ESRF User Meeting 2025
PyFAI tutorial

Jérôme Kieffer

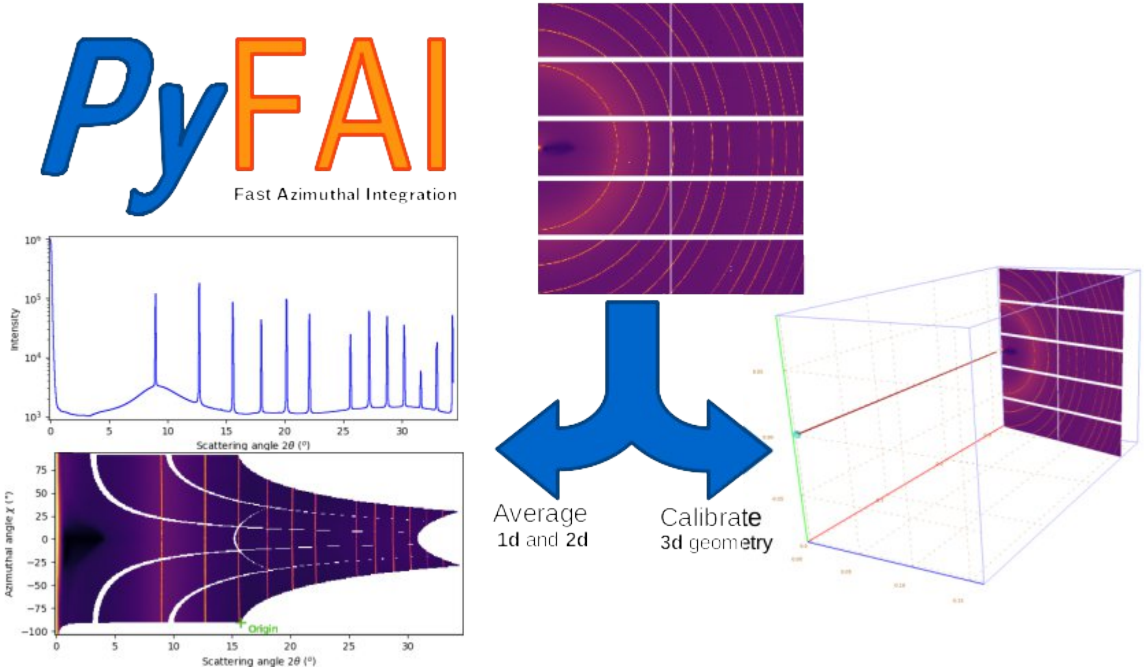
Edgar Gutiérrez Fernández

10/02/2025

PyFAI tutorial - overview

- **10/02/2025 / 14:00 – 17:00**
- *Each user should use its own computer (Windows, Linuc, MacOS)*
- **1st half: concepts of pyFAI**
    - Motivations
    - Applications
    - Working philosophy

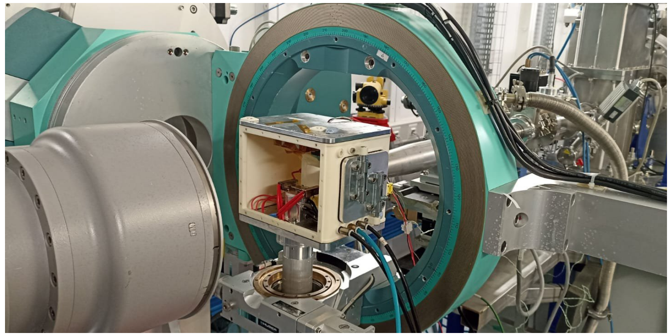
- **Coffee break (~15 minutes)**
- **2nd half: hands-on**
    - Installation of python: venv/conda/visa
    - Installation of pyFAI
    - Calibration GUI
    - AzimuthalIntegratrion
    - Other pyFAI applications: integrate, diffmap-view, worker…



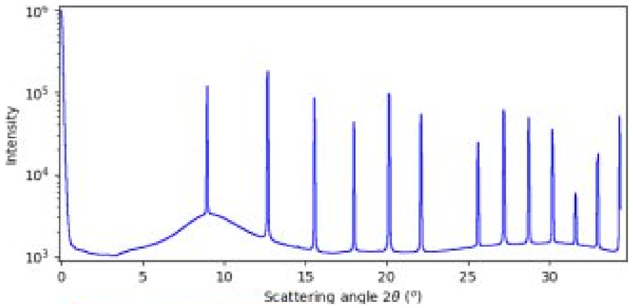
edgar.gutierrez-fernandez@esrf.fr

PyFAI = Python Fast Azimuthal Integrator

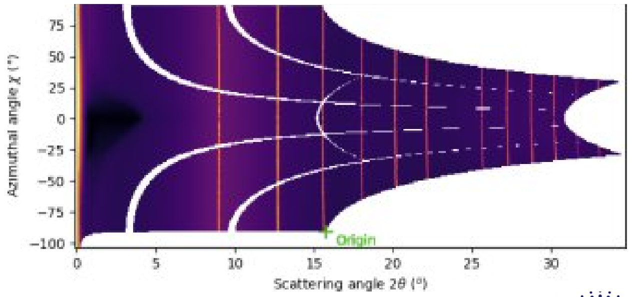
*BM28



PyFAI
Fast Azimuthal Integration



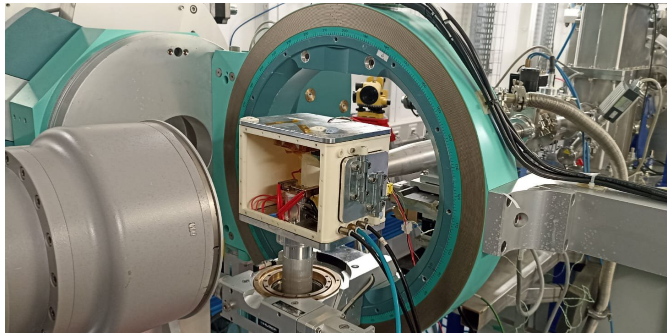
Setup experiment → Data collection → Data reduction → Data analysis



edgar.gutierrez-fernandez@esrf.fr

PyFAI = Python Fast Azimuthal Integrator

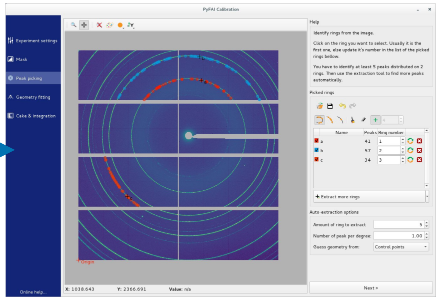
**BM28*



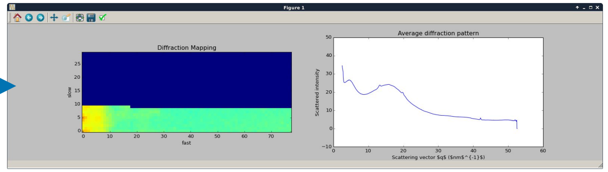
PyFAI
Fast Azimuthal Integration

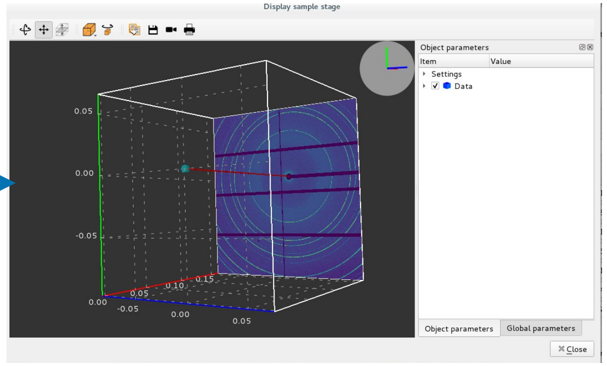
Setup experiment → **Data collection** → **Data reduction**

Calibration



Diffraction mapping



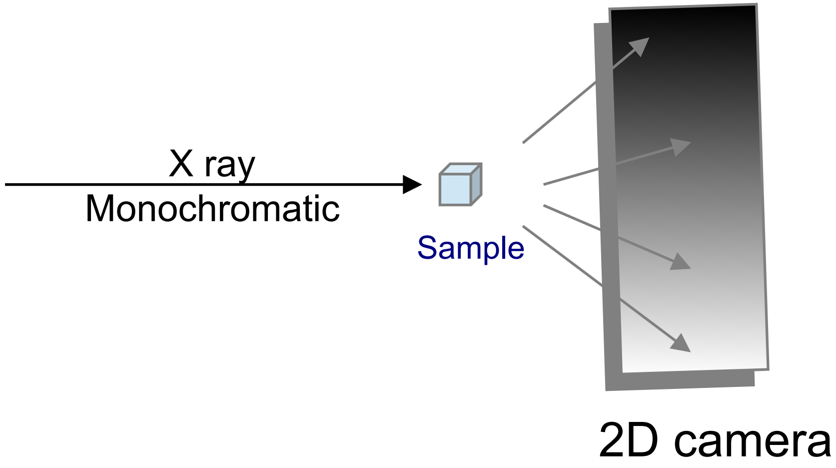


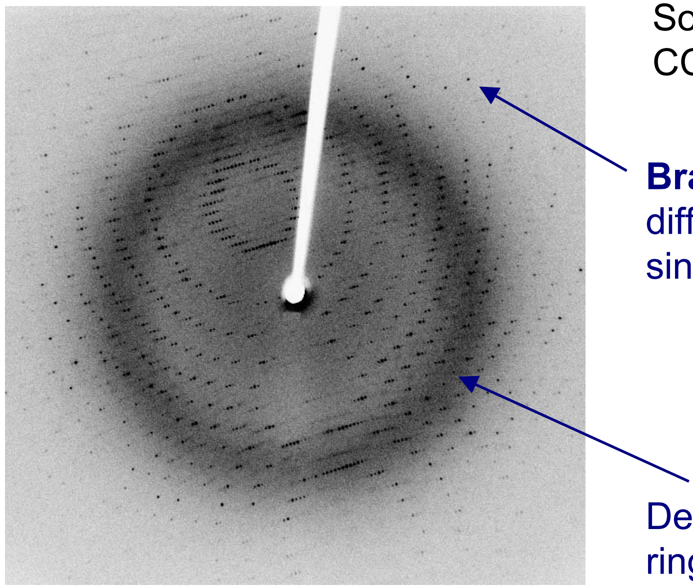
3D geometry

edgar.gutierrez-fernandez@esrf.fr

X-ray scattering techniques

- Scattering is the deflection of photons upon interaction with matter.





Source: Wikipedia

Bragg spots: diffraction from single crystal

Debye-Scherrer ring: diffraction from polycrystals

Monochromatic beam
$\Delta\lambda \approx 0$

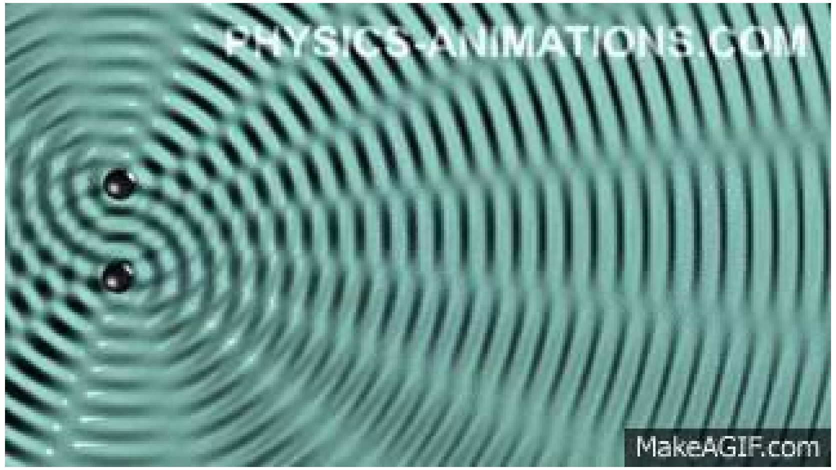
2-Dimensional detectors
Shape = (x,y)

Elastic scattering
$E_{inc} = E_{scat}$
$\Delta E = 0$

edgar.gutierrez-fernandez@esrf.fr

X-ray scattering techniques: diffraction / scattering

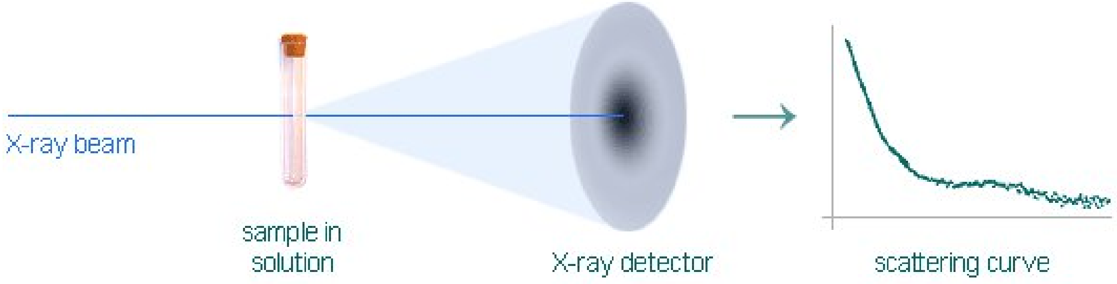
- If the material is **crystalline**, the scattered photons create constructive interferences, like water waves.



- Constructive interference between scattered X-ray takes place if Bragg relation is fulfilled:

$$n\lambda = 2d \sin\theta$$

- If the material is **disordered**, the scattered photons create broad distributions of intensity.



https://biosaxs.com/technique.html

- More ambiguous and hard to analyze. Usually requires complementary techniques.

edgar.gutierrez-fernandez@esrf.fr

X-ray scattering techniques: diffraction / scattering

- The study of highly **crystalline** materials (metals, ceramics, oxides…) is named '**diffraction**''.

- **Powder Diffraction**: isotropic
  - Phase identification
  - Crystallinity
  - Lattice parameters
  - Thermal expansion
  - Phase transition
  - Strain/crystallite size

- The study of largely/inherently **disordered** materials (polymers, proteins, colloids…) is named '**scattering**''.

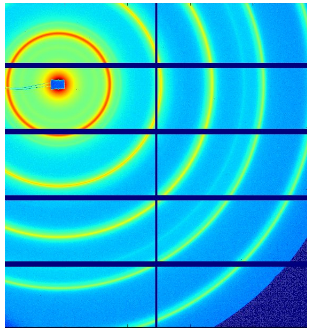
- Wide-Angle X-ray Scattering (**WAXS**): analog to diffraction:
  - Phase identification
  - Crystalllinity/orientation

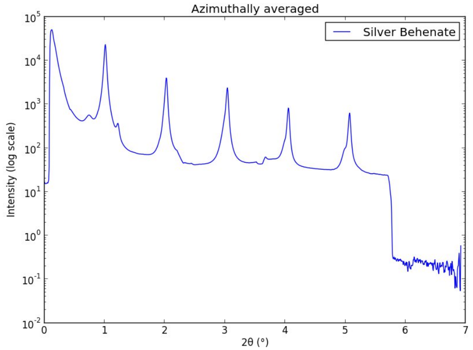
- Small-Angle X-ray Scattering (**SAXS**): micro/nano scale prove:
  - Particle shape/surface
  - Proteins domains
  - Protein folding
  - Colloid parameters
  - Fiber orientation

edgar.gutierrez-fernandez@esrf.fr

X-ray scattering techniques: diffraction / scattering

- The study of highly **crystalline** materials (metals, ceramics, oxides…) is named ‘**diffraction**’’.
- The study of largely/inherently **disordered** materials (polymers, proteins, colloids…) is named ‘**scattering**’’.
- **Powder Diffraction**: isotropic
- **WAXS**: analog to diffraction.
- **SAXS**: micro/nano scale prove.
- Both rely on the same transformation: **2D image to azimuthal average.**

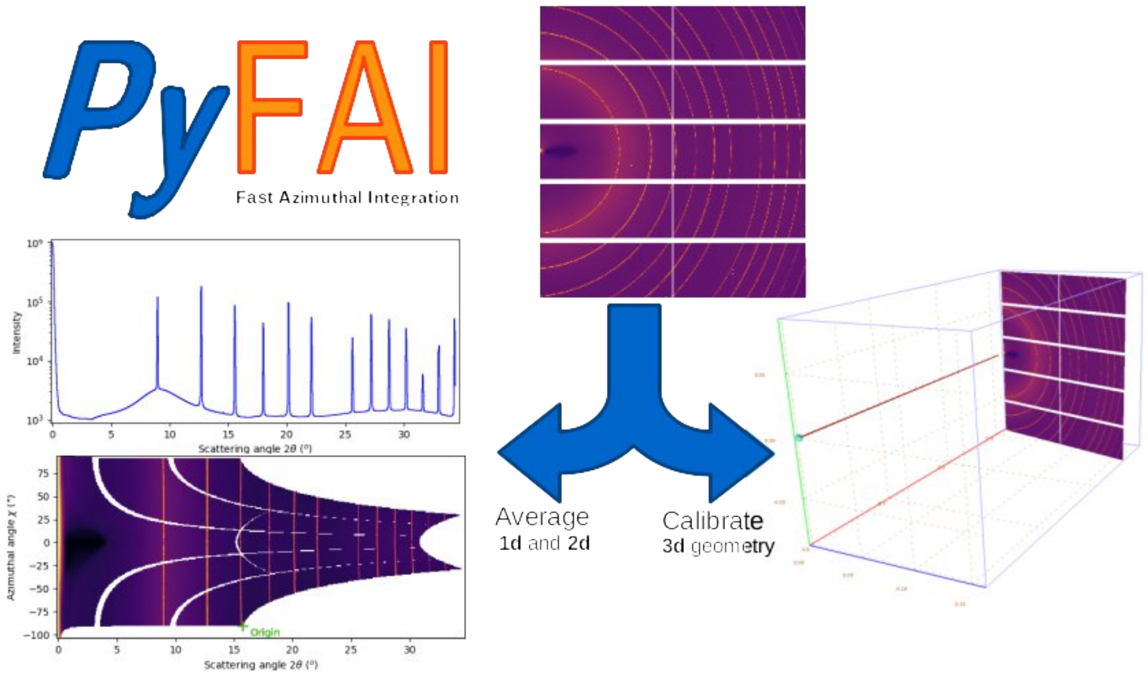




- **PyFAI is the first tool to be used after data acquisition.**

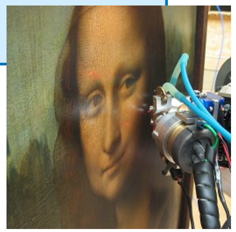
edgar.gutierrez-fernandez@esrf.fr

pyFAI: pythonic tool to reduce 2D patterns



Why pyFAI?

- Python as the most accessible and widespread programming language in science.
- Developed and generally used at the ESRF:
  - Data acquisition (BLISS)
  - Data visualization (silx)
  - Data analysis (PyMCA)



- PyFAI combines python API with fast algorithms written in **Cython and OpenCL**

edgar.gutierrez-fernandez@esrf.fr

Alternatives to pyFAI

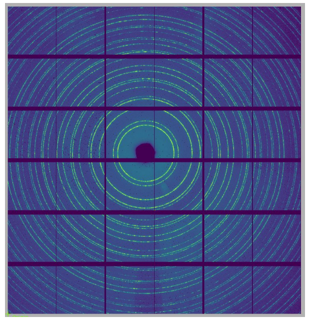
- **Fit2D**
  - MIT licensed from ESRF, written in Fortran, now discontinued
- **XRDUA**
  - GPL licensed from U. Antwerp, written in IDL, focuses on diffraction mapping
- **DAWN**
  - EPL licensed from Diamond Light Source, written in Java
- **DataSqueeze**
  - Freeware from U. Pennsylvania
- **Foxtrot**
  - Commercial, from XENOCS & SOLEIL synchrotron, written in Java
- **MAUD**
  - Freeware from U. Trento, written in Java
- **GSAS-II**
  - Freeware tool from U. Chicago & APS, written in Python
- **Scikit-beam**
  - BSD licensed from NSLS-II, written in Python

edgar.gutierrez-fernandez@esrf.fr

PyFAI: core concepts

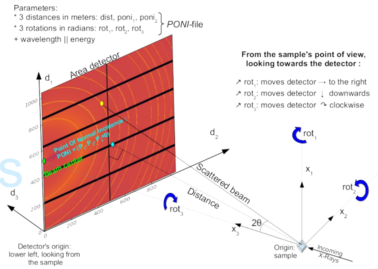
Detector
- detector_factory
- Generic detector

Image / Frame / Pattern
- Numpy 2D array
- To be read using FabIO, silx, h5py…

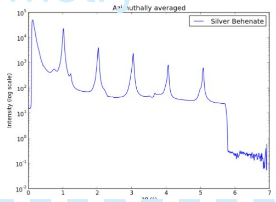


PyFAI
Fast Azimuthal Integration

Geometry
- X-ray wavelength
- Position and rotations of the detector



Integrator
- Takes the detector and geometry
- Takes the image
- Performs the integration

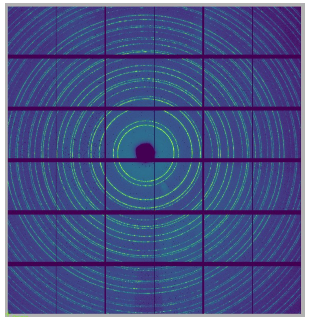


edgar.gutierrez-fernandez@esrf.fr

PyFAI: core concepts

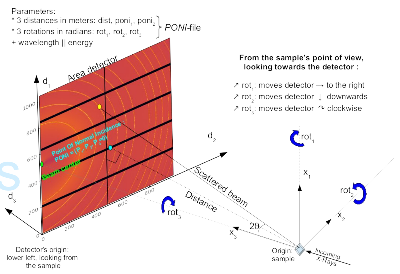
Detector
- detector_factory
- Generic detector

Image / Frame / Pattern
- Numpy 2D array
- To be read using FabIO, silx, h5py…

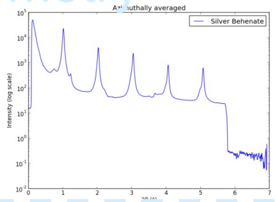


PyFAI
Fast Azimuthal Integration

Geometry
- X-ray wavelength
- Position and rotations of the detector



Integrator
- Takes the detector and geometry
- Takes the image
- Performs the integration



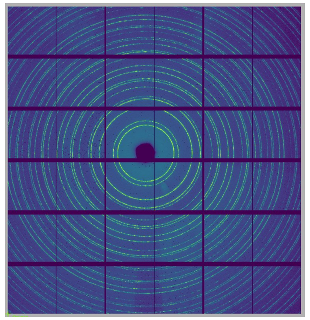
edgar.gutierrez-fernandez@esrf.fr

ESRF
The European Synchrotron

PyFAI: core concepts

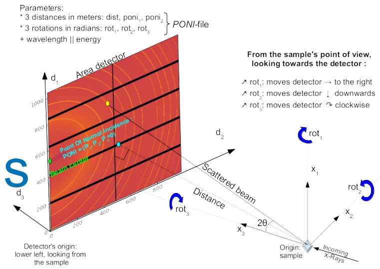
Detector
- detector_factory
- Generic detector

Image / Frame / Pattern
- Numpy 2D array
- To be read using FabIO, silx, h5py…

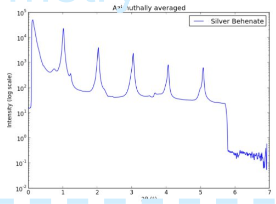


Fast Azimuthal Integration

Geometry
- X-ray wavelength
- Position and rotations of the detector



Integrator
- Takes the detector and geometry
- Takes the image
- Performs the integration

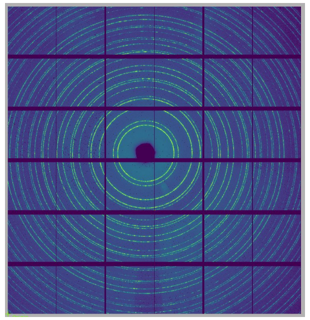


edgar.gutierrez-fernandez@esrf.fr

PyFAI: core concepts

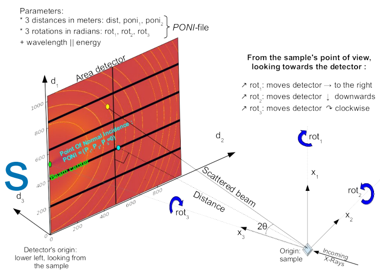
Detector
- detector_factory
- Generic detector

Image / Frame / Pattern
- Numpy 2D array
- To be read using FabIO, silx, h5py…

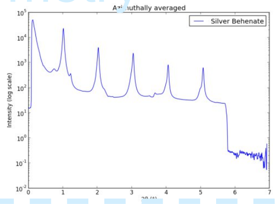


PyFAI
Fast Azimuthal Integration

Geometry
- X-ray wavelength
- Position and rotations of the detector



Integrator
- Takes the detector and geometry
- Takes the image
- Performs the integration

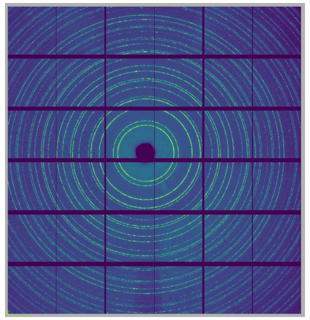


edgar.gutierrez-fernandez@esrf.fr

PyFAI: core concepts

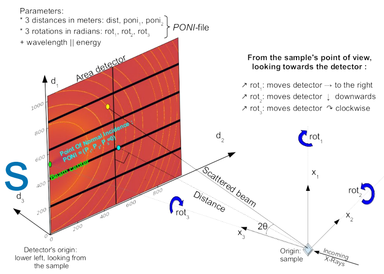
Detector
- detector_factory
- Generic detector

Image / Frame / Pattern
- Numpy 2D array
- To be read using FabIO, silx, h5py…

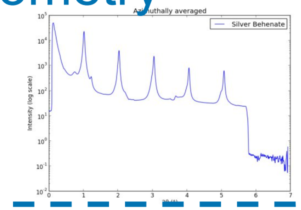


PyFAI
Fast Azimuthal Integration

Geometry
- X-ray wavelength
- Position and rotations of the detector



Integrator
- Takes the detector and geometry
- Takes the image
- Performs the integration

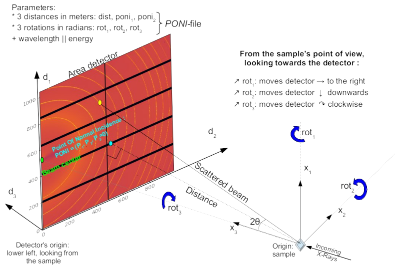


edgar.gutierrez-fernandez@esrf.fr

Integrator

Detector

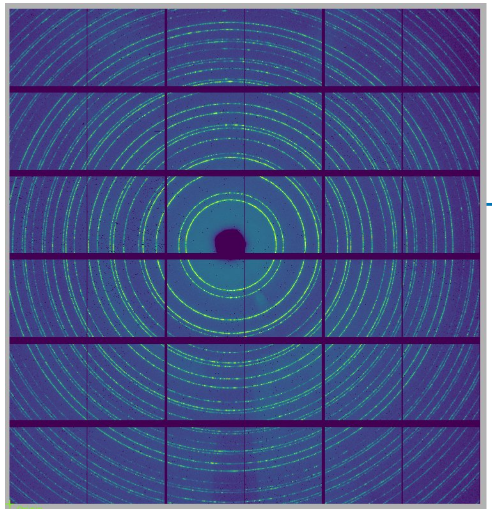
Geometry



edgar.gutierrez-fernandez@esrf.fr

PyFAI: core concepts

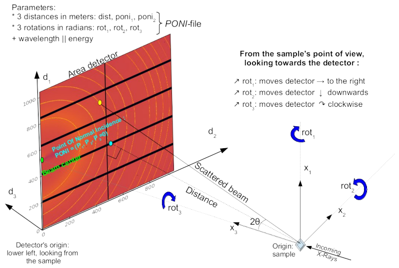
RAW DATA



Integrator

Detector

Geometry

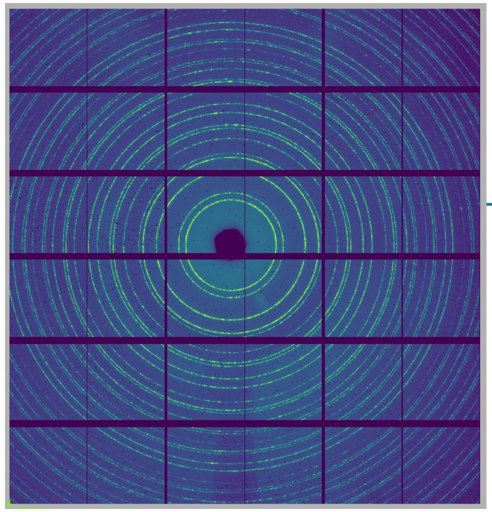


edgar.gutierrez-fernandez@esrf.fr

ESRF
The European Synchrotron

PyFAI: core concepts

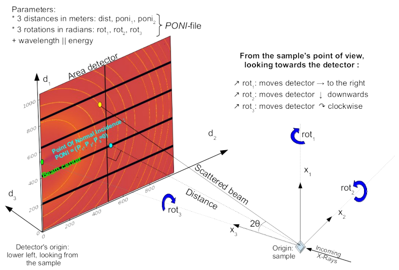
RAW DATA



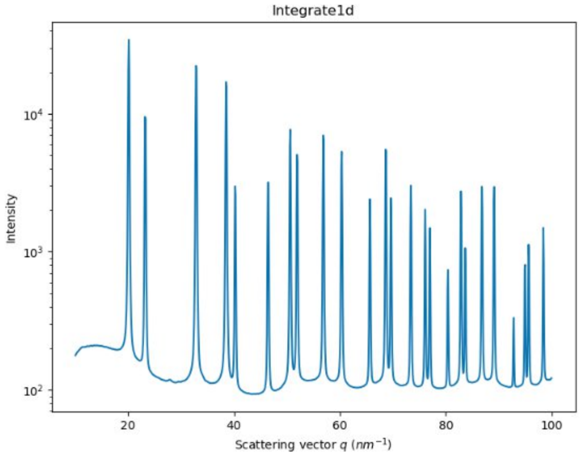
Integrator

Detector

Geometry



PROCESSED DATA



edgar.gutierrez-fernandez@esrf.fr

Detector instance

- Import detector from **detector**_factory (e.g. Pilatus1M from Dectris)

1) Detector instance: use `pyFAI.detector_factory`

Import the module

```
[3]: from pyFAI import detector_factory
```

We just need to know the name of the detector

```
[38]: my_detector = detector_factory(name="Eiger2_9M")
```

```
[39]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (3262, 3108) pixels
```

Set the binning

```
[76]: my_detector.set_binning((2,2))
```

```
[77]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (1631, 1554) pixels
```

edgar.gutierrez-fernandez@esrf.fr

Detector instance

- Import detector from **detector**_factory (e.g. Pilatus1M from Dectris)

1) Detector instance: use `pyFAI.detector_factory`

Import the module

```
[3]: from pyFAI import detector_factory
```

We just need to know the name of the detector

```
[38]: my_detector = detector_factory(name="Eiger2_9M")
```

```
[39]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (3262, 3108) pixels
```

Set the binning

```
[76]: my_detector.set_binning((2,2))
```

```
[77]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (1631, 1554) pixels
```

Detector information:

- Pixel size 1
- Pixel size 2
- Binning: default (1,1)
- Max_shape
- Orientation (1→4)

edgar.gutierrez-fernandez@esrf.fr

Detector instance

- Import detector from **detector**_factory (e.g. Pilatus1M from Dectris)

1) Detector instance: use `pyFAI.detector_factory`

Import the module

```
[3]: from pyFAI import detector_factory
```

We just need to know the name of the detector

```
[38]: my_detector = detector_factory(name="Eiger2_9M")
```

```
[39]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (3262, 3108) pixels
```

Set the binning

```
[76]: my_detector.set_binning((2,2))
```

```
[77]: print(f"My detector is {my_detector.name}\nIts shape is {my_detector.shape} pixels")
```

```
My detector is Eiger2 9M
Its shape is (1631, 1554) pixels
```

Detector information:

- Pixel size 1
- Pixel size 2
- Binning: default (1,1)
- Max_shape
- Orientation (1→4)

Most of the time, by knowing the name of your detector is enough!

edgar.gutierrez-fernandez@esrf.fr

Detector instance

- Import detector from **detector**_factory (e.g. Pilatus1M from Dectris)

- Customized (generic) detector

```
[5]: det = detector_factory(
         name="Detector",
         config={
             "max_shape" : (2000,2000),
             "pixel1" : 50e-6,
             "pixel2" : 50e-6,
             "orientation" : 3,
     })
```

```
[6]: print(f"""
     The detector {det.name} has a shape of {det.shape}
     The size of the pixels is {det.pixel1*1e6} microns x {det.pixel2*1e6} microns
     """)
```

```
The detector Detector has a shape of (2000, 2000)
The size of the pixels is 50.0 microns x 50.0 microns
```

Detector information:

- Pixel size 1
- Pixel size 2
- Binning: default (1,1)
- Max_shape
- Orientation (1→4)

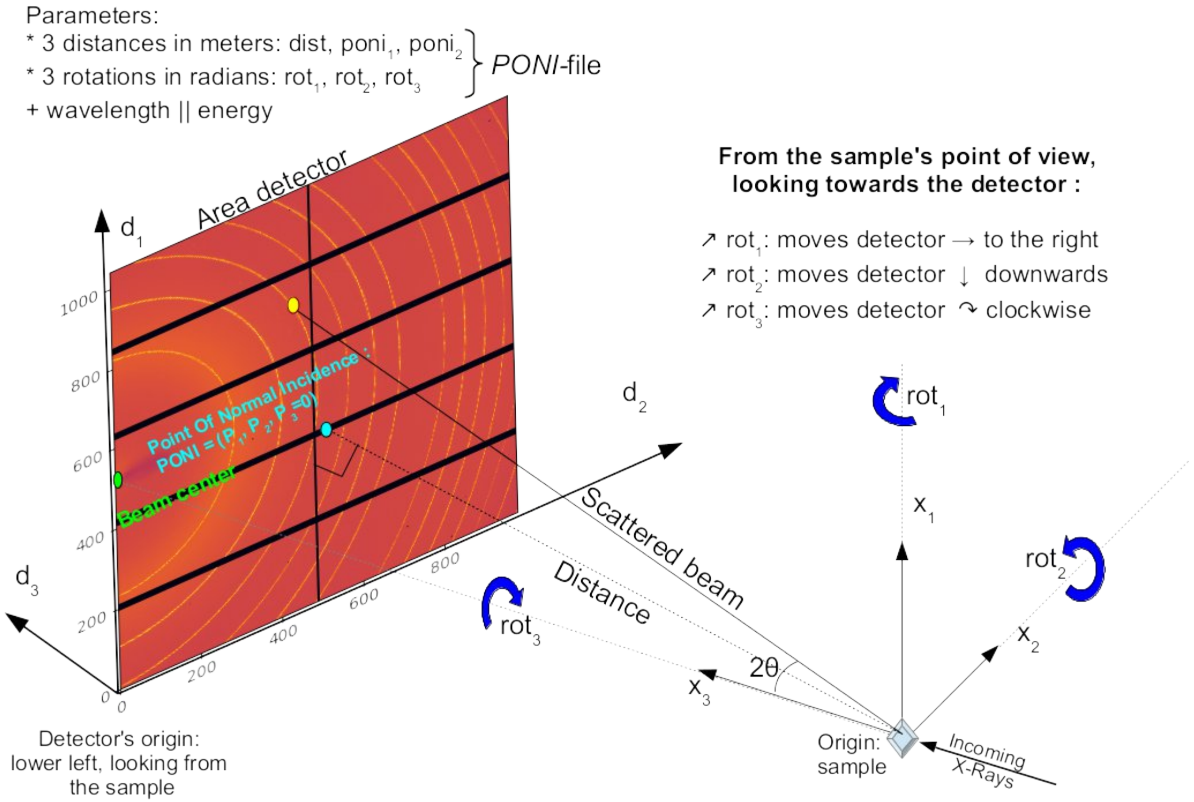
Normally, this step is skipped as it is done through the calibration graphical interface

Bet: > 50% pyFAI crashes are related to wrong detector shapes

edgar.gutierrez-fernandez@esrf.fr

Geometry instance

- A geometry is fully defined by:
  - A **detector** instance
  - Sample to detector **distance** (in meters)
  - **Wavelength** of the beam (in meters)
  - Three **rotations** of the detector
  - Coordinates of the point of normal incidence (**PONI**), from the sample to the detector plane.



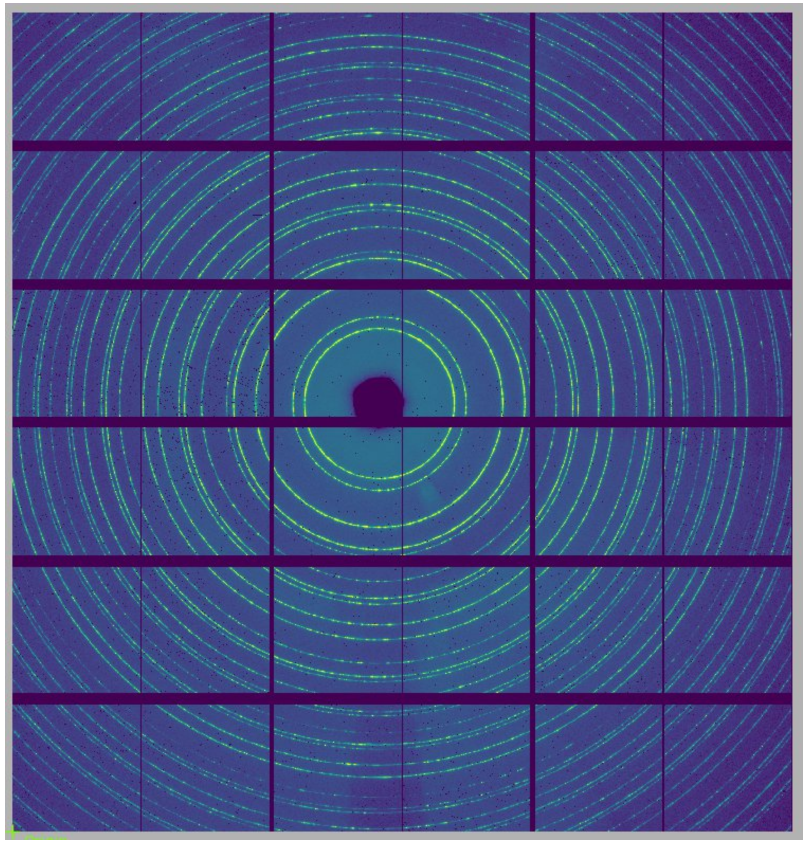
Normally, the calibration of the geometry is fully done through the graphical interface.

edgar.gutierrez-fernandez@esrf.fr

Calibration of geometry: pyFAI-calib2

- Calibration is made after measuring a standard sample:
  - LaB6, Cr2O3, AgBh…
- Choosing the correct detector (+ orientation, binning, mask…)
- Selecting the Debye-Scherrer rings associated to the standard
- Fitting the rings
- Refinement
- Validation
- Saving of .poni file

.poni file



edgar.gutierrez-fernandez@esrf.fr

Calibration of geometry: pyFAI-calib2

- Calibration is made after measuring a standard sample:
  - LaB6, Cr2O3, AgBh…
- Choosing the correct detector (+ orientation, binning, mask…)
- Selecting the Debye-Scherrer rings associated to the standard
- Fitting the rings
- Refinement
- Validation
- Saving of .poni file

.poni file

home > edgar > eiger9m_bin.poni

```
# Nota: C-Order, 1 refers to the Y axis, 2 to the X axis
# Calibration done on Sat Feb  8 15:34:34 2025
poni_version: 2.1
Detector: Eiger2_9M
Detector_config: {"orientation": 3}
Distance: 0.1850492110887503
Poni1: 0.13007134675074164
Poni2: 0.1124624632248358
Rot1: 0.014602501624582428
Rot2: -0.012061870930842628
Rot3: 0.0
Wavelength: 3.75709692221819e-11
# Calibrant: CeO2
# Image: Eiger2_bin2_CeO2_33keV.h5
```

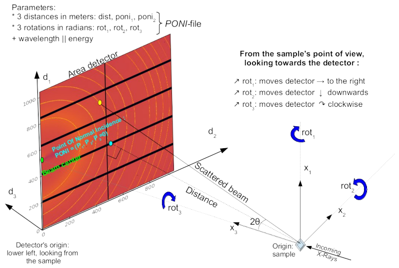
edgar.gutierrez-fernandez@esrf.fr

PyFAI: creating an integrator

Integrator

Detector

Geometry



edgar.gutierrez-fernandez@esrf.fr

PyFAI: creating an integrator

Integrator

home > edgar > eiger9m_bin.poni

```
# Nota: C-Order, 1 refers to the Y axis, 2 to the X axis
# Calibration done on Sat Feb  8 15:34:34 2025
poni_version: 2.1
Detector: Eiger2_9M
Detector_config: {"orientation": 3}
Distance: 0.1850492110887503
Poni1: 0.13007134675074164
Poni2: 0.1124624632248358
Rot1: 0.014602501624582428
Rot2: -0.012061870930842628
Rot3: 0.0
Wavelength: 3.75709692221819e-11
# Calibrant: CeO2
# Image: Eiger2_bin2_CeO2_33keV.h5
```

edgar.gutierrez-fernandez@esrf.fr

PyFAI: creating an integrator

Integrator

```
home > edgar > eiger9m_bin.poni
# Nota: C-Order, 1 refers to the Y axis, 2 to the X axis
# Calibration done on Sat Feb  8 15:34:34 2025
poni_version: 2.1
Detector: Eiger2_9M
Detector_config: {"orientation": 3}
Distance: 0.1850492110887503
Poni1: 0.13007134675074164
Poni2: 0.1124624632248358
Rot1: 0.014602501624582428
Rot2: -0.012061870930842628
Rot3: 0.0
Wavelength: 3.75709692221819e-11
# Calibrant: CeO2
# Image: Eiger2_bin2_CeO2_33keV.h5
```

After using `pyFAI-calib2`, we load the `poni` file

[130]:
```python
from pyFAI import load
file_poni = "eiger9m_bin.poni"
ai = load(filename=file_poni)
ai.detector = my_detector
```

[131]:
```python
print(f"Information from the AzimuthalIntegrator:\n\n {ai}")
```

```
Information from the AzimuthalIntegrator:

 Detector Eiger2 9M      PixelSize= 150µm, 150µm         BottomRight (3)
Wavelength= 3.757097e-11 m
SampleDetDist= 1.850492e-01 m   PONI= 1.300713e-01, 1.124625e-01 m      rot1=0.014603
rot2=-0.012062  rot3=0.000000 rad
DirectBeamDist= 185.082 mm      Center: x=731.734, y=852.260 pix        Tilt= 1.085°
tiltPlanRotation= -140.440° λ= 0.376Å
```

edgar.gutierrez-fernandez@esrf.fr

PyFAI: creating an integrator

Integrator

home > edgar > eiger9m_bin.poni

```
# Nota: C-Order, 1 refers to the Y axis, 2 to the X axis
# Calibration done on Sat Feb  8 15:34:34 2025
poni_version: 2.1
Detector: Eiger2_9M
Detector_config: {"orientation": 3}
Distance: 0.1850492110887503
Poni1: 0.13007134675074164
Poni2: 0.1124624632248358
Rot1: 0.014602501624582428
Rot2: -0.012061870930842628
Rot3: 0.0
Wavelength: 3.75709692221819e-11
# Calibrant: CeO2
# Image: Eiger2_bin2_CeO2_33keV.h5
```

After using `pyFAI-calib2`, we load the `poni` file

[130]:
```python
from pyFAI import load
file_poni = "eiger9m_bin.poni"
ai = load(filename=file_poni)
ai.detector = my_detector
```

[131]:
```python
print(f"Information from the AzimuthalIntegrator:\n\n {ai}")
```

```
Information from the AzimuthalIntegrator:

 Detector Eiger2 9M      PixelSize= 150µm, 150µm         BottomRight (3)
Wavelength= 3.757097e-11 m
SampleDetDist= 1.850492e-01 m    PONI= 1.300713e-01, 1.124625e-01 m      rot1=0.014603
rot2=-0.012062  rot3=0.000000 rad
DirectBeamDist= 185.082 mm      Center: x=731.734, y=852.260 pix        Tilt= 1.085°
tiltPlanRotation= -140.440° λ= 0.376Å
```

Novelty in `pyFAI 2025.01`: `FiberIntegrator`

[132]:
```python
fiber_integrator = load(filename=file_poni,
                        type_="pyFAI.integrator.fiber.FiberIntegrator",
)
```

[133]:
```python
print(f"Information from the FiberIntegrator:\n\n {fiber_integrator}")
```

```
Information from the FiberIntegrator:

 Detector Eiger2 9M      PixelSize= 75µm, 75µm   BottomRight (3)
Wavelength= 3.757097e-11 m
SampleDetDist= 1.850492e-01 m    PONI= 1.300713e-01, 1.124625e-01 m      rot1=0.014603
rot2=-0.012062  rot3=0.000000 rad
DirectBeamDist= 185.082 mm      Center: x=1463.468, y=1704.519 pix      Tilt= 1.085°
tiltPlanRotation= -140.440° λ= 0.376Å
Incident angle: 0.00° Tilt angle 0.00° Sample orientation 1
```

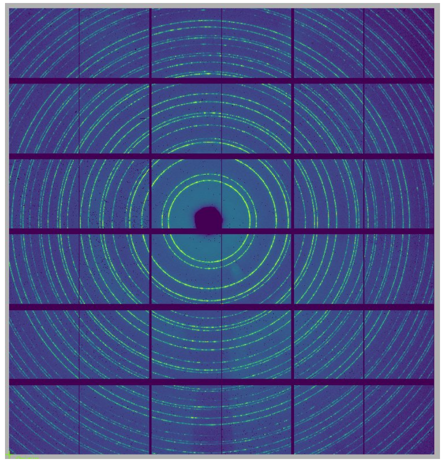
***FiberIntegrator is targeted to Grazing Incidence experiments**

edgar.gutierrez-fernandez@esrf.fr

PyFAI: loading data

Integrator

RAW DATA



home > edgar > eiger9m_bin.poni

```
# Nota: C-Order, 1 refers to the Y axis, 2 to the X axis
# Calibration done on Sat Feb  8 15:34:34 2025
poni_version: 2.1
Detector: Eiger2_9M
Detector_config: {"orientation": 3}
Distance: 0.1850492110887503
Poni1: 0.13007134675074164
Poni2: 0.1124624632248358
Rot1: 0.014602501624582428
Rot2: -0.012061870930842628
Rot3: 0.0
Wavelength: 3.757096922218l9e-11
# Calibrant: CeO2
# Image: Eiger2_bin2_CeO2_33keV.h5
```

edgar.gutierrez-fernandez@esrf.fr

PyFAI: loading data

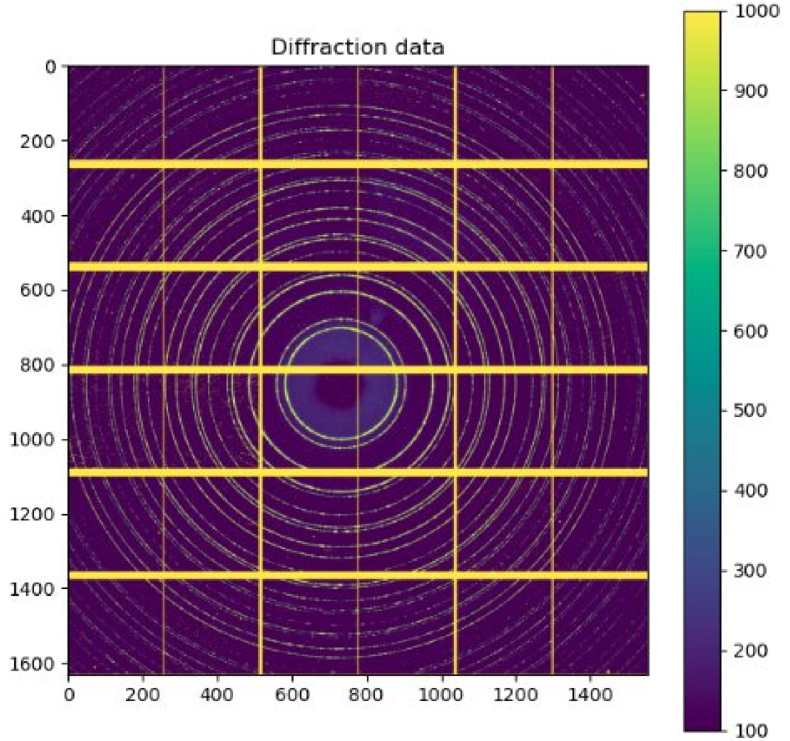
- Importing data is made through other python packages:
  - FabIO
  - Silx
  - h5py
- Common file formats:
  - **.edf**
  - .tiff
  - .h5
- Visualizing is made through:
  - matplotlib
  - seaborn
  - silx

3) Load `.h5` data with `h5py`

[108]:
```python
with h5py.File(name="Eiger2_bin2_CeO2_33keV.h5", mode="r") as f:
    data = f["/entry_0000/measurement/data"][0]
```

Plot the data with `matplotlib`

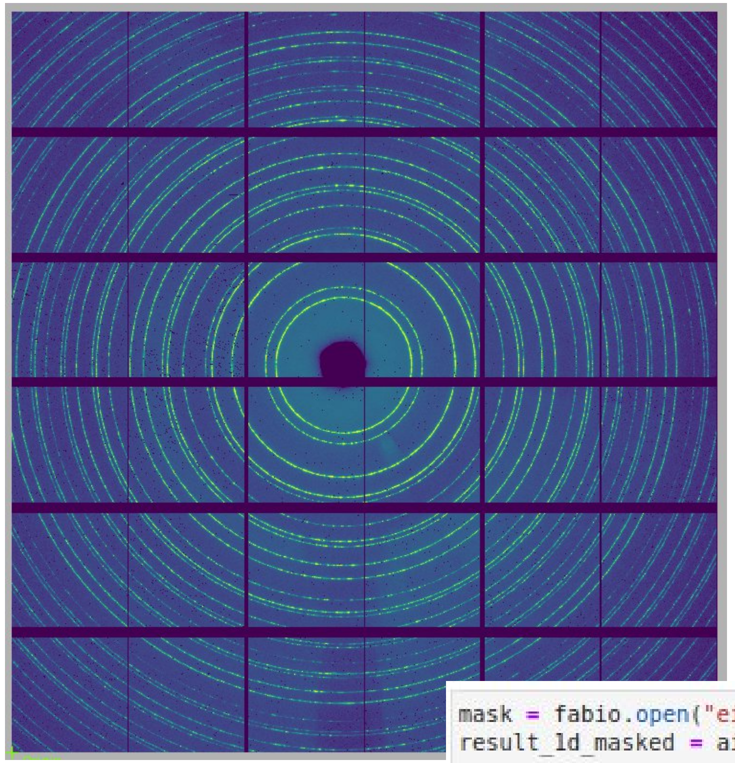
[109]:
```python
fig, ax = plt.subplots(figsize=(7,7))
img = ax.imshow(data, cmap="viridis", vmin=100, vmax=1000)
fig.colorbar(img)
ax.set_title("Diffraction data")
pass
```



edgar.gutierrez-fernandez@esrf.fr

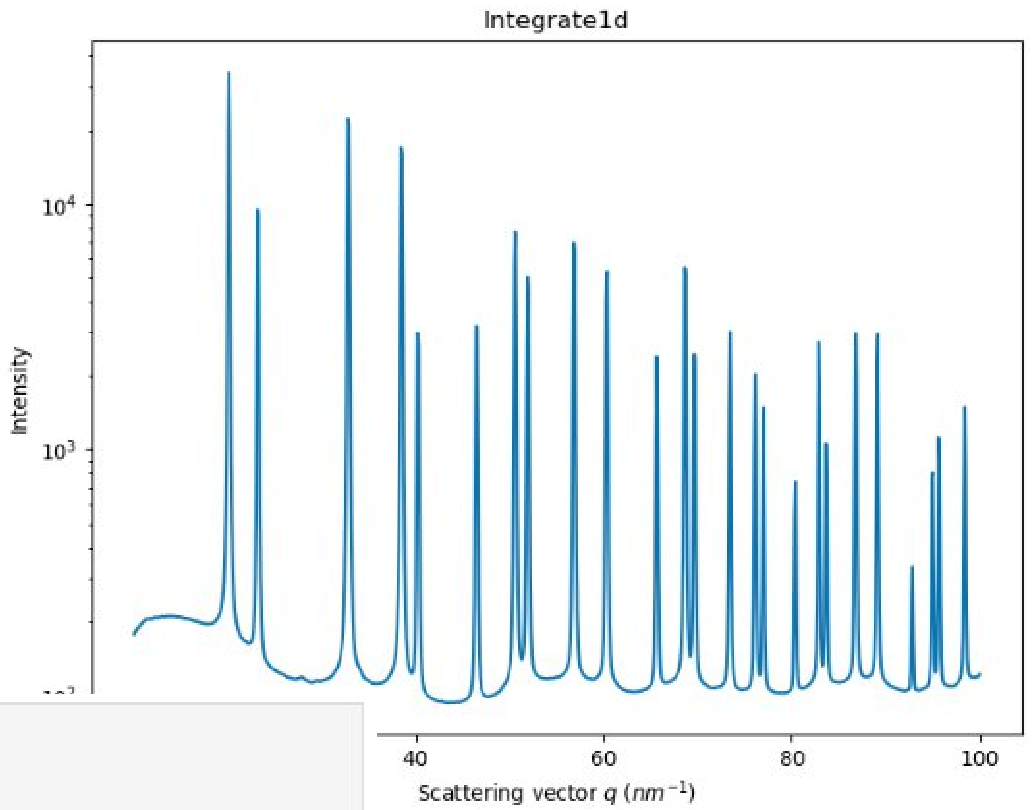
PyFAI: loading data

RAW DATA



Integrator

integrate1d



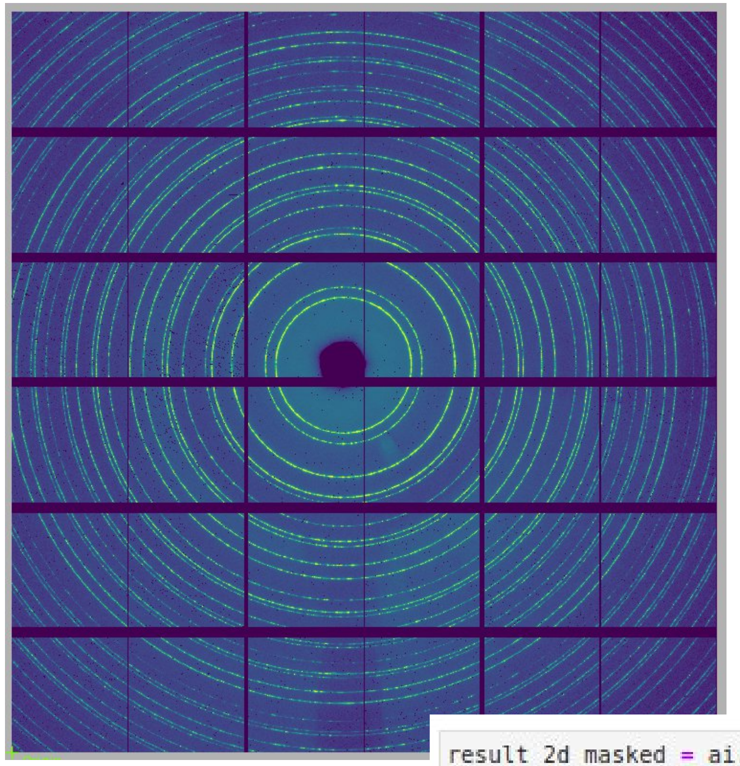

```python
mask = fabio.open("eiger9_mask.edf").data
result_1d_masked = ai.integrate1d(data=data,
                                  npt=1000,
                                  mask=mask,
                                  radial_range=[10,100],
                                 )
```

```python
fig, ax = plt.subplots(figsize=(8,6))
jupyter.plot1d(result=result_1d_masked,
               ax=ax,
)
ax.set_title("Integrate1d")
ax.set_yscale("log")
pass
```

edgar.gutierrez-fernandez@esrf.fr

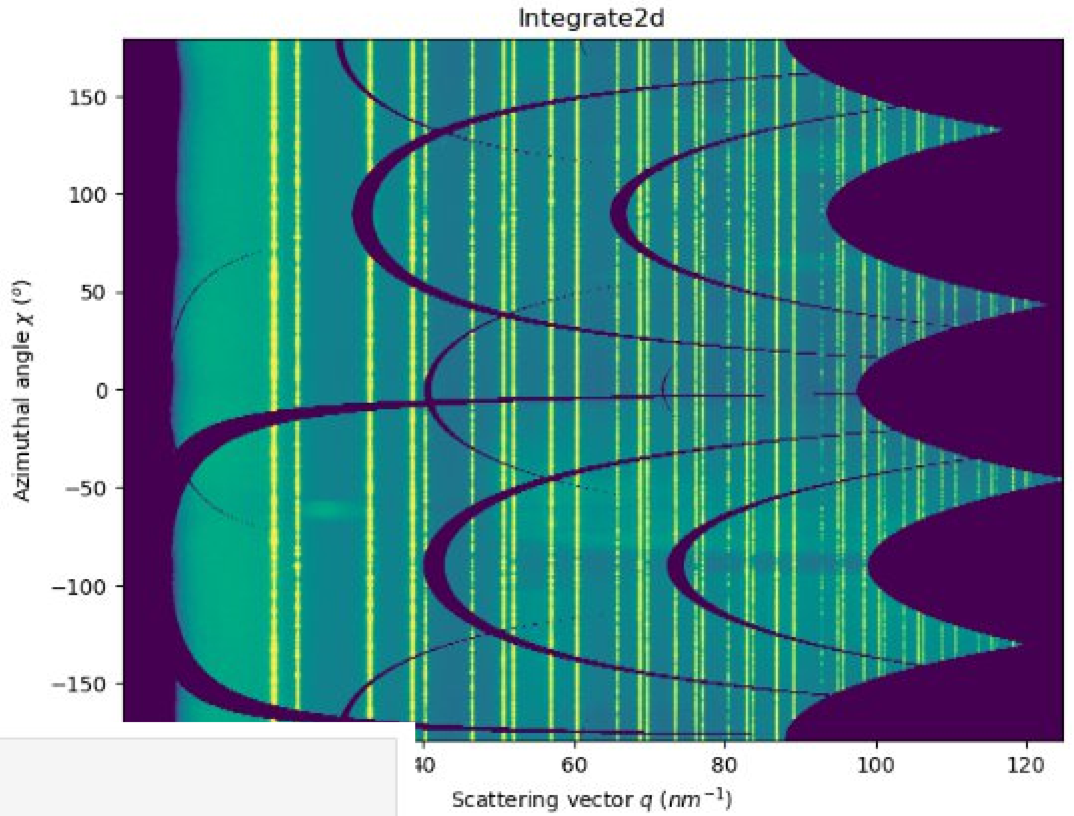
PyFAI: loading data

RAW DATA



Integrator

integrate2d




```
result_2d_masked = ai.integrate2d(data=data,
                                  npt_rad=1000,
                                  npt_azim=360,
                                  mask=mask,
                                 )
```

```
fig, ax = plt.subplots(figsize=(8,6))
jupyter.plot2d(result=result_2d_masked,
               ax=ax,
)
ax.set_title("Integrate2d")
pass
```

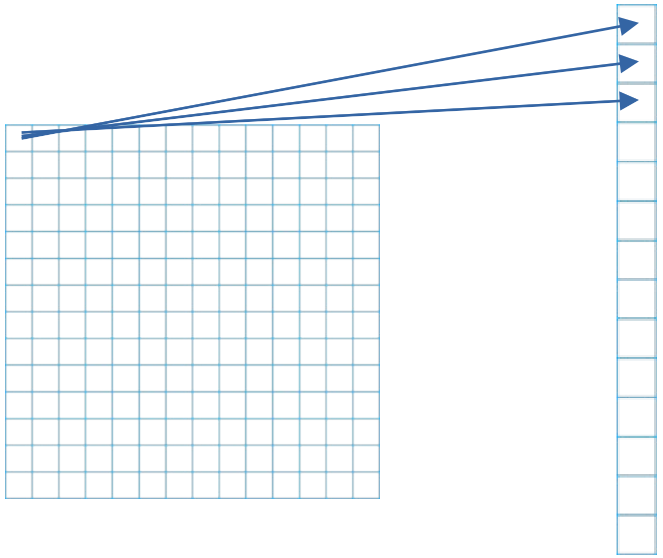
edgar.gutierrez-fernandez@esrf.fr

What happens during an integration

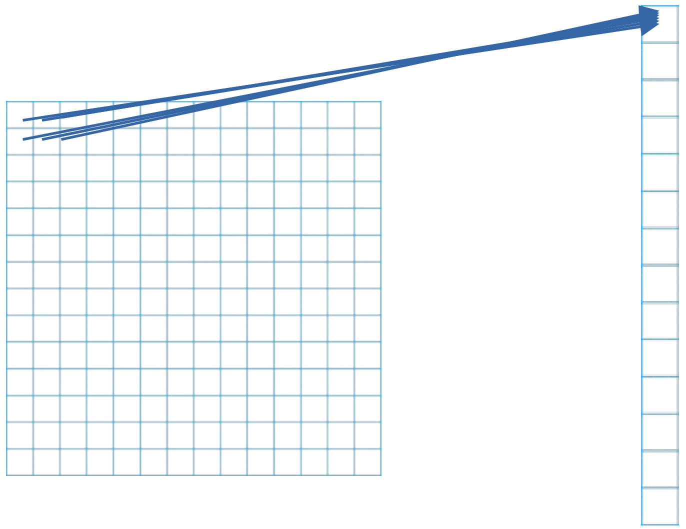
- 1) Get the coordinates of every **corner of every pixel of the detector** (in meters).
  - 3 coordinates per corner, 4 corners per pixel
  - Detector of 1000x1000 = 10^6 pixels = 1Mpix * 4 (corners) * 3 (coordinates) * 4 (bytes) = 48 Mbytes
- 2) Offset the **detector's origin** to the PONI and **rotate** around the sample.
- 3) Calculate the **radial (2theta) and azimuthal (chi) positions of each corner.**
- 4) Calculate **normalization** matrix: polarization, solid-angle, flat-field...
- 5) Assign each pixel to one or multiple **bins**.
- 6) Histogram bin position with **associated intensities**
- 7) Histogram bin position with **associated normalizations**
- 8) Return bin position and the ratio of **sum of intensities / sum of norm.**

Integration algorithms

- **Histogram**
  - Pixel by pixel procedure.
  - Each pixel is split over the bins it covers.
  - Corner coordinates have to be calculated (4x slower initialization).
  - The slow down is function of the oversampling factor, for every image.
  - Serial read → Random write



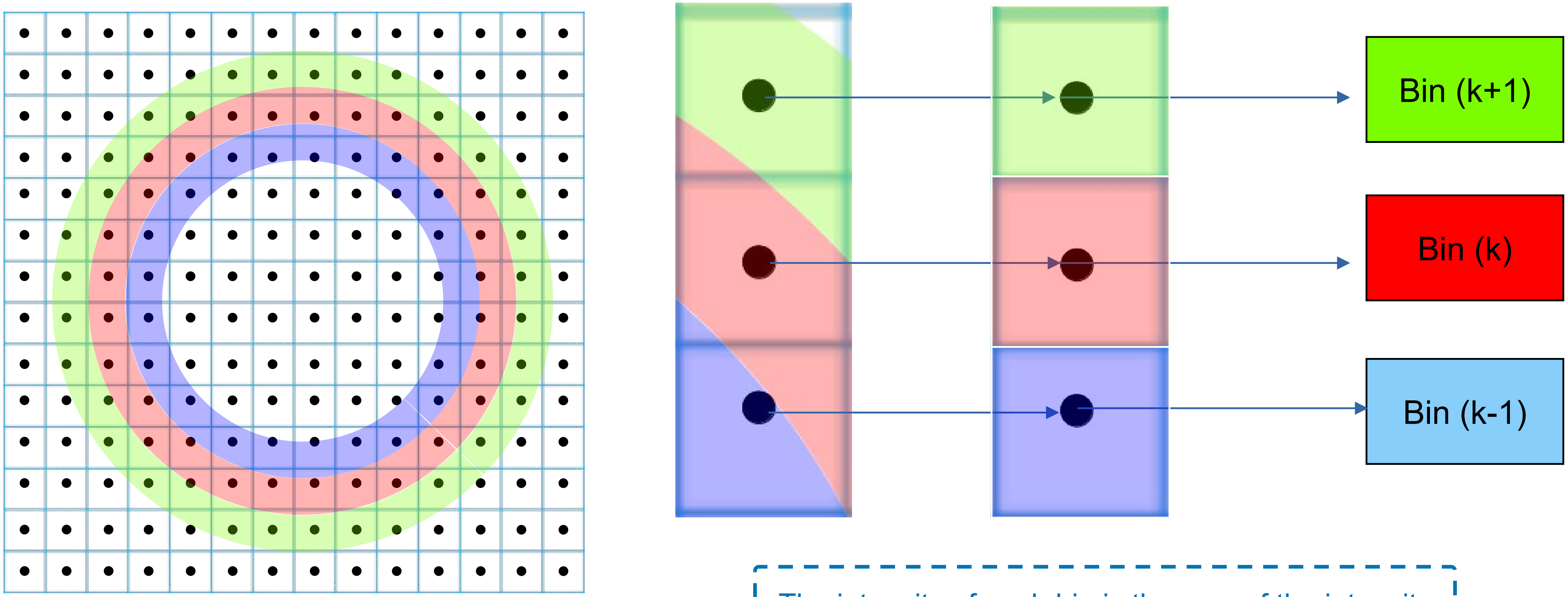
- **Sparse Matrix Multiplication**
  - Bin by bin procedure.
  - Creates and stores a sparse matrix with all the integration information.
  - The sparse matrix can be huge: longer initialization related to the oversampling factor.
  - No performance penalty on the integration itself.
  - Serial write ← Random read



edgar.gutierrez-fernandez@esrf.fr

Pixel splitting

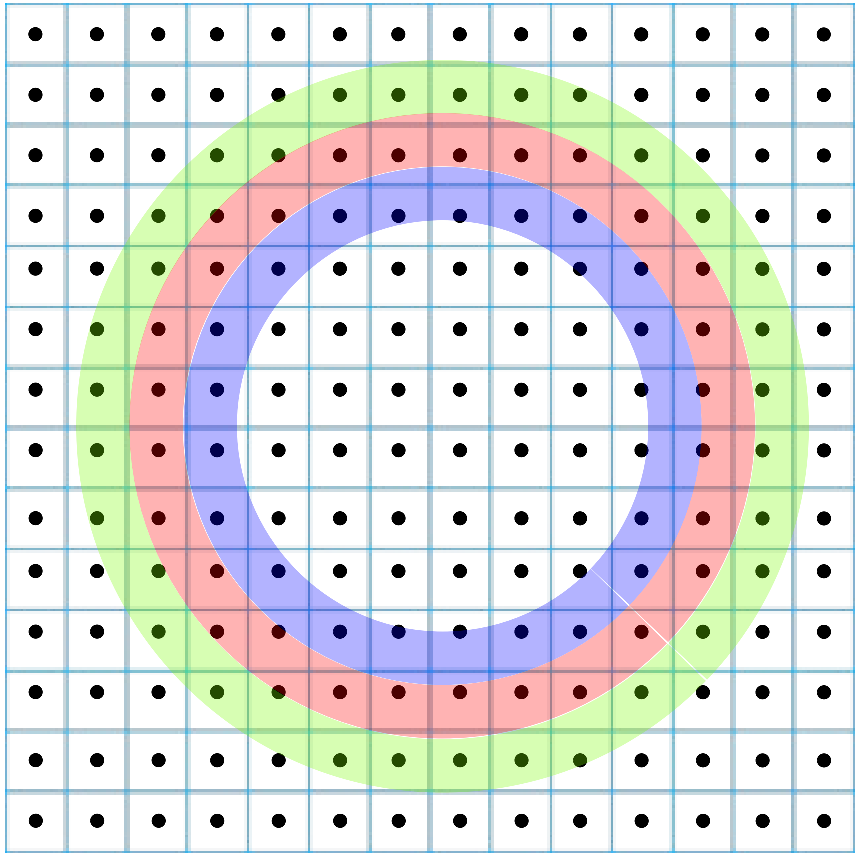
- **No splitting**: one pixel to one single bin upon in which bin the center of the pixel is falling.



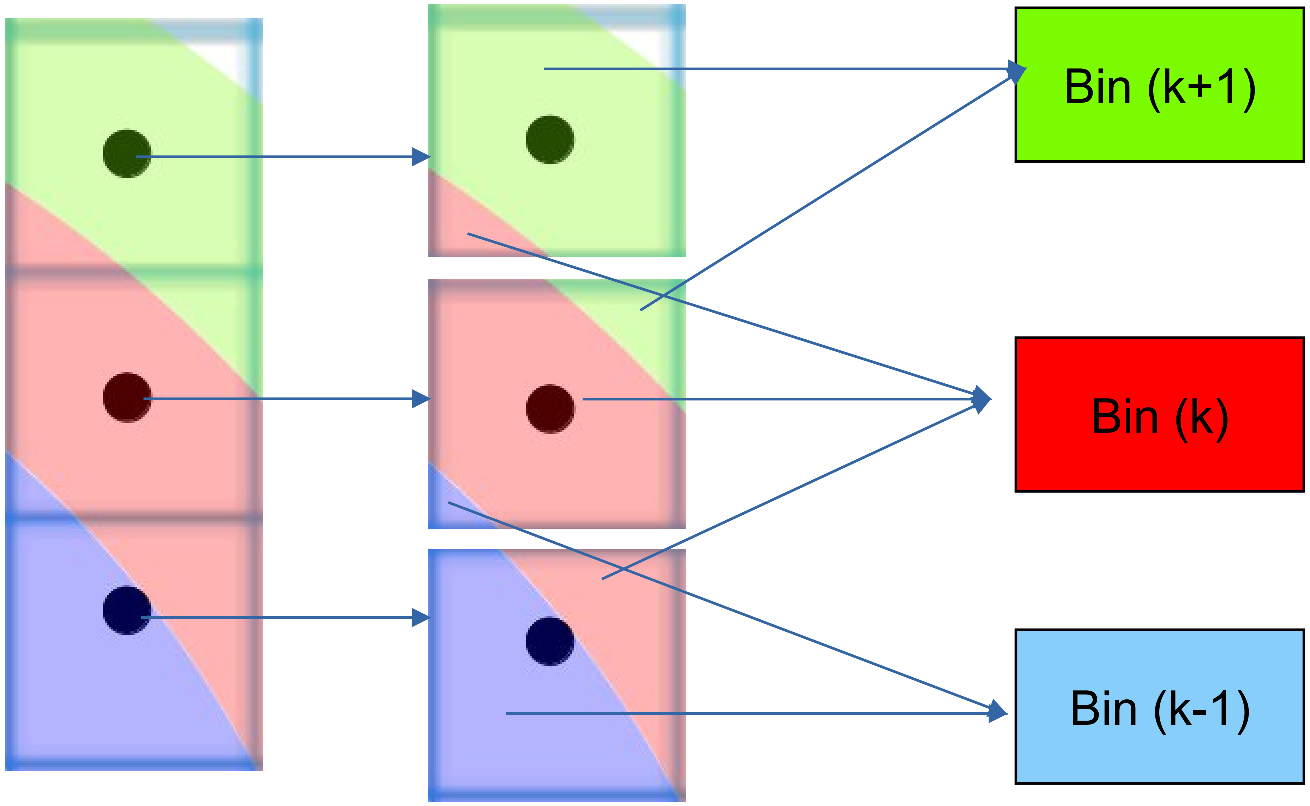
The intensity of each bin is the sum of the intensity of the pixels whose center falls into the radius bin

Pixel splitting

- **Splitting**: each pixel intensity is shared between consecutive bins.



*Full pixel splitting



Each bin is the sum of the intensities of different pixels multiplied by a weight between 0.0 and 1.0

edgar.gutierrez-fernandez@esrf.fr

Implementation

Python

- Use numpy methods
- No initialization
- Slow, no cache
- Non-recommended (not popular)

Cython

- Use cython methods
- No initialization
- Faster than python, no cache
- Recommended to integrate 10s frames

OpenCL

- Use parallelization through CPU/GPU
- Initialization ~1-3 s
- Cache
- Potentially fast, depending on the GPU
- Best option for large data

```
[15]: # Method = ("split-pixel scheme, algorithm, implementation")
method_default = ("bbox", "csr", "cython")
method_fast = ("bbox", "csr", "opencl")
```

edgar.gutierrez-fernandez@esrf.fr

pyFAI interfaces

- Applications:
  - GUI applications: pyFAI-calib2, pyFAI-integrate, pyFAI-diffmap-view
  - Batch mode: worker
  - Scriptable applications: pyFAI-average, pyFAI-saxs, pyFAI-waxs, diff_tomo

- Python interface:
  - High level, direct API: scripts, Jupyter notebook
  - Mid level: manually creation of detectors, integrators, units…
  - Low level: manually setting the integrator engines

It is up to the user to choose the way he/she uses pyFAI

edgar.gutierrez-fernandez@esrf.fr

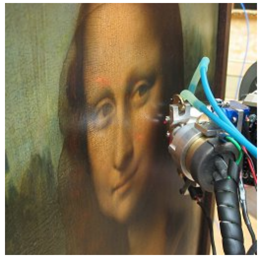
Latest news from pyFAI

- **PyFAI-2025.1.0: release on 31/01/2024**
- Median filter (cython, OpenCL)
- Unified WorkerConfiguration
- New API for Grazing Incidence experiments

edgar.gutierrez-fernandez@esrf.fr

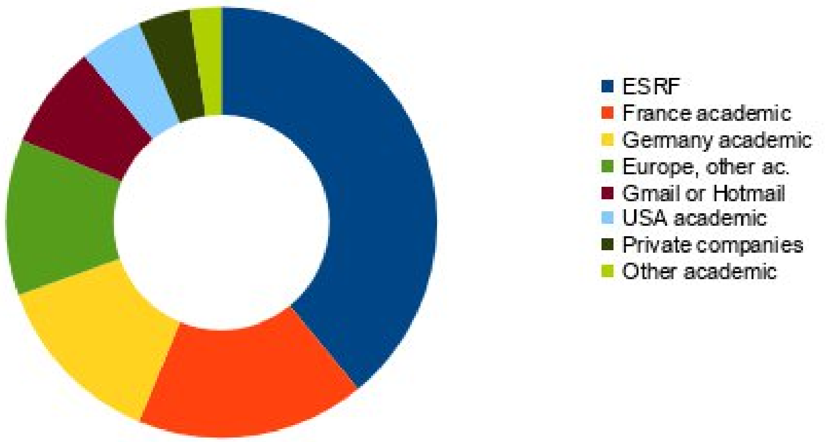
Project management: Silx & pyFAI

- Compatible with Windows, MacOS, Linux
- MIT licensed: compatible with both science and business
- **PyFAI** is embedded in the silx-kit project: https://github.com/silx-kit/
- **Silx-kit** project is python-based, developed at the ESRF and includes:
- Open to collaborations:
  - About 20 direct contributors from ESRF, from other synchrotrons, XFELs (Soleil, NSLS-II, Petra-III, Eu-XFEL) and companies (Xenocs)
  - Used by ~90 other projects from many other X-ray sources in the world (SLAC, ALS, APS, ALBA, NSLS-II, Petra-III, Soleil, Diamond, SLS, MaxIV...)



pyFAI user community

- **PyFAI is used in most European and American synchrotrons/FELs**



- **User support is provided via the mailing list: pyFAI@esrf.fr (185 subscribers)**
- **Bugs are discussed via Github issue tracker**
  - https://github.com/silx-kit/pyFAI/issues

Reasons to choose pyFAI

- **Faster** than others
  - First tool using sparse matrix multiplication to perform integration
  - All computation intensive kernels are ported to C, C++ or OpenCL
  - PyFAI is the only azimuthal integration tool benefiting from GPU

- **Versatile** (increasing with every version)
  - Wide space to vary the integration protocol
  - Generic geometry
  - Most detectors already defined (+ custom detector through Nexus file)
  - Graphical user interface alternatives (thanks to Valentin Valls)

Acknowledgements

- **Main author: Jerome Kieffer**
- **Contributors: 43**
- Former data analysis unit colleagues:
  - Valentin Valls
  - Loic Huder
  - Thomas Vincent
  - Claudio Ferrero
- ESRF Beamlines:
  - BM01, BM02, ID02, ID11, ID13, ID15a, ID15b, ID21, ID22, ID23, BM26, ID27, BM28, ID28, BM29, ID29, ID30, ID31...
- Trainees:
  - Aurore Deschildre
  - Frederic Sulzmann
  - Guillaume Bonamis
- Other synchrotron/labs
  - Soleil: Fred Picca
  - Clemens Prescher (Dioptas)
  - Sesame: Philipp Hans
  - NSLS-II, ALS, APS…
- International Grants:
  - LinkSCEEM-2 grant:
    - Dimitris Karkoulis
    - Giannis Ashiotis
    - Zubair Nawaz

edgar.gutierrez-fernandez@esrf.fr

ESRF User Meeting 2025
PyFAI tutorial

Jérôme Kieffer

Edgar Gutiérrez Fernández

10/02/2025