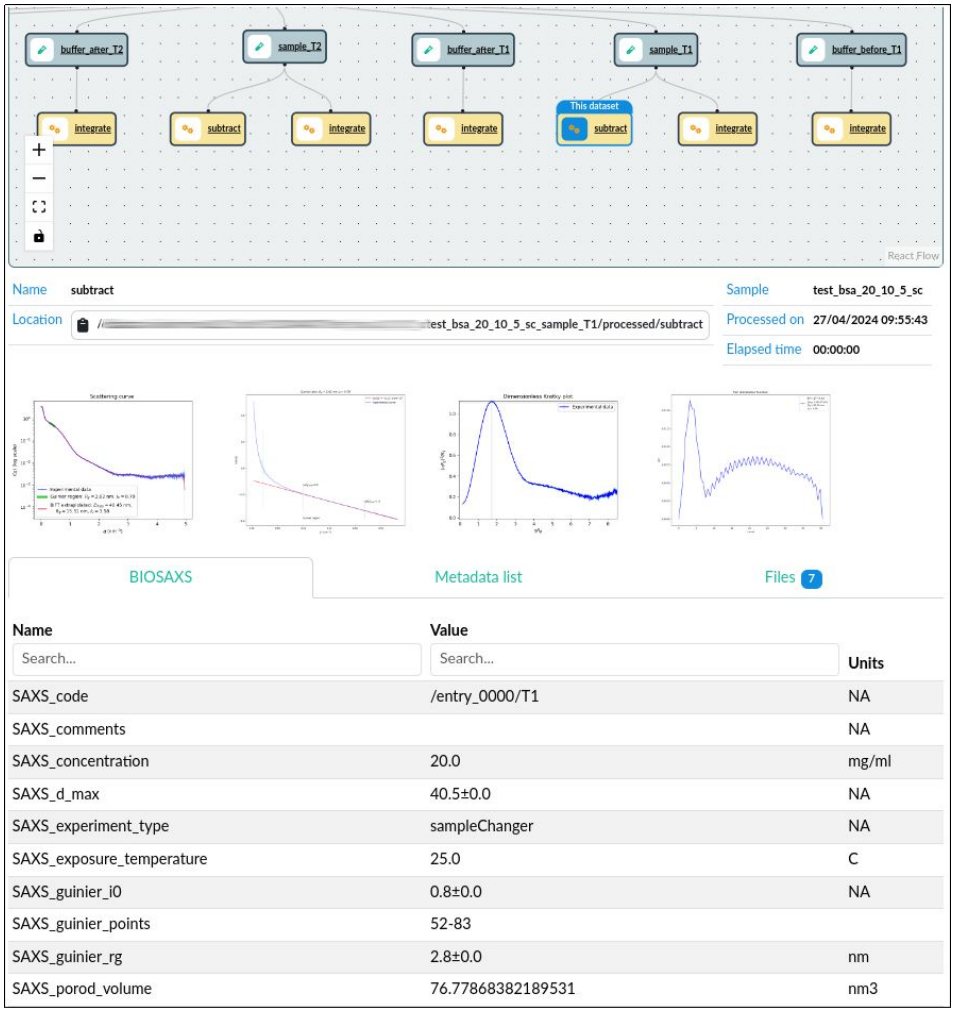
BIOSAXS | Metadata list | Files 3

Name	Value	Units
Search...	Search...	
SAXS_beam_center_x	564.7292386512835	NA
SAXS_beam_center_y	1127.4001384603537	NA
SAXS_code	T1	NA
SAXS_comments		NA
SAXS_concentration	20.0	mg/ml
SAXS_detector_distance	2.826838431400575	NA
SAXS_diode_currents	[0.00046159 0.00046383 0.00046334 0.00046435 0.00046416 0.00046382 0.00046296 0.00046471 0.00046462 0.00046444]	NA
SAXS_experiment_type	sample-changer	NA
SAXS_exposure_temperature	25.0	C
SAXS_frames_averaged	6-9	NA

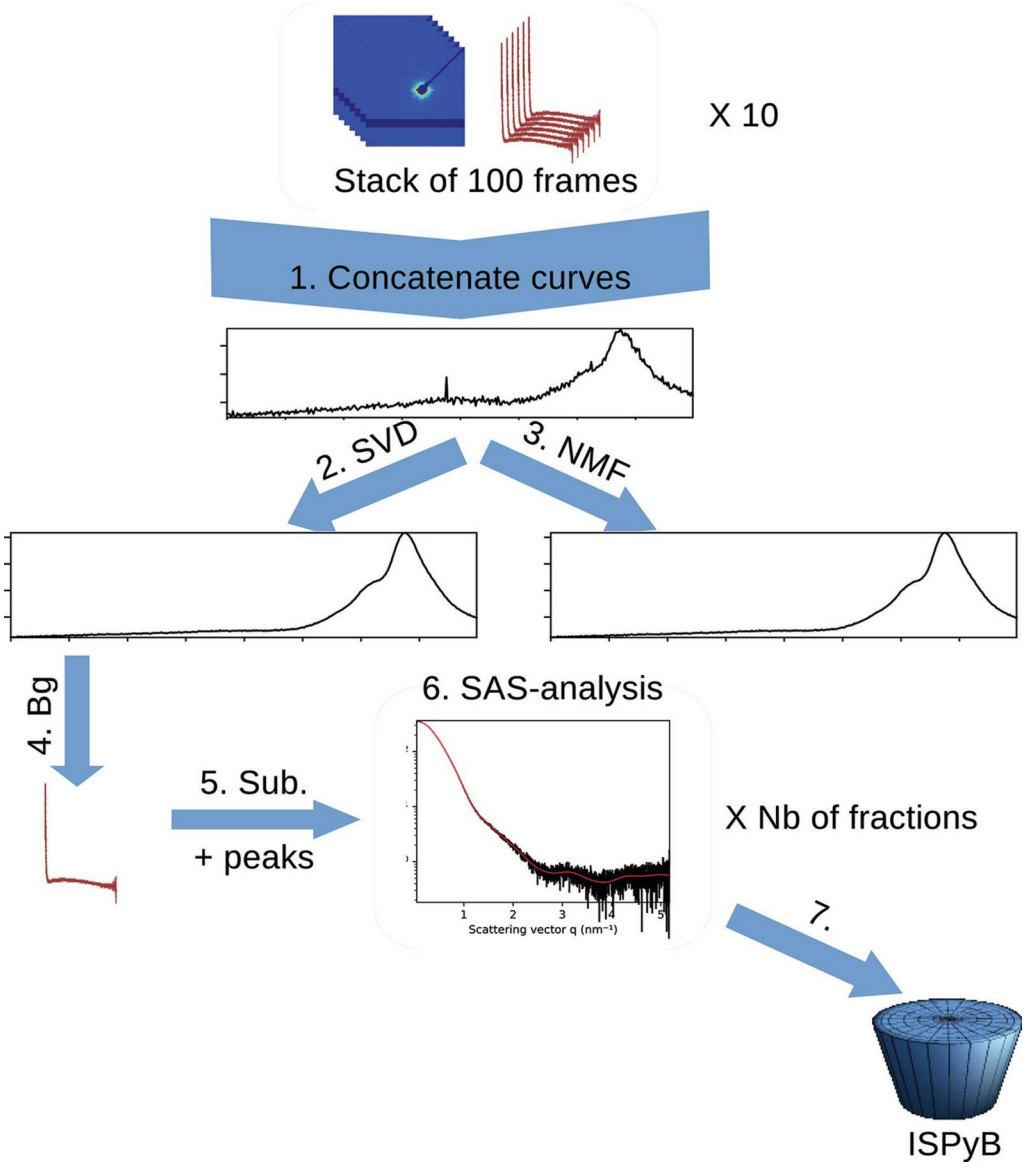
Data Analysis: Buffer averaging and subtraction



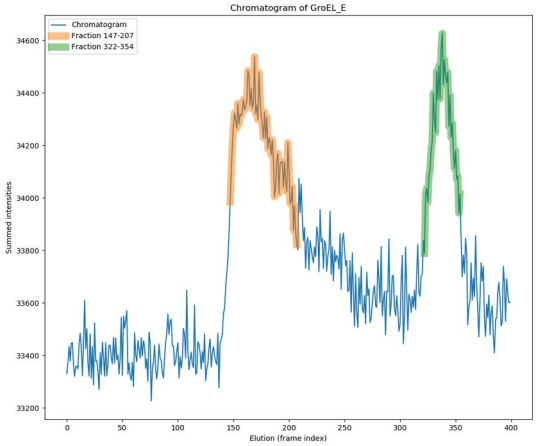
Data portal: https://data.esrf.fr

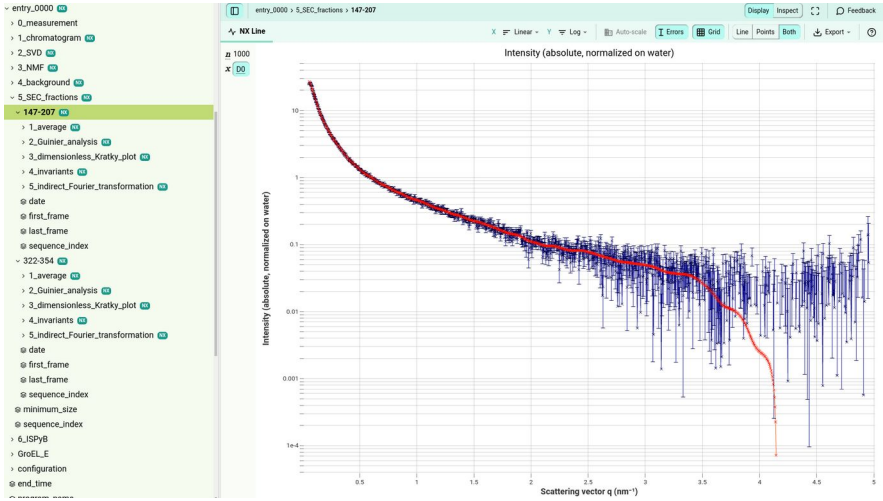


Data Analysis: HPLC mode

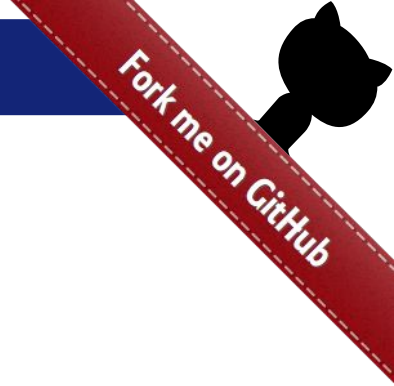


Chromatogram analysis





FreeSAS



Beamline independent SA(X)S software

Features:
- Guinier analysis
- Bayesian Inverse Fourier Transform
- Corelation Map

Credits:
- Martha Brennich
- Guillaume Bonami
- Jesse B. Hopkins
- Mayank Yadav

https://github.com/silx-kit/freesas

Kieffer et al., J. Synch Rad. (2022) doi:10.1107/S1600577522007238

How to determine THE best Guinier region ?

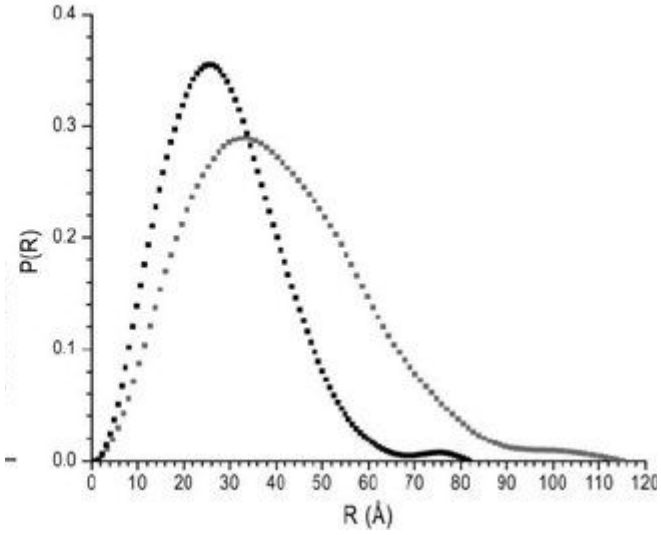
- **First order Taylor expansion at q=0:**
  - Issue with aggregation !

$$I(q) = I_0 \exp\left(-\frac{R_g^2}{3}q^2\right)$$

- **Several implementation:**
  - Like ATSAS*:*
    - **Petoukhov et al, J. Appl. Cryst (2007)** **https://doi.org/10.1107/S0021889807002853**
  - Like BioXTAS-RAW
    - **J. B. Hopkins, J. Appl. Cryst (2024)** **https://doi.org/10.1107/S1600576723011019**
  - Like GPA:
    - **C. D. Putnam, J. Appl Cryst (2016)** **https://doi.org/10.1107/S1600576716010906**
  - DNN inspired from:
    - **Molodenskiy et al. Structure (2022)** **https://doi.org/10.1016/j.str.2022.03.011**

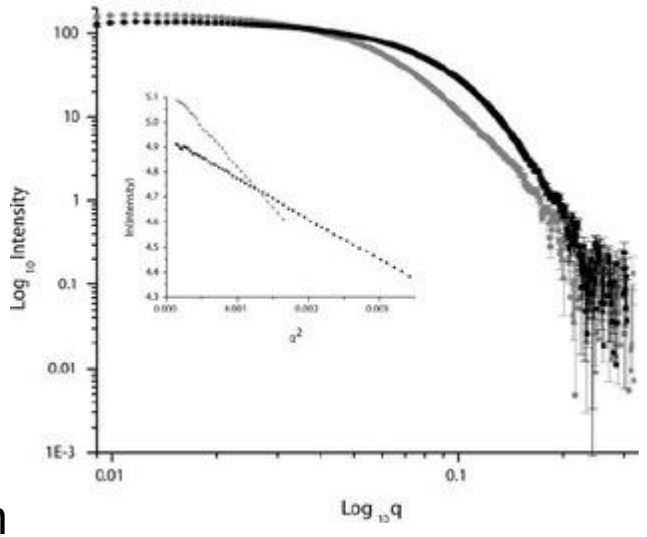
Bayesian Inverse Fourier Transform

Figures: Joanna K. Krueger



Easy

Difficult



Ill-posed inverse problem

GNOM: Tikhonov regularization
BIFT: Bayesian regularization

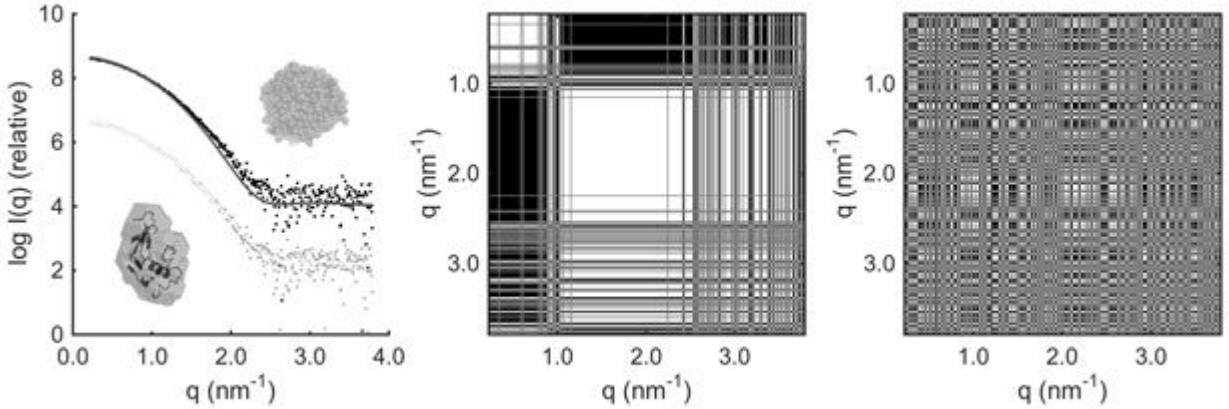
- Implemented in Cython,
- Multi-threaded
- BLAS matrix multiplication
- Based on S. Hansen's paper
- J. Hopkins' code analysis (RAW)
  → Solution in ~1 second

Hansen, S. J Appl Cryst (2000) 33, 1415-1421.
DOI: 10.1107/S0021889800012930

Radiation damage assessment

CorMap:

Sign of the difference between 2 SAXS datasets



Large regions: Not equivalent

Small regions: Equivalent

The largest region is compared with the total size of the pattern:
The probability is the same a getting that many successive heads/tails when throwing a coins as many times as the pattern has points.

$$A_n(x) = \begin{cases} \sum_{j=0}^{x} A_{n-1-j}(x) & \text{for } n > x; \\ 2^n & \text{for } n \le x. \end{cases}$$

$$B_n(x) = 2A_{n-1}(x-1) \quad \text{for } x \ge 1.$$

$$P_n(x) = 2^{\log_2(2^n - B_n(x)) - n}$$

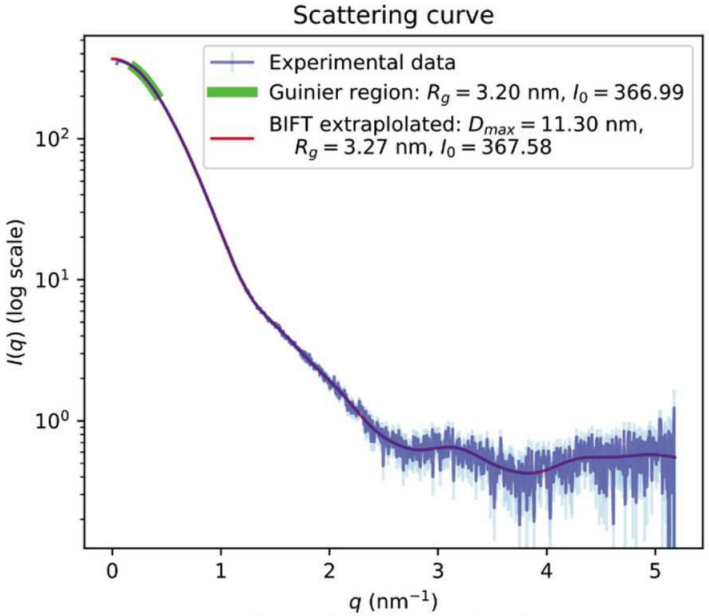
Franke et al. Correlation Map, Nat Methods 12, 419–422 (2015). https://doi.org/10.1038/nmeth.3358

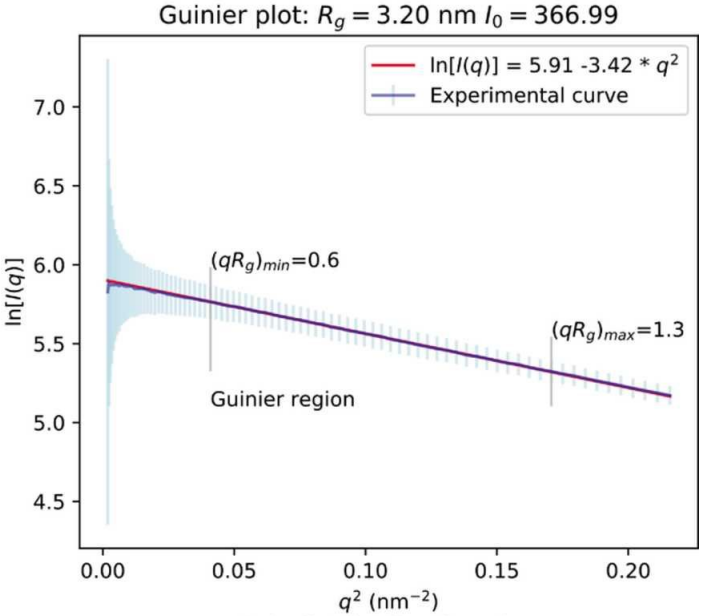
Image: https://www.embl.org

SAS analysis → all-in-one tool

sample_BSA_5mg-integrate-sub.dat

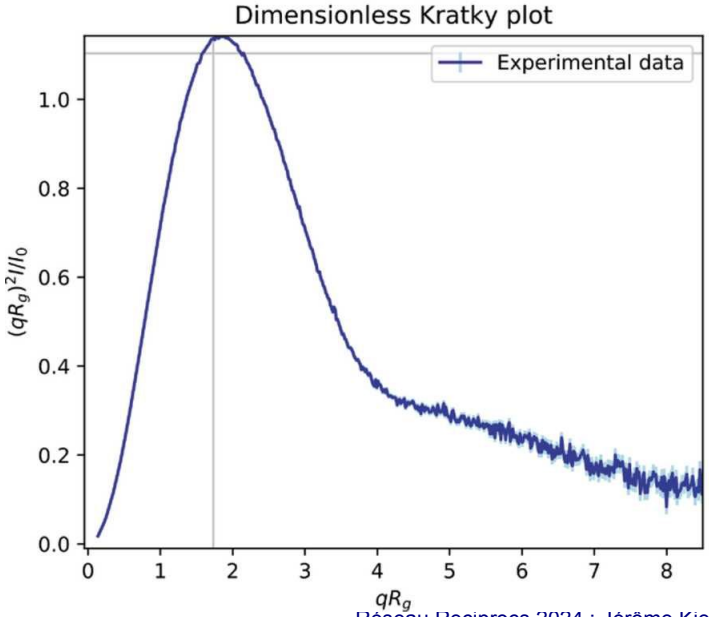
SAS plot

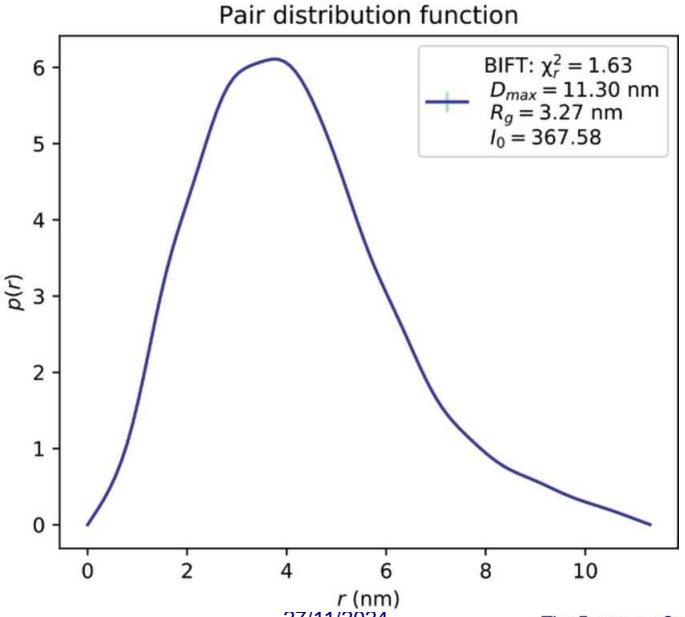




Guinier plot

Kratky plot



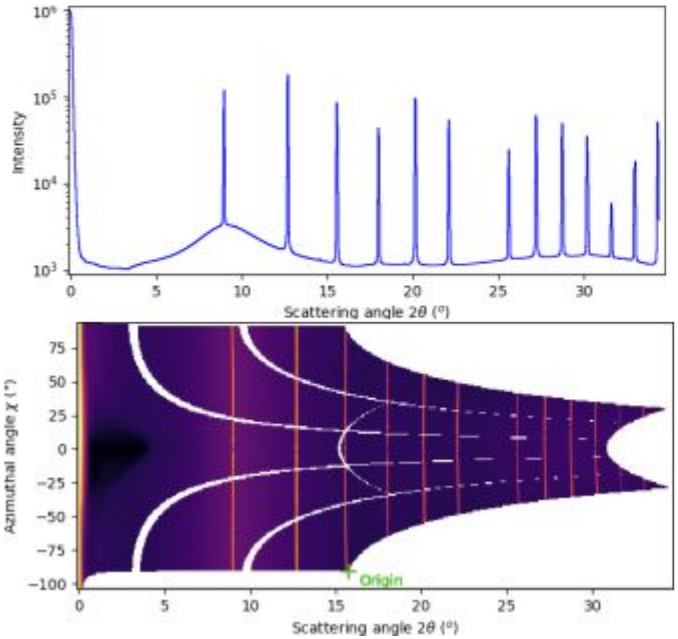


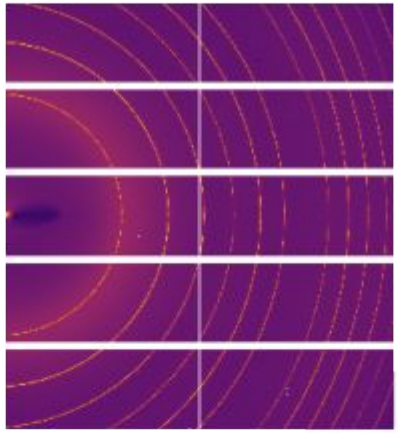
BIFT

Fast Azimuthal Integration library using Python

PyFAI

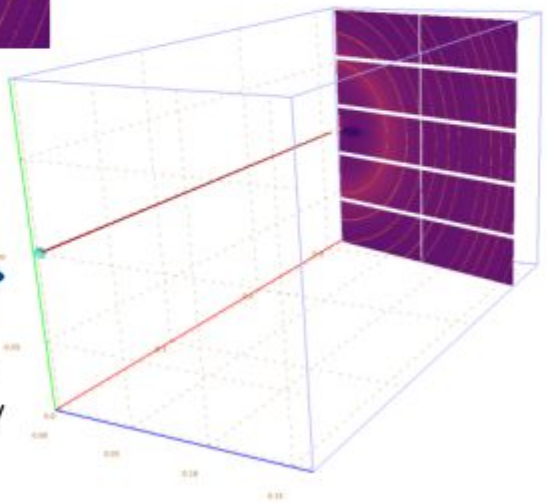
Fast Azimuthal Integration





Average
1d and 2d

Calibrate
3d geometry



Kieffer, J., Valls, V., Gutierrez-Fernandez, E., Ashiotis, G., Karkoulis, D., Nawaz, Z., Deschildre, A., Vincent, T., Picca, F. E., Massahud, E., Payno, H., Huder, L., Wright, J. P., Pandolfi, R., Jankowski, M., Paleo, P., Faure, B., Storm, M., De Nolf, W., Wright, C. J., Hopkins, J. B., Pascal, E., Weninger, C., Detlefs, C., Plaswig, F., Lavanchy, A., gbenecke, zxs-un, iltommi, & dodogerstlin
https://doi.org/10.5281/zenodo.13756914

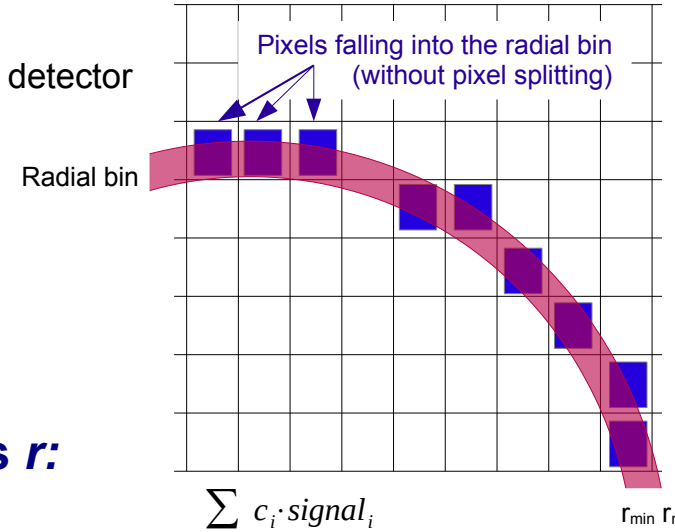
How azimuthal integration works

- **Pixel-wise corrections:**

$$I_{cor} = \frac{I_{raw} - I_{dark}}{F \cdot \Omega \cdot P \cdot A \cdot I_0} = \frac{signal}{normalization}$$

Where: I_0 is the incoming flux (pixel independent)

  - I_{raw} and I_{dark} are the signal measured from the detector
  - F is the flat-field correction
  - Ω is the solid angle for this pixel
  - P is the polarization factor
  - A is the parallax correction factor



- **Averaging over a bin defined by the radius *r*:**
  - Need for pixel splitting?
  - c_i being the fraction of the pixel i contributing to bin_r

$$\langle I \rangle_r = \frac{\sum_{i \in bin_r} c_i \cdot signal_i}{\sum_{i \in bin_r} c_i \cdot normalization_i}$$

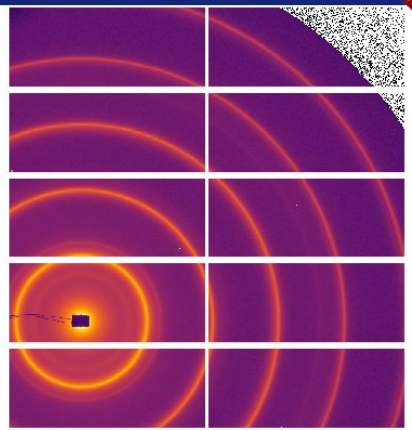
- **Associated uncertainty propagation:**
  - Assuming there is no correlation between pixels
  - Pixel splitting can create correlation between bins...

$$\sigma(I_r) = \sqrt{\frac{\sum_{i \in bin_r} c_i^2 \cdot variance_i}{\sum_{i \in bin_r} c_i^2 \cdot normalization_i^2}}$$

$$\sigma(\langle I \rangle_r) = \sigma(I_r) \frac{\sqrt{\sum_{i \in bin_r} c_i^2 \cdot normalization_i^2}}{\sum_{i \in bin_r} c_i \cdot normalization_i}$$

Math from Kieffer et al.; *J. Synch. Radiation* (2020)
https://doi.org/10.1107/S1600577520000776

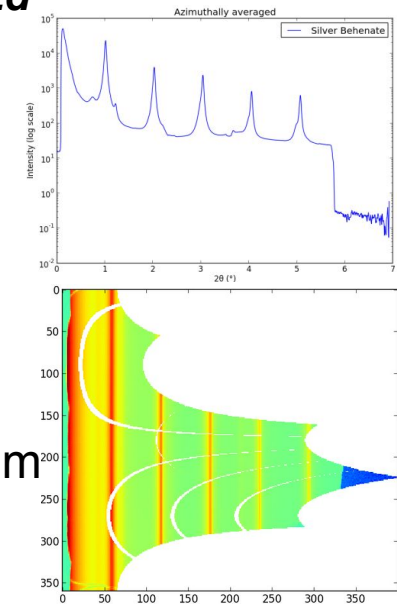
Concepts in PyFAI



- **Image**

  2D array of pixels as *numpy* array
  read using *silx, fabio, h5py,* ...

- **Azimuthal integrator: core object**
  - **powder diagram using *integrate1d***
  - **"cake" image, azimuthally regrouped using *integrate2d***



- **Detector object**
  - **Calculates the pixel position (distortions)**
  - **Handles the mask for invalid pixels**

  → **saved as a HDF5 file**

- **Geometry**

  Position of the detector from the sample & incoming beam

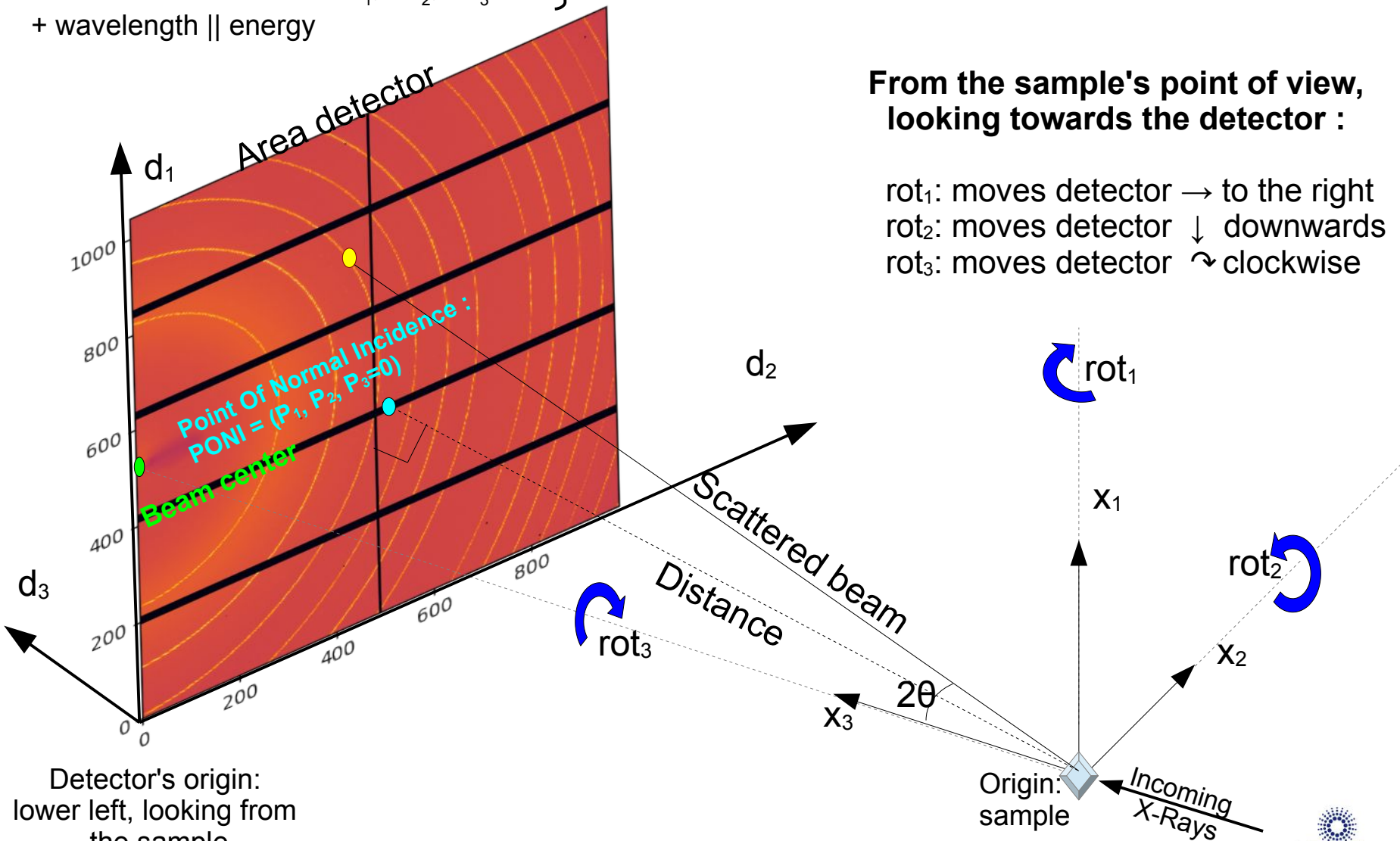
  → **saved as *PONI*-file**

http://www.silx.org/doc/pyFAI/dev/geometry.html#detector-position

Geometry in pyFAI

Parameters:
* 3 distances in meters: dist, $poni_1$, $poni_2$
* 3 rotations in radians: rot_1, rot_2, rot_3
+ wavelength || energy

} *PONI*-file



From the sample's point of view, looking towards the detector :

rot_1: moves detector → to the right
rot_2: moves detector ↓ downwards
rot_3: moves detector ↷ clockwise

Detector's origin: lower left, looking from the sample

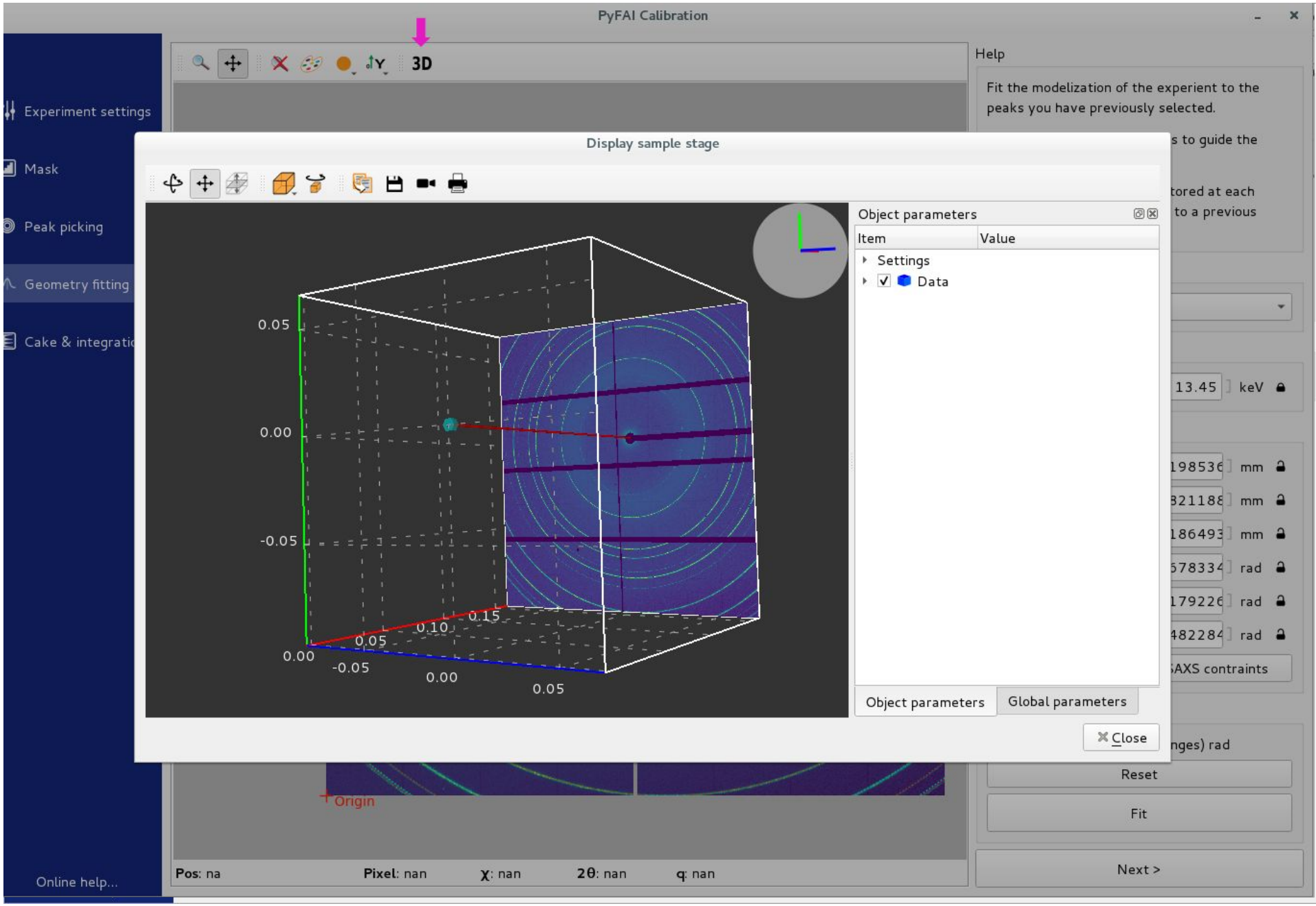
Calibration

Determine the position of the detector vs sample

Calibration in pyFAI

- **Geometry is best determined from the analysis of a known reference sample**
- **This calibration step is preferred to measuring distances and beam center position**
  - The prerequisite is:
    - **detector geometry and mask,**
    - **calibrant (LaB_6, CeO_2, AgBh, …)**
    - **wavelength or energy used**
  - Only the position of the detector and the rotation needs to be refined:
    - **3 translations: dist, $poni_1$ and $poni_2$**
    - **~~3~~2 rotations: rot_1, rot_2, ~~rot_3~~**
- **It is divided into 4 major steps:**
  1) Extraction of groups of peaks
  2) Identification of peaks and groups of peaks belonging to same ring
  3) Least-squares refinement of the geometry parameters on peak position
  4) Validation by a human being of the geometry
- **PyFAI assumes this setup does not change during the experiment → Put the geometry in cache**

Calibration GUI in pyFAI



Azimuthal integration

A quest for speed:

- Histogram based algorithms
- Sparse matrix multiplication

What happens during an integration

1) **Get the pixel coordinates from the detector, in meter.**

   There are 3 coordinates per pixel corner, and usually 4 corners per pixel.

   1Mpix image → 48 Mbyte !

2) **Offset the detector's origin to the PONI and rotate around the sample**

3) **Calculate the radial (2θ) and azimuthal (χ) positions of each corner**

4) **Assign each pixel to one or multiple bins.**

   If a look-up table is used, just store the fraction of the pixel.

   Then for each bin sum the content of all contributing pixels.

5) **Histogram bin position with associated intensities**

6) **Histogram bin position with associated normalizations (i.e. solid angle)**

7) **Return bin position and the ratio of the two sums:**

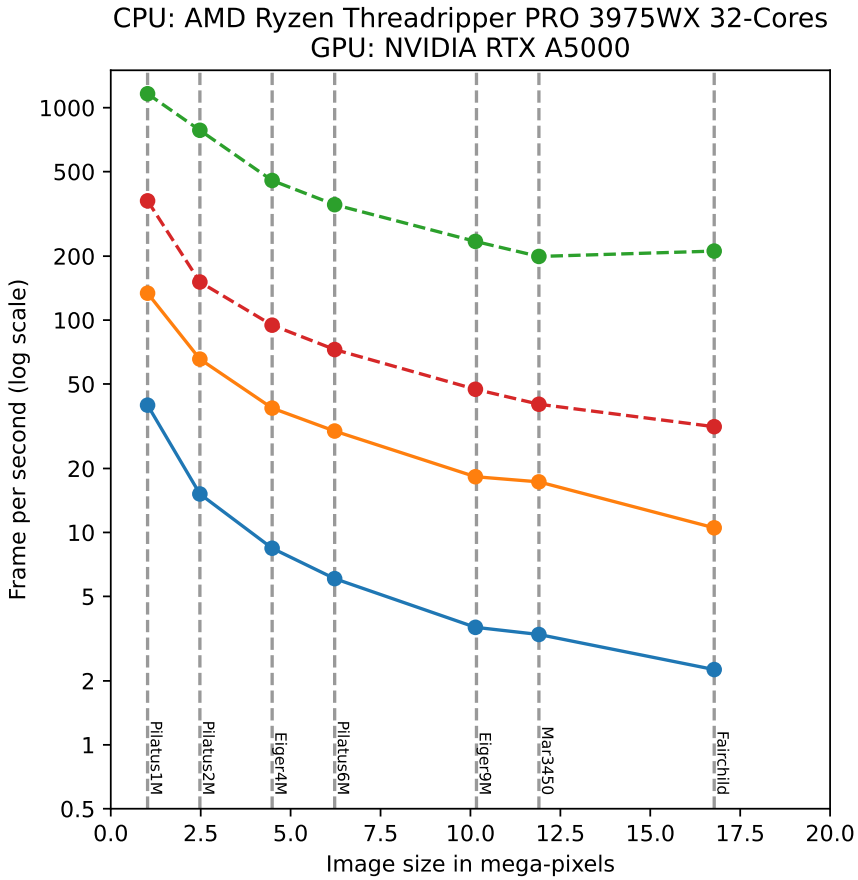
Can be performed with sparse-matrix multiplications

$$\langle I \rangle_r = \frac{\sum_{i \in bin_r} c_i \cdot signal_i}{\sum_{i \in bin_r} c_i \cdot normalization_i}$$

Advantages of *histograms* vs matrix multiplication

	Histograms	*CSR* matrix multiplication
Pro	• **Easier to understand** • **Low memory consumption** • **Fast initialization**	• **Faster, even on a single core** • **Many-core friendly** – OpenMP and OpenCL
Con	• **Pretty slow** • **Hardly parallelizable**	• **Slower initialization** • **The sparse matrix can be large**
Rule of thumb:	< 5 frames	≥ 5 frames

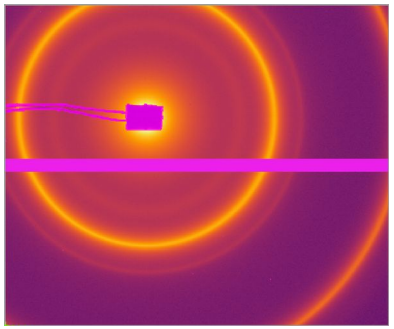
Benchmark: Let's speak about speed !



High frequency noise issue

Where pixel splitting comes-in

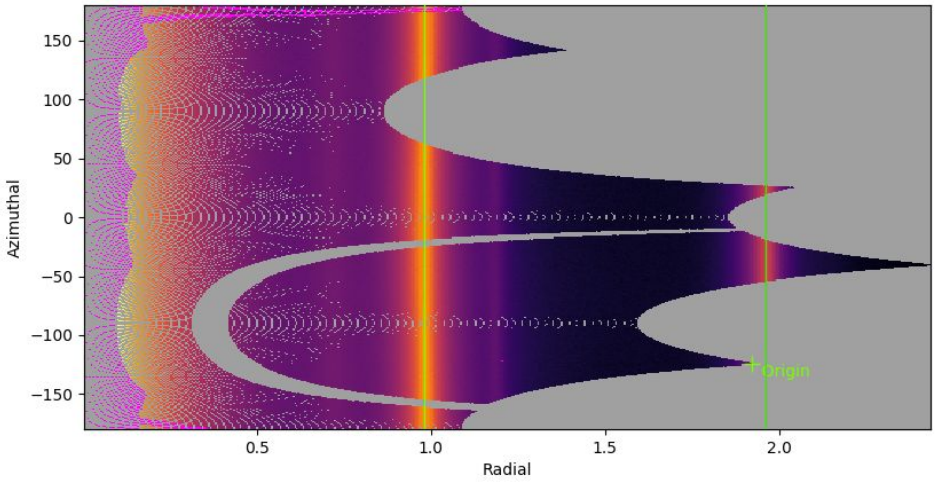
Example with SAXS data integrated in 2D



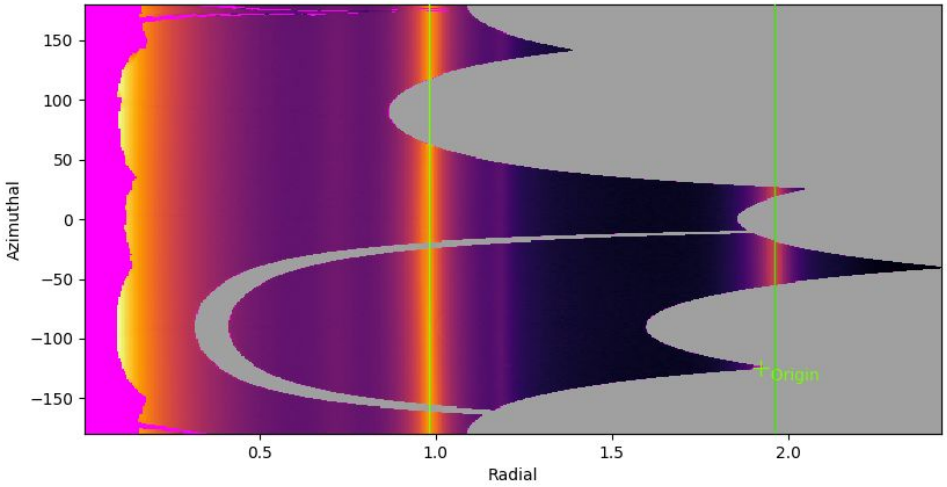
Pilatus 200k:
~500 x 400 pixels

2D averaging over 512x360 bins

Without pixel splitting



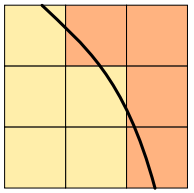
With pixel splitting



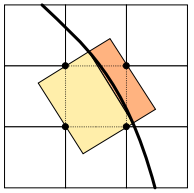
creates bin cross-correlation

Pixel splitting schemes available in pyFAI

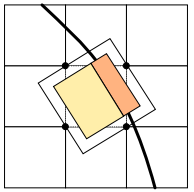
- **No pixel splitting: default histograms**
  - Each pixel contributes to a single bin of the result
  - No bin correlation but noisy
  - The pixel has no surface: sharpest peaks



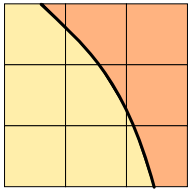
- **Bounding-box pixel splitting**
  - The smoothest integrated curve
  - Blurs a bit the signal



- **Pseudo pixel splitting (deprecated)**
  - Scale down the bounding box to the pixel area, before splitting.
  - Good cost/precision compromise, similar to FIT2D



- **Full pixel splitting**
  - Split each pixel as a polygon on the output bins.
  - Costly high-precision choice



Impact of pixel splitting on integration speed

- **Histogram based algorithms:**
  - Each pixel is split over the bins it covers.
  - The corner coordinates have to be calculated (4x slower initialization)
  - The slow down is function of the oversampling factor, for every image

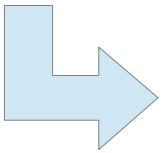
- **Sparse matrix multiplication based algorithms**
  - The sparse matrix contains already the pixel splitting scheme
  - Longer initialization time related to the oversampling factor
  - There are *NO* performance penalty on the integration itself

All those consideration are independent of the programming language

Nevertheless, Python which is interpreted is expected to be 1000x slower than:
- compiled code like C, C++, Fortran, ...
- JIT compiled code like Java, Julia or numba

Grazing incidence data visualization

Edgar Gutierrez-Fernandez
Thomas Dane



Poster

Grazing-Incidence / Fiber diffraction support

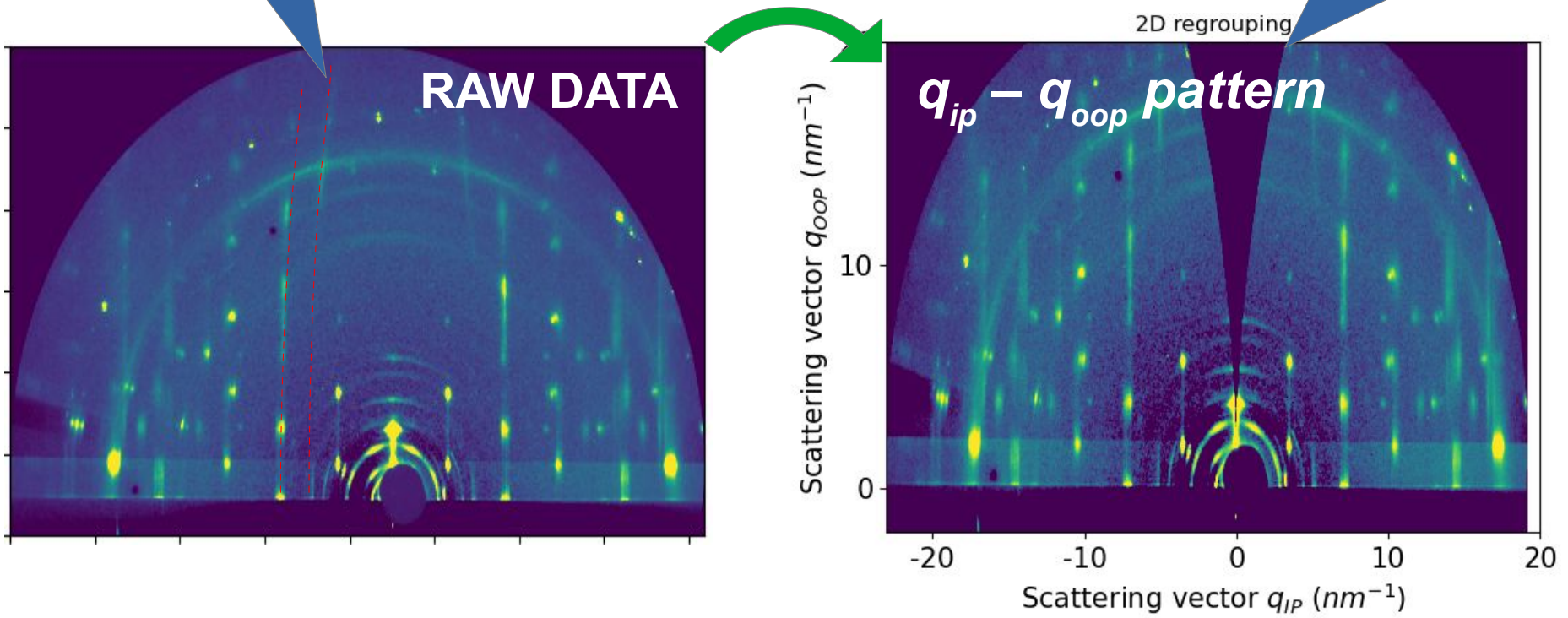
Reshaping into the components of $\vec{q}$

$$\vec{q} = \vec{q}_x(beam) + \vec{q}_y(horz) + \vec{q}_z(vert)$$

$$q_{IP} = \sqrt{q_X^2 + q_y^2} \qquad q_{OOP} = q_z$$

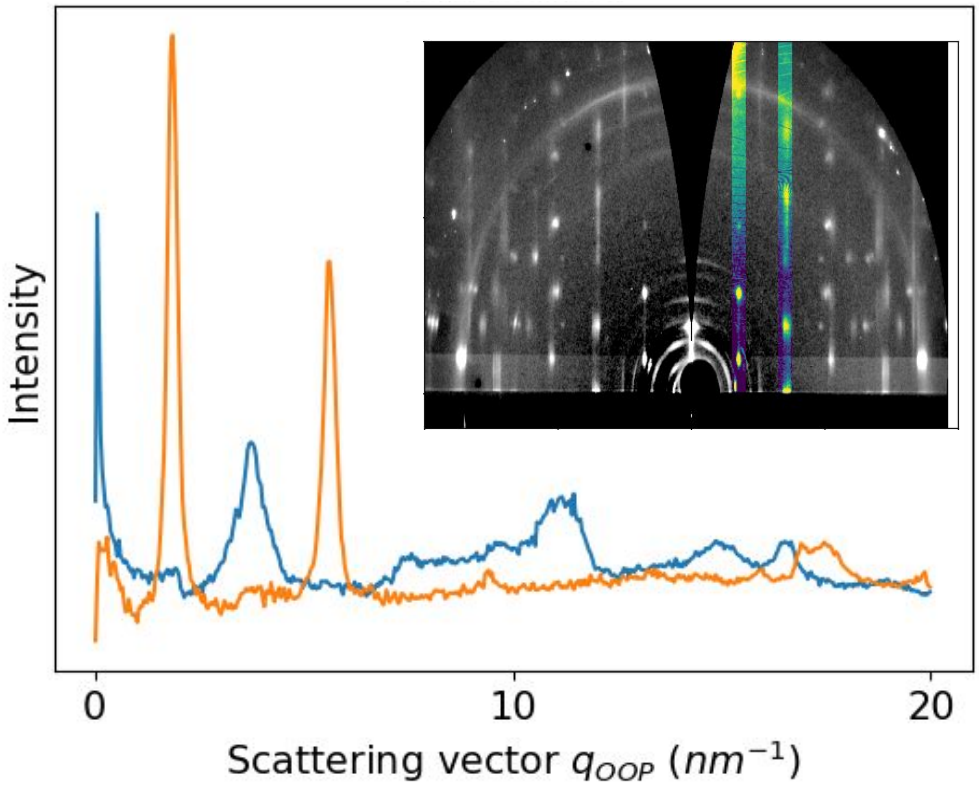
In a flat detector, the diffraction rods are curved

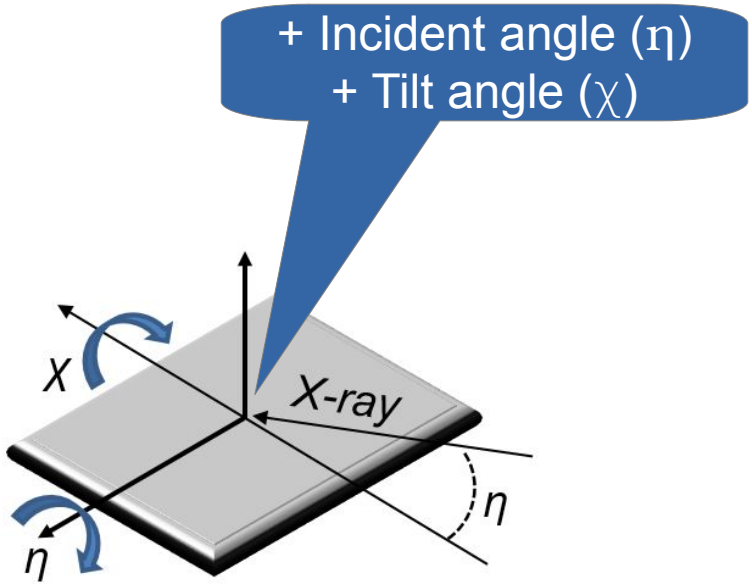
Missing wedge: the probed region of the reciprocal space is a sphere.



Grazing-Incidence / Fiber diffraction support

1D-Integration along $\vec{q}$ components



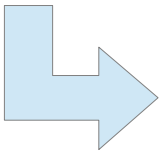


Outlier rejection

Outlier rejection:

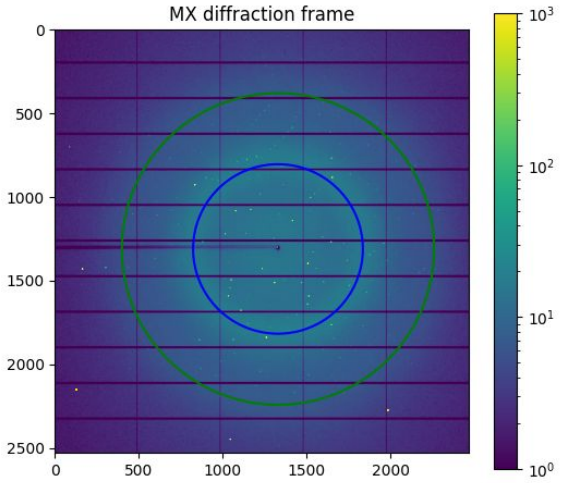
Sigma-clipping

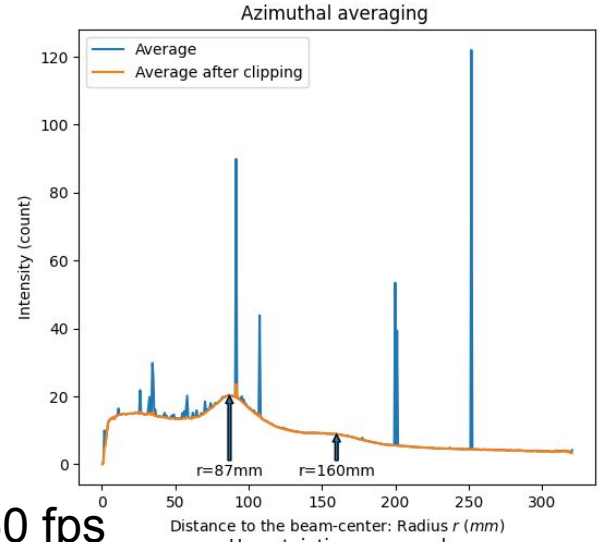
Median filtering



Poster

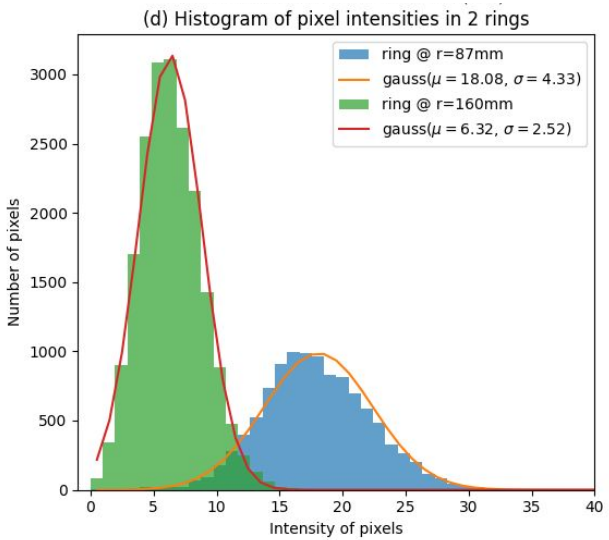
Sigma clipping

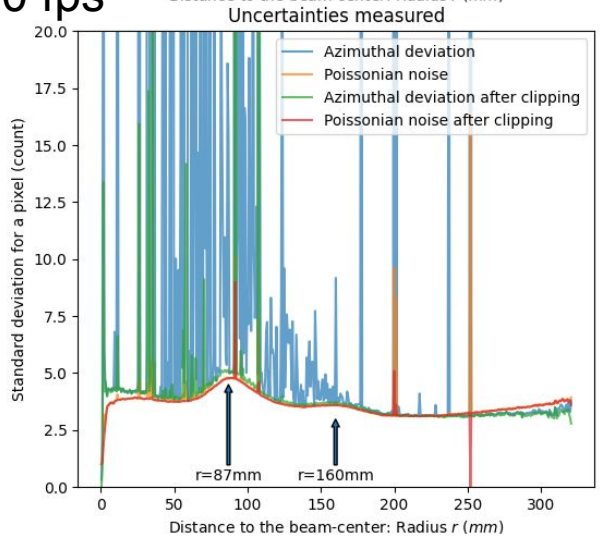




Implemented in:

- Python
- Cython
- OpenCL → 250 fps



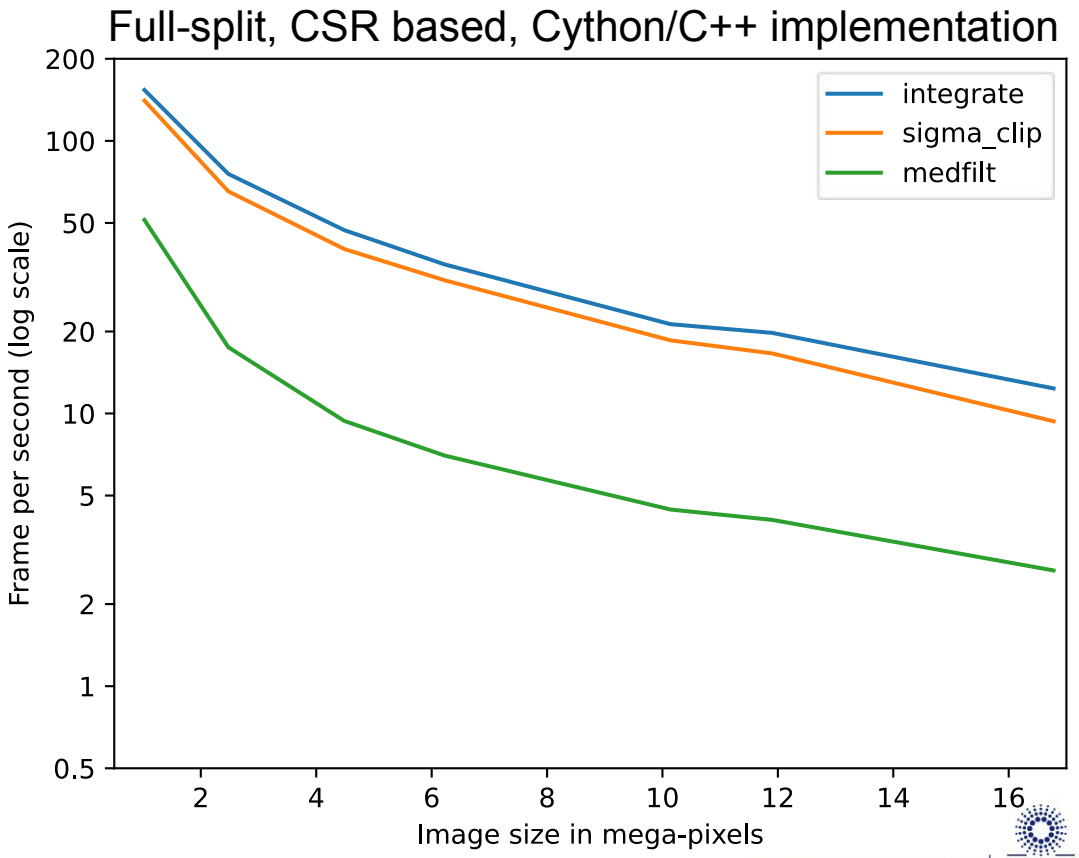


No pixel splitting

Kieffer et al. J. Appl. Cryst (2024)
Accepted, https://arxiv.org/pdf/2411.09515

Median filtering / Quantile averaging in azimuthal space

- **Outlier rejection based on extrema rejection**
  - Oversampling for Rietveld refinement with large pixel size
- **Requires one sort per azimuthal bin, thus expensive !**
  - Implementations:
    - **Python**
    - **Cython**
    - **OpenCL (*)**



* New parallel sort algorithm suitable for variable size ensembles.

Unpublished / experimental work

Community & alternatives

- Using pyFAI as a library
- Alternative software

Other applications relying on pyFAI: **40** on github!

- **NanoPeakCell: Serial crystallography pre-processing**
  - Nicolas Coquelle, IBS Grenoble
- **PySaxs: data analysis for SAXS experimental station**
  - Olivier Taché, CEA Saclay
- **Dpdak: online data analysis for Saxs data**
  - Gunthard Benecke, Petra III
- **Dioptas: offline data analysis for high pressure diffraction**
  - Clemens Percher, APS → EuXFEL
- **Bubble: online data analysis for Saxs/Waxs data**
  - Vadim Diadkin, Dubble & SNBL CRG beamlines
- **Project for materials and strain analysis**
  - Jozef Keckes, Loeben university, Austria
- **xPDFsuite**
  - Prof. Simon Billinge, U. of Columbia

https://github.com/silx-kit/pyFAI/network/dependents?dependent_type=PACKAGE

http://www.silx.org/doc/dev/pyFAI/ecosystem.html

Many different tools exist …

Name	License	Language	Owner	Details
PyFAI	MIT	Python/ C++	ESRF	GPU possible
FIT2D	MIT	Fortran	ESRF	Discontinued
XRDUA	GPL	IDL	U. Antwerp	diff-map
Dawn	EPL	Java	Diamond	
DataSqueze	Shareware	Java	U. Pennsylvania	
Foxtrot	Academic	Java	Xenocs/Soleil	SAXS only
Maud	Freeware	Java	U.Trento	Focus on materials
GSAS-II	BSD	Python/ Fortran	APS / U.Chicago	
Scikit-Beam	BSD	Python	NSLS-II / BNL	
AzInt	MIT	Python/C++	Max-IV	FPGA possible
SAXSdog	-	Python	TU-Graz	

Questions ?