Fast 2D azimuthal integration using PyFAI

ESRF | The European Synchrotron

Jérôme Kieffer, online data analysis @ ESRF

Introduction to pyFAI

- **X-ray scattering/diffraction experiments using 2D detectors:**
  - X-ray scattering by electrons
  - Benefits of using 2D detectors
- **Azimuthal integration in python**
  - How azimuthal integration is performed in Python
  - What is PyFAI: installation, testing and basic use
  - Improvements provided by pyFAI
    - **Pixel splitting algorithms**
    - **Parallelization of the algorithms**

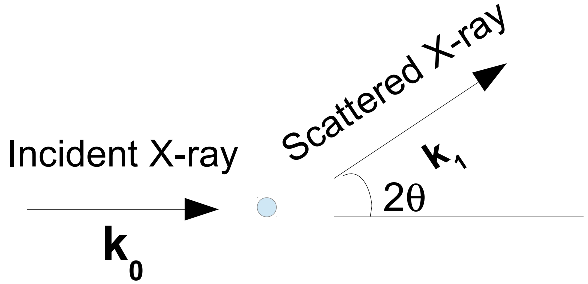
    by Giannis Ashiotis
  - Scripts to perform integration, plugins for EDNA, LimA, ...
- **Calibration of the experimental setup**
  - Peak picking methods & geomety optimization setup
- **Extra tools**

by Aurore Deschildre

X-ray scattering and 2D detectors

X-ray scattering

- **Scattering by an electron:**



  - $\boldsymbol{q} = \boldsymbol{k}_1 - \boldsymbol{k}_0$
  - $A = b\, exp(-2i\pi\, (\boldsymbol{k}_1 - \boldsymbol{k}_0).\boldsymbol{r}/\lambda)$
    where b is the scattering cross-section of an electron (Thomson or elastic scattering)

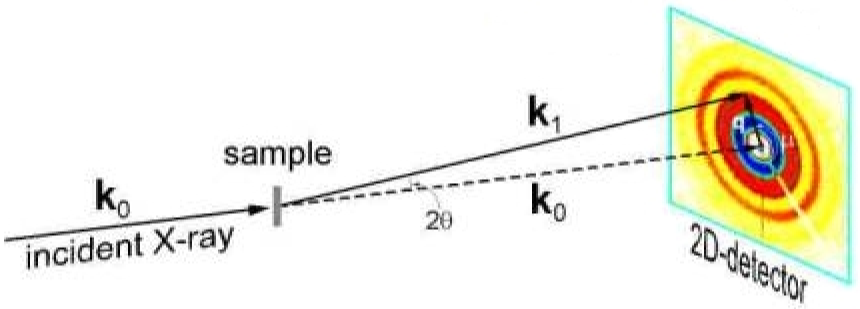
  The cross section of an atom or scattering density is increasing with Z, the number of electrons

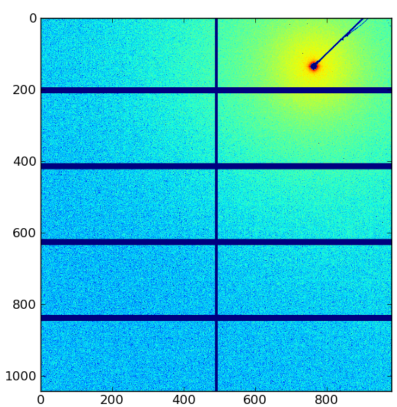
- Intensity of scattering: Only the norm of the scattered intensity is accessible:
  - $I(\mathbf{q}) = AA^* = \Sigma_j \Sigma_k b_j b_k\, exp(4\pi(\boldsymbol{k}_1 - \boldsymbol{k}_0).(\boldsymbol{r}_j\text{-}\boldsymbol{r}_k)/\lambda)$
    - *Sum over all pairs of electrons*

Small angle X-ray diffraction

Experimental setup:

- $q = |\boldsymbol{k}_1 - \boldsymbol{k}_0| = 4\pi \sin(2\theta/2) / \lambda$
- Measure $I = f(q)$
- Small deviation: $2\theta < 5°$
- *Sample detector distance: few meters*
- *Detector orthogonal to incident beam (tilt neglected)*



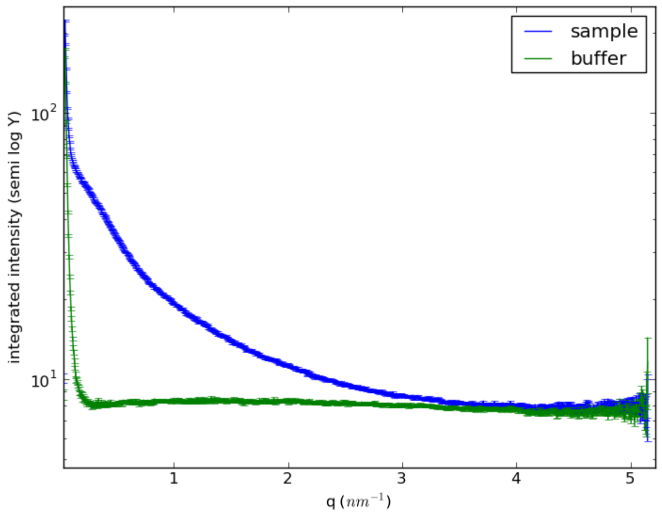


Data from BM29

Azimuthal integration

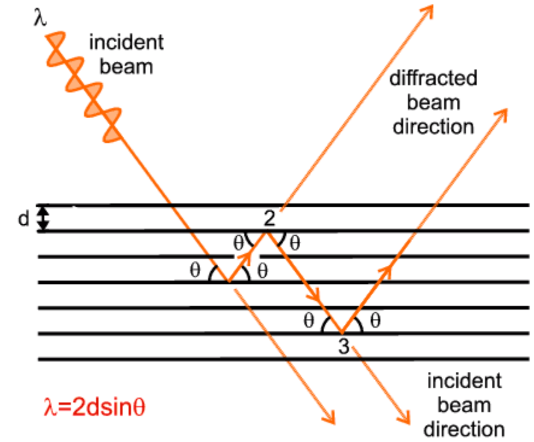
PyFAI

<u>Or other</u>
FIT2D
SPD
XRDUA
Foxtrot …

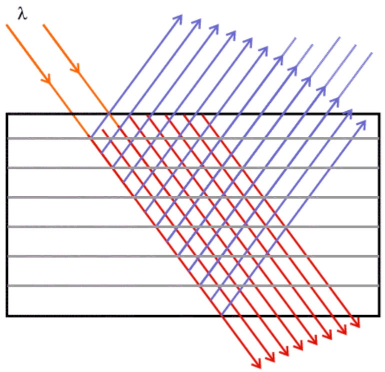


X-ray diffraction and powder diffraction

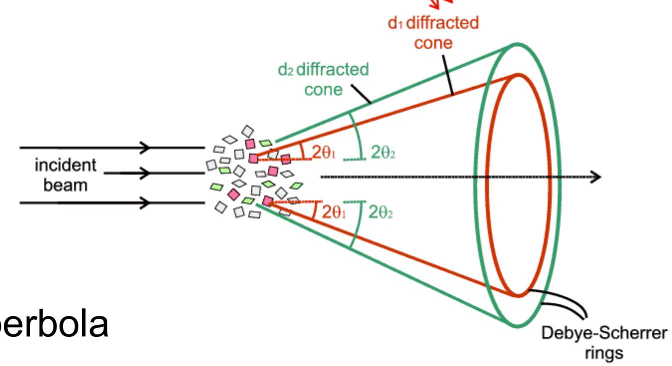
- **Atom plans in a crystal acts like a grating:**
  - Constructive interference for:
    - *n λ = 2d sin(θ)*
    - Known as **Bragg's law** (1913)



- **The angle between incident and diffracted beam is discrete, depending on the d-spacing inside the crystal**
  - Angle of diffraction: $2\theta = 2\ sin^{-1}(\ n\lambda\ /\ 2d\ )$



- **Powder are composed from randomly oriented cristallites: only those in Bragg condition diffracts**
  - Diffraction along cones centered on the sample
  - Intersection with the detector plan: conic
    - Called **Debye-Scherrer rings**
    - PyFAI is compatible with ellipses, parabola or hyperbola



Use of 2D detectors

Pros:

- Collect more photons thanks to larger solid angle
- Great statistic

Cons:

- CCD & CMOS detector need dark-current subtraction
- Most detector need flat-field correction to equalize the pixel response
- Optical fiber taper used together with CCD detectors induce distortion
- Each pixel has a different solid angle when viewed from sample
- Polarization of the X-ray beam induces variation of signal
- Intensity can need to be linearized (power $p \approx 1.0$)
- Pixel are always too large and limit your resolution.

PyFAI made the choice of:

- Pixel-wise correction $I_{corr} = \dfrac{I^p - I^p_{dark}}{flat * \delta\Omega * Polarization * Absorption}$
- Geometry transformation directly to destination space (r, χ) or (q, χ) or $(2\theta, \chi)$
  - **This includes the taper distortion: no taper corrected image is produced**

Part II

PyFAI for azimuthal integration

General description of pyFAI

- **Primarily a Python library to perform azimuthal integration**
  - The main object is an AzimuthalIntegrator instance which has:
    - **The geometry of the experimental setup**
    - **High performance integration/averaging routines**
    - **Pixel-wise correction of the image for:**

      dark current, flat field, solid angle, polarization, mask, absorption…
- **Tools to fit the geometry using a reference sample and diffracted rings**
- **Set of detectors (almost 40 different detectors)**
- **Set of reference sample (about 10 different calibrants)**
- **Tools to perform distortion correction for a fibre optic taper**
- **Tools to write easily plugins for other applications**
  - Lima, EDNA, PyMca, …

Working with pyFAI from the sources

- **PyFAI is an open source project ...**
  - Hosted on http://github.com/kif/pyFAI
  - Bug and issue tracker on github:
  - Mailing list: pyFAI@esrf.fr
    - **Open outside ESRF and archived**

- **Dependencies:**
  - Requested
    - **Python 2.6 or 2.7**
    - **Numpy, scipy and matplotlib**
    - **FabIO to be able to read images**
  - Optional
    - **Cython and a C compiler to compile binary modules**
    - **PyOpenCL and a supported GPU for best performances**
    - **Sphinx to build the documentation**

Tutorial 1: Test pyFAI

- **Download PyFAI and test it (under a UNIX computer):**

At ESRF:
$ export https_proxy=http://proxy.esrf.fr:3128/
$ export http_proxy=http://proxy.esrf.fr:3128/

$ git clone https://github.com/kif/pyFAI

$ cd pyFAI

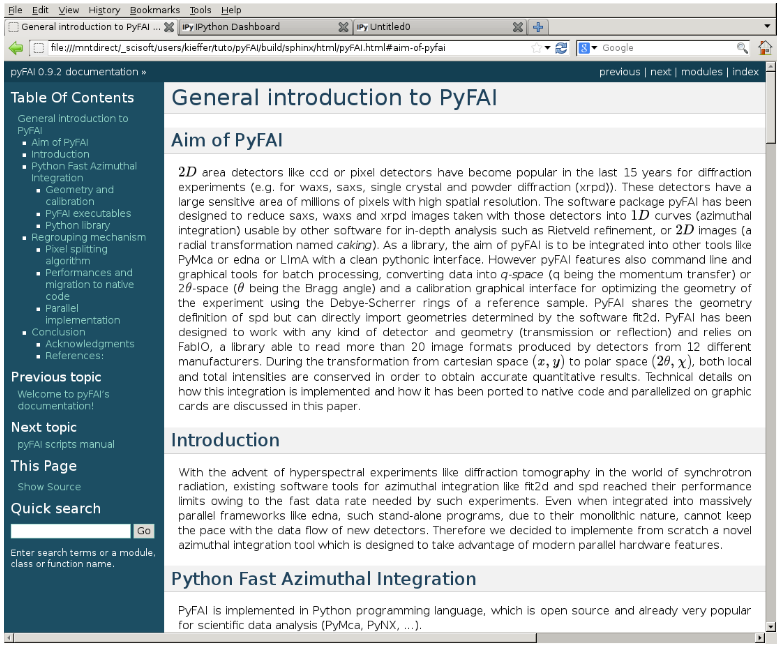
$ python setup.py build

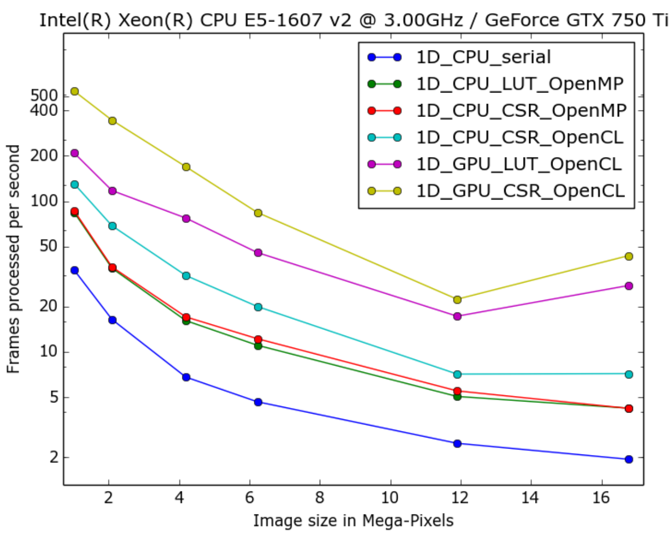
$ python setup.py test ← takes a variable amount of time due to downloads

$ python setup.py build_doc

$ firefox ./build/sphinx/html/index.html

$ cd benchmark; ./benchmark.py -c -g





← takes a lot of time but nice graphics

Python library for Fast Azimuthal Integration

- **Why Python:**
  - Widely used general-purpose, high-level programming language
  - Design philosophy emphasizes code readability: fewer lines of code
  - Clear programs on both a small and large scale.
- **PyFAI code analysis (line of code):**
  - **Python 12k lines of code**
  - **Cython 8k lines of code → generated 400k lines of C**
  - **OpenCL 2k lines of kernel code → to run on GPU for example**
  - **Tests: 3k lines of python**
- **Most of the tutorial will use Python**
  - For scripting: ipython notebook with the pylab interface:

    $ ipython notebook --pylab

A bit theory about azimuthal integration

- **Re-grid data on a regular, polar grid**
  - Input data: x, y
  - Output data: (r, χ) or (q, χ) or $(2\theta, \chi)$

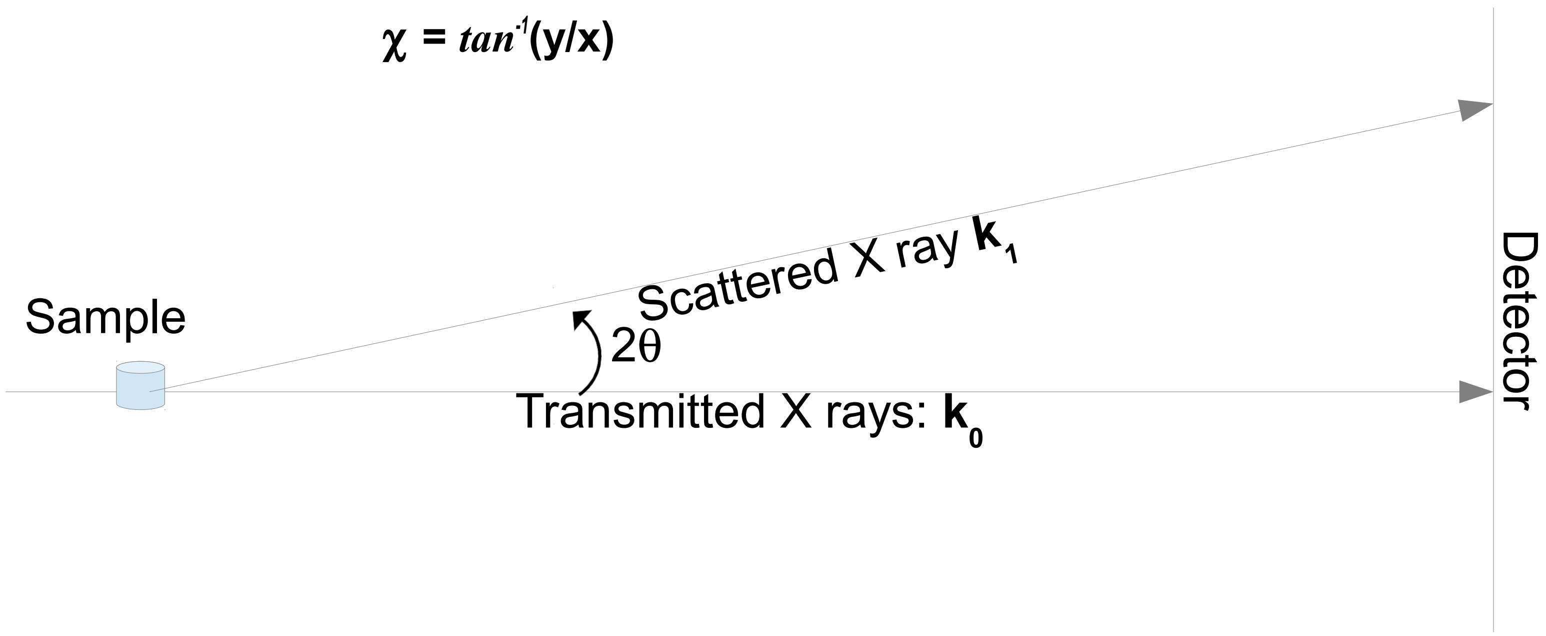
$$r = \sqrt{x^2 + y^2}$$

$$2\theta = \tan^{-1}(r/d)$$

$$q = 4\pi \sin(2\theta/2) / \lambda$$

Not linear functions:
each needs a specific re-griding
or interpolation

$$\chi = \tan^{-1}(y/x)$$



Tutorial: Generate a test image with rings

- **Create a test image with concentric rings**
  - **Work in radius, math are easier but the method is the same as q**

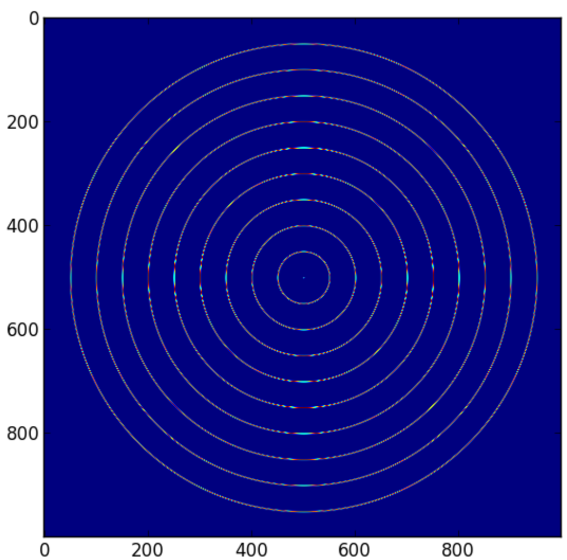
    $r = sqrt(x^2 + y^2)$
  - **Image shape: 1000x1000**
  - **center: 500,500,**
  - **rings spaces by 50 pixels**
  - **Gaussian shape for the rings:** $I = exp(-(r-r_0)^2)$

Correction: Generate a test image with rings

IP[y]: Notebook   Untitled0   Last saved: May 07 4:32 PM

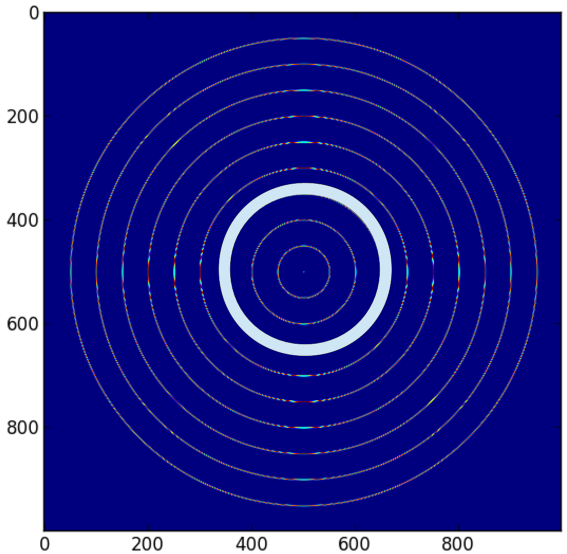
File Edit View Insert Cell Kernel Help

```
In [4]: shape = 1000,1000        # shape of image
        xc,yc = 500,500          # position of center
        rings = range(0,500,50) #position of rings in pixel

In [5]: y,x = numpy.ogrid[:shape[0],:shape[1]]
        x -= xc
        y -= yc
        r = numpy.sqrt(x*x+y*y)
        img = numpy.zeros(shape)
        for radius in rings:
            img += numpy.exp(-(r-radius)**2) #gaussian profile

In [6]: imshow(img)

Out[6]: <matplotlib.image.AxesImage at 0x4cb6d50>
```



Tutorial: Azimuthal integration, first implementation

- **Average the intensity of all pixels with radius in [r,r+dr]**
  - Get pixel coordinates of all pixel between r and r+dr (numpy.where)
  - Take intensities and average them (mean method of numpy array)
  - Loop over r
- **Make a function taking an image and the radius array as input**
  - Should output the I = f(radius)
  - Benchmark it (use %timeit)



Correction: Azimuthal integration, first implementation

- **One possible solution:**

In [41]:
```
def integrate0(img, radius, bins):
    rmin = radius.min()
    rmax = radius.max()
    delta_r = (rmax-rmin)/(2.0*bins)
    integrated = numpy.zeros(bins)
    positions = linspace(rmin+delta_r, rmax-delta_r, bins)
    for i,r0 in enumerate(positions):
        valid = abs(radius-r0) < delta_r
        pix_pos = numpy.where(valid)
        intensities = img[pix_pos]
        integrated[i] = intensities.mean()
    return positions, integrated
```

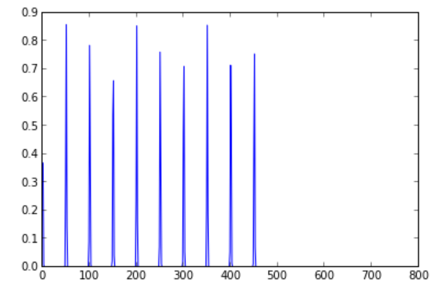
If you decompose mean = sum / number
- number can be obtained from histogram
- sum can be obtained from weighted histogram
→ numpy has a faster implementation for histograms

In [43]:
```
%timeit R,I = integrate0(img,r, 500)
```

```
1 loops, best of 3: 6.11 s per loop
```

In [42]:
```
clf()
R,I = integrate0(img,r, 500)
plot(R,I)
```

Out[42]: `[<matplotlib.lines.Line2D at 0x840d590>]`



- **Drawback: Terrible speed !**

Tutorial: histogram based integration

- **Look at the documentation of numpy.histogram**
- **Perform the histogram of the radius array**
  - What is the meaning of it ?
- **Perform the histogram of the radius array weighted by image intensity**
  - What is the meaning of it ?
  - The mean being the sum divided by the number of element …
    can you perform the integration ?
- **Re-implement the former function using histograms**
- **What are the performances ?**

Correction: histogram based integration

```
In [82]: def integrate1(img, radius, bins):
             u_hist, edges = numpy.histogram(radius, bins)
             w_hist, edges = numpy.histogram(radius, bins, weights=img)
             positions = (edges[:-1]+edges[1:])/2.0
             integrated = w_hist/(1.0*u_hist)
             integrated[u_hist==0] = 0
             return positions, integrated
```

- **Performances are much better**

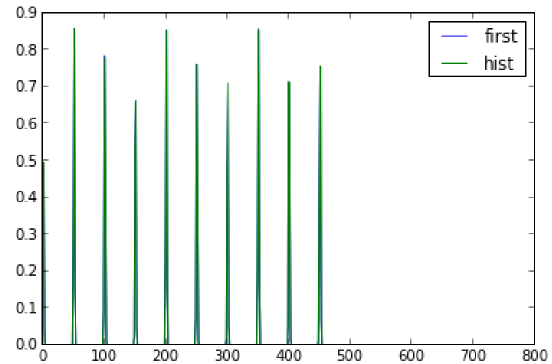
```
In [83]: %timeit R,I = integrate1(img,r, 500)

1 loops, best of 3: 118 ms per loop
```

```
In [84]: clf();
         R0,I0 = integrate0_corr(img,r, 500)
         plot(R0,I0,label="first")
         R1,I1 = integrate1(img,r, 500)
         plot(R1,I1,label="hist")
         legend()

Out[84]: <matplotlib.legend.Legend at 0x98e69d0>
```

- **Note the small differences:**

  Related to the number of

  pixels/bin



About the precision of this method …

- **Perform the same analysis when the original image has 20% noise**
- **Perform the analysis data with 100, 200, 500, 1000 and 2000 bins**
  - Overlay all results on a single graph
  - What do you see ?

Two major steps are performed during azimuthal integration:

- Pixel wise corrections
- Regridding on a polar grid

Two main methods:

- *`AzimuthalIntegrator.integrate1d`*
- *`AzimuthalIntegrator.integrate2d`*

Exercise: look at the function signature

Pre-processing: pixel-wise corrections

- **Masked pixels are not treated …**
  - Hence mask should be defined on the raw image and not on the distortion corrected image.
    - **Define the mask using the "drawMask_pymca" script provided**
    - **Perform the "un-correction" of a mask defined for FIT2d:**
      - Use the p*yFAI.distortion.Distortion.uncorrect(img)* method (slow)
- **Intensity correction:**
  - Dark current subtraction: $I - I_{dark}$
  - Flat field correction I / I_{flat}
  - Solid angle correction: $I/\cos^3(\alpha)$ where α is the incidence angle
  - Polarization correction: $I*2/((1+\cos(2\theta) - f \cos(2\chi)*(1 - \cos(2\theta))))$
  - Linearity correction and absorption correction shall be addressed upstream
- **All calculation are performed in floating point:**
  - single precision with error compensation on GPU (Kahan summation_
- **All correction can be activated or disable.**
  - Only solid angle correction is activated by default

Regridding is not interpolation

- **Interpolations are not well adapted to experimental needs:**
  - Most common are nearest neighbor, bi-linear, bi-cubic, …
  - No conservation of the intensity, nor locally nor globally
- **Integration provides intensity conservation.**
- **Various patterns of pixel splitting implemented in pyFAI:**
  - No pixel splitting (like EMBL Hamburg):
    - **Histogram type algorithms: noisy**
  - Fixed splitting on 2 bins (like Foxtrot, Soleil):
    - **Bins and image size are linked, all pixels have the same size**
  - Bounding box splitting (like FIT2D, ESRF)
    - **Pixels are assumed to be parallel to the output grid**
    - **Over estimation of the pixel size, tend to smear-out features**
  - Full pixel splitting
    - **Pixels are split according to their actual area, assuming they are quadrilateral**

Nota: PyFAI integration algorithms provide intensity conservation only without pixel-wise preprocessing, i.e. solid angle correction deactivated

Tutorial: influence of the pixel splitting scheme

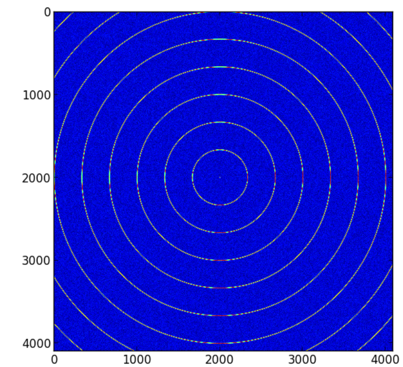
- **Use PyFAI.AzimuthalIntegrator to generate the geometry:**
  - Define a Fairchild detector (4096x4096 detector, 15µm square pixel size)
  - Place the beam-center at (2000, 2000), the detector at 100 mm from sample
  - Generate the radius array: r=f(x,y)
- **Create an image with rings on it**
  - Add noise to the signal
- **Perform integration using no splitting, bounding box and pixel splitting**
- **Look at the result where the statistics are low**
- **Look at the peak shape**

Correction: influence of the pixel splitting scheme

In [6]:
```
import pyFAI, numpy
ai = pyFAI.AzimuthalIntegrator(detector="fairchild") # Fairchild is a 4k x 4k detector with 15 micons pixel size
shape = ai.detector.max_shape                       # retrieve the size of the image
bins = 2000                                          # number of bins for integration
ai.setFit2D(100, 2000, 2000)                         # Place the detector at 100mm with the beam center at 2000, 2000
r = ai.rArray(shape)                                 # contains the radius array
```

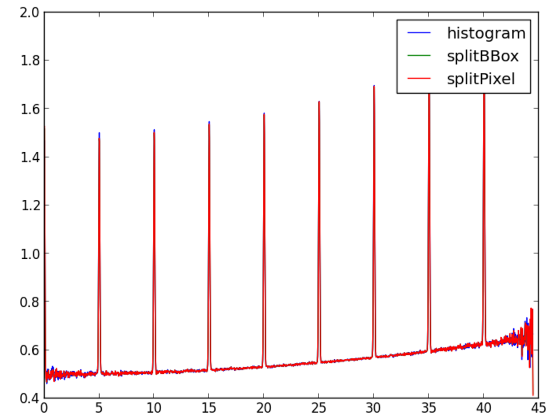
In [9]:
```
img = numpy.zeros(shape)
sigma = 50e-6    #that >3 pixels
for radius in numpy.arange(0, 0.045, 0.005):
    img += numpy.exp(-(r-radius)**2/(2*sigma**2))
img_noise = img + numpy.random.random(shape) #ratio S/N = 1
imshow(img_noise)
```

Out[9]: <matplotlib.image.AxesImage at 0x7f7edd8cc490>

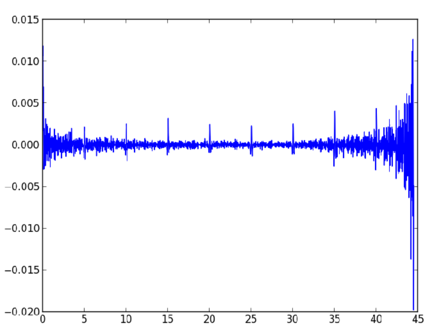


In [10]:
```
clf();
plot(*ai.integrate1d(img_noise, bins, unit="r_mm",method="numpy"),label="histogram")
plot(*ai.integrate1d(img_noise, bins, unit="r_mm",method="splitBBox"),label="splitBBox")
plot(*ai.integrate1d(img_noise, bins, unit="r_mm",method="splitPixel"),label="splitPixel")
legend()
```

Out[10]: <matplotlib.legend.Legend at 0x7f7eddb6a990>

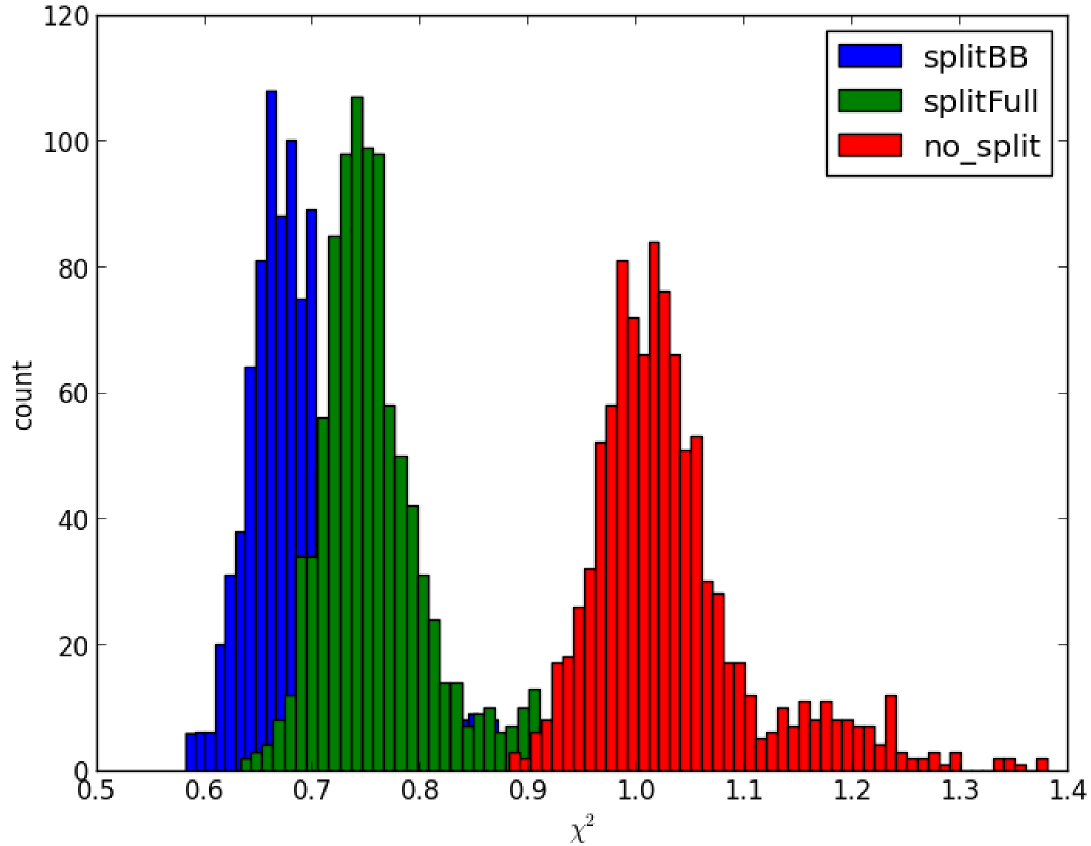


In [11]:
```
#difference betweem splitBBox and splitPixel:
R,I_bbox = ai.integrate1d(img_noise, bins, unit="r_mm", method="splitBBox")
R,I_pixel = ai.integrate1d(img_noise, bins, unit="r_mm", method="splitPixel")
plot(R,I_pixel-I_bbox)
```



Statistical consideration on pixel splitting ...

- **Influence of pixel splitting on errors:**
  - 1000 frames taken without sample (water only) and average to get
    - **χ^2 is too low when using pixel splitting (regardless the scheme)**
      → Correction specific for each geometry (under study)



Question: Do you know the Point Spread function of your detector ?

Direct integration(histogram) vs look-up table (SmDv)

- **Histogram or direct integration:**
  - Efficient on a single core (serial execution)
  - Hard to parallelize due to write conflicts
    - **Either have large locked region (atomic operation)**
    - **Either transform the problem is a parallel problem**
  - Well suited to changing geometries
- **“Scatter to Gather” transformation**
  - Store the contribution of each pixel to each bin in a matrix
  - Processing similar to a Sparse Matrix Dense Vector (SmDv) multiplication
- **Look-up table integration or backwards integration:**
  - Easy to parallelize, highly efficient on GPU
  - Need to calculate the LUT in a “serial” way
    - **Not suited if geometry changes often !**

Tutorial on azimuthal integration

- **Create a detector object for a Titan from Oxford diffraction**
  - See the factory or the class ...
- **Create an AzimuthalIntegrator object with the detector and the geometry**
  - Play with set getFit2D/setFit2D/getPyFAI/setPyFAI
- **Create a calibrant object for Lantanide hexaboride**
  - Use the fake_calibration_image to generate an image
- **Try out integrate1d and integrate2d**
  - Plenty of options, all described in the documentation
  - Benchmark it

- **Issue: how do I get the geometry in real life ???**
  - Use the calibration: next chapter

Programs to perform integration: pyFAI-integrate

- **pyFAI-integrate: the graphical interface for integration**
- **pyFAI-waxs: command line interface for integration**
- **pyFAI-saxs: command line interface for integration**
- **diff_tomo: diffraction mapping & tomography tool**

Graphical user interface to integrate a set of images

1) Click here for file dialog to load PONI file

Parameters populated from poni-file

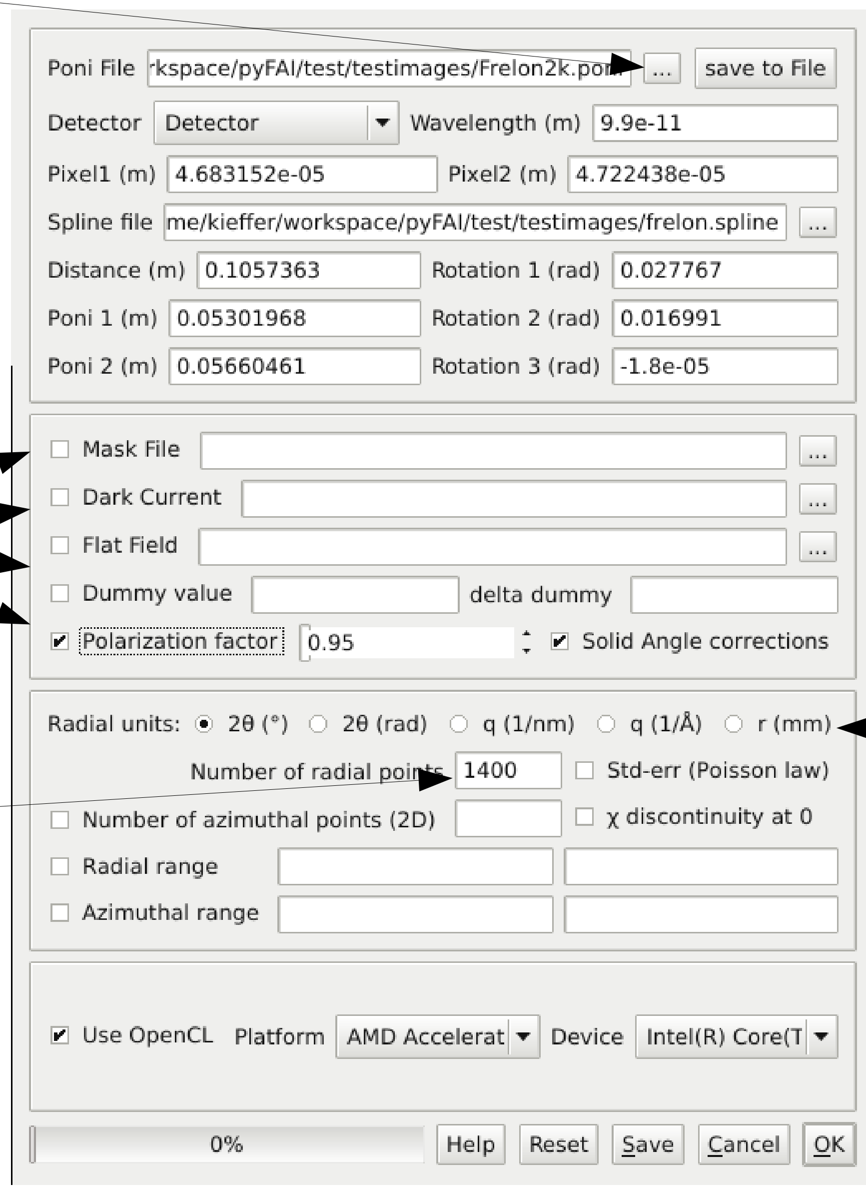
Check to activate correction

File dialogue To select images

Chose your output space

Mandatory

GPU selection



pyFAI-waxs --help

- **Usage: pyFAI-waxs [options] -p ponifile file1.edf file2.edf …**
- Options:

```
--version            show program's version number and exit
-h, --help           show this help message and exit
-p PONIFILE          PyFAI parameter file (.poni)
-n NPT               Number of points in radial dimension

-w WAVELENGTH, --wavelength=WAVELENGTH
                wavelength of the X-Ray beam in Angstrom

-e ENERGY, --energy=ENERGY
                energy of the X-Ray beam in keV (hc=12.398419292keV.A)

-u DUMMY, --dummy=DUMMY
                dummy value for dead pixels

-U DELTA_DUMMY, --delta_dummy=DELTA_DUMMY
                delta dummy value

-m MASK, --mask=MASK  name of the file containing the mask image
-d DARK, --dark=DARK  name of the file containing the dark current
-f FLAT, --flat=FLAT  name of the file containing the flat field
-P POLARIZATION_FACTOR, --polarization=POLARIZATION_FACTOR
                Polarization factor, from -1 (vertical) to +1
                (horizontal),                 default is None
                for no correction, synchrotrons are around 0.95

--error-model=ERROR_MODEL
                Error model to use. Currently on 'poisson' is
                implemented

--unit=UNIT          unit for the radial dimension: can be q_nm^-1, q_A^-1,
                2th_deg, 2th_rad or r_mm

--ext=EXT            extension of the regrouped filename (.xy)
```

pyFAI-saxs --help

- **usage: pyFAI-saxs [options] -n 1000 -p ponifile file1.edf file2.edf …**

  Azimuthal integration for SAXS users.

- positional arguments:

```
  FILE                  List of files to calibrate
```

- optional arguments:

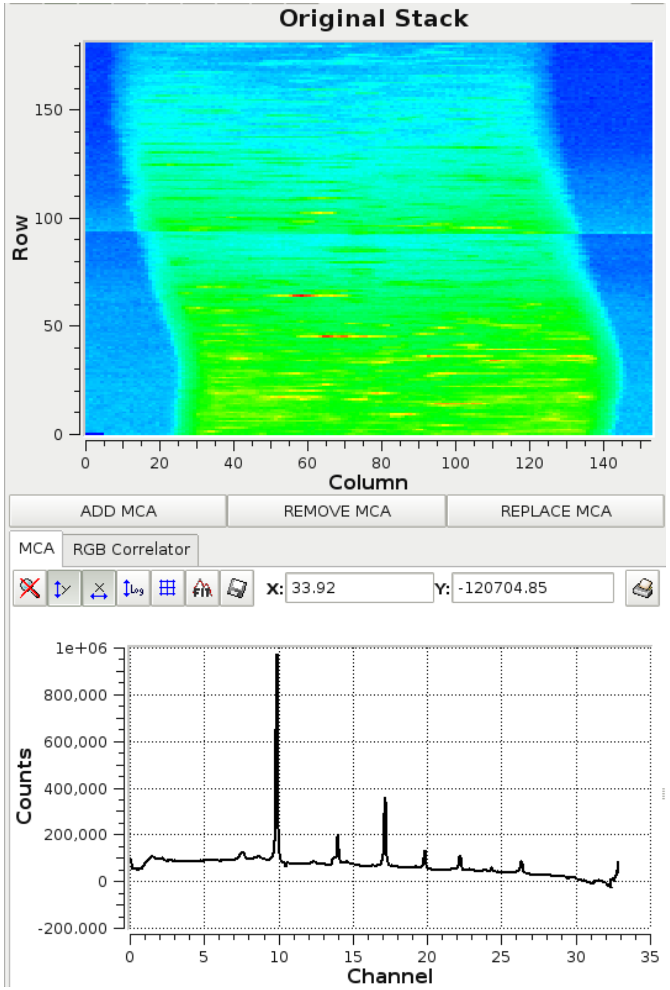
```
  -h, --help            show this help message and exit
  -v, --version
  -p PONIFILE           PyFAI parameter file (.poni)
  -n NPT                Number of points in radial dimension
  -w WAVELENGTH, --wavelength WAVELENGTH
                        wavelength of the X-Ray beam in Angstrom
  -e ENERGY, --energy ENERGY
                        energy of the X-Ray beam in keV (hc=12.398419292keV.A)
  -u DUMMY, --dummy DUMMY
                        dummy value for dead pixels
  -U DELTA_DUMMY, --delta_dummy DELTA_DUMMY
                        delta dummy value
  -m MASK, --mask MASK  name of the file containing the mask image
  -d DARK, --dark DARK  name of the file containing the dark current
  -f FLAT, --flat FLAT  name of the file containing the flat field
  -P POLARIZATION_FACTOR, --polarization POLARIZATION_FACTOR
                        Polarization factor, from -1 (vertical) to +1
                        (horizontal), default is None for no correction,
                        synchrotrons are around 0.95
  --error-model ERROR_MODEL
                        Error model to use. Currently on 'poisson' is
                        implemented
  --unit UNIT           unit for the radial dimension: can be q_nm^-1, q_A^-1,
                        2th_deg, 2th_rad or r_mm
  --ext EXT             extension of the regrouped filename (.dat)
```

pyFAI-saxs is the SAXS script of pyFAI that allows data reduction (azimuthal integration) for Small Angle Scattering with output axis in q space.

Tool: diff_tomo --help

Description of the options:

```
diff_tomo      #Name of the program
-o dt.h5       # output file
-t 153         # number of translation
-r 181         # number of rotations
-d dark.edf    # image with dark-current
-p Si.poni     # File with the calibration
-g             # Perform the calculation on the GPU
data/*.edf     # all 2D diffraction images
```



→ see tutorial

Performances measured on the processing: 100 ms/frame
40Mpix/s → overhead of the IO !!!
Reading/writing 217GB over an hour → 75MB/s

Dataset from ID22 NiNa

Geometry determination

Come back to real world:
Given a reference sample 2D diffraction pattern,
determine the geometry of the experiment

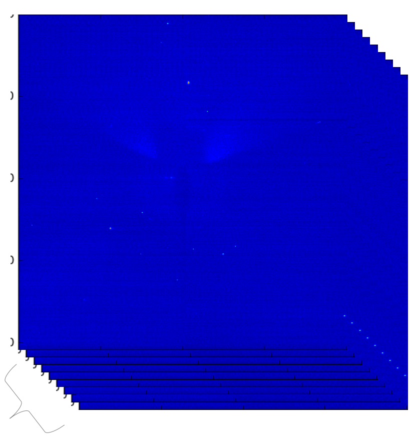
Calibration

- **The determination of the geometry is also known as calibration**
- **PyFAI assumes this setup does not change during the experiment**
- **It is divided into 4 major steps:**
  - Pre-processing of images: averaging, dark/flat correction, desaturation
  - Identification of peaks and groups of peaks belonging to same ring
  - Least-squares refinement of the geometry parameters on peak position
  - Validation by an human being of the geometry

Pre-processing tool: PyFAI-average

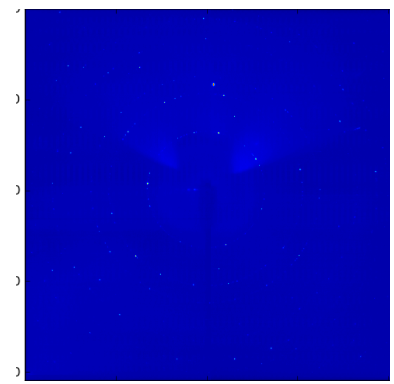
- **Average out many files using various functions:**
  - Mean: by default
  - Median: remove zinger
  - Max/min → see example
  - Any other function available from ndarray stacks
- **Average all but outliers**
  - cutoff based on standard deviation

Do this for your:
- Data image
- Dark-current image
- Flat field image

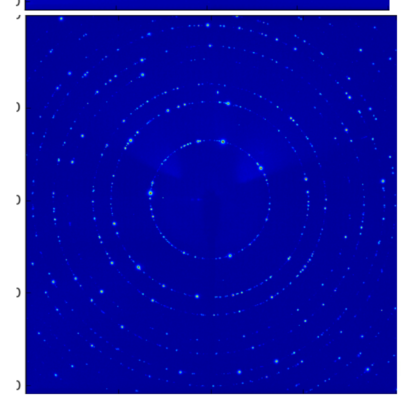


100 frames (Si from ID22 NiNa)

Mean filter



Max filter



Note on flat field images

- **Flat-field images integrate the signal response of each pixel**

  → It is not the image of the beam without sample !!!

  - They are acquired using an isotropic source (fluorescence)
  - Multiple detector/source configuration should be averaged
  - The flat image is rather stable in time but varies with energy !
  - The flat field image should ideally have a average signal of 1

- **PyFAI can estimate the contribution from the taper:**

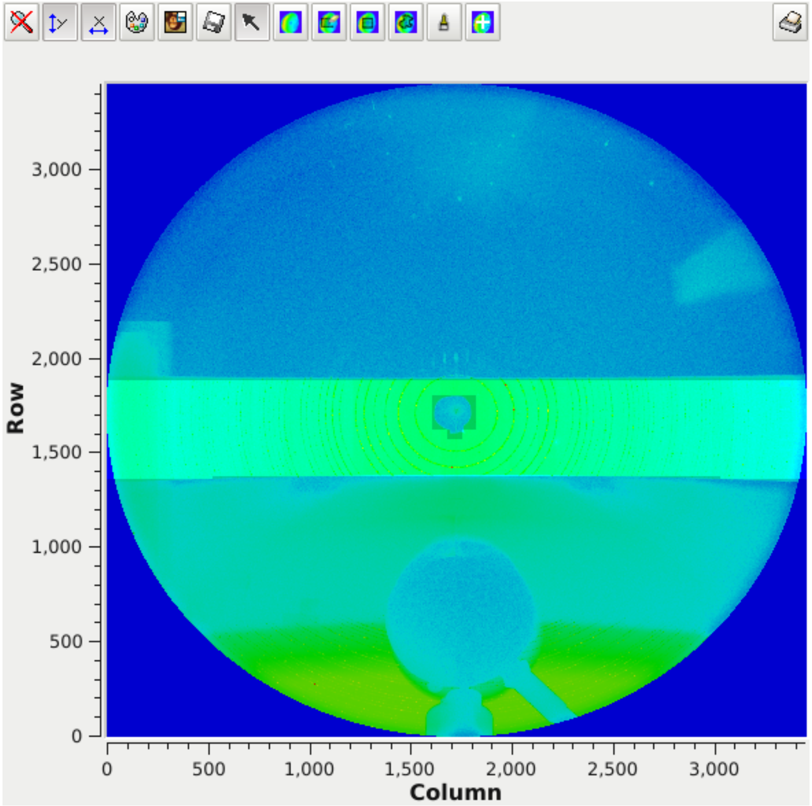
  - Taken into account in the solid-angle correction
  - Needs to be deactivated if you provide a flat-field image

    ```
    AzimuthalIntegrator.set_correct_solid_angle_for_spline(False)
    ```

Pre-processing tool: masking tool

- **Most detectors have some invalid areas:**
  - Inter-module space (Pilatus detectors)
  - CCD pixels not connected to the taper

    → **pyFAI provides a set of 39 detecors (68 labels) with their masks**
- **You will need to mask the beamstop + other shadows**
- **drawMask_pymca**
  - Provided by PyFAI
  - Requires PyMca4 installed



Note about masks … when using a distorted detector

- **PyFAI can use FIT2D masks but …**
  - FIT2D masks are drawn on distorted corrected images
  - PyFAI applies the mask on the raw image
- **If you have a distorted detector and a mask drawn with FIT2D:**
  - Either draw a new mask, using `drawMask_pymca`
  - Or distort your mask to apply it on the raw image:

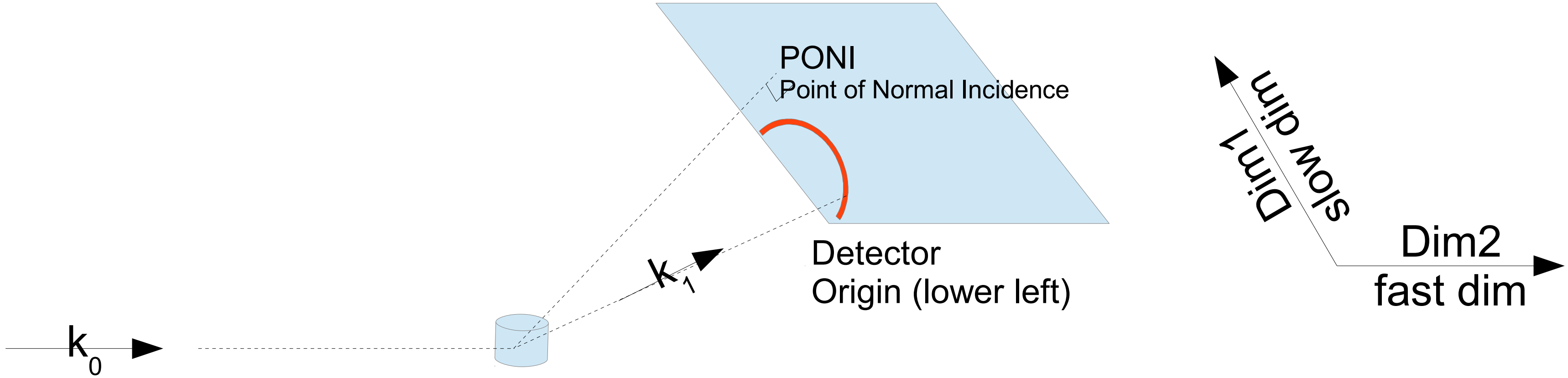
```
import pyFAI, fabio, pyFAI.distortion
fit2d_msk = fabio.open("fit2d.msk").data
detector = pyFAI.detectors.FReLoN("eo2k.spline")
distortion = pyFAI.distortion.Distortion(detector)
pyfai_mask = distortion.uncorrect(fit2d_msk)[0]>0
edf = fabio.edfimage.edfimage(data=pyfai_mask.astype('int8'))
edf.save('pyfai_mask.edf')
```

- **Retrieve the default mask for a detector:**

```
import pyFAI, fabio
det = pyFAI.detectors.detector_factory("mar345").astype('int8')
fabio.edfimage.edfimage(data=det.mask).write('mar345_mask.edf')
```

Geometry optimization: least squares refinement

- **Geometry:**
  - 3 distances, in meter: dist, poni1, poni2
  - 3 angles in radians: rot1, rot2, rot3 (later not changed usually)
  - Optionally the wavelength can be optimized (correlated to dist at low 2θ)



- **Optimization:**
  - Optimization of square of the distance in 2θ to the calibrant in radians
    - **Can weight optimization using intensities**
  - Minimize the cost function using Sequential Least SQuares Programming

    → **scipy.optimize.fmin_slsqp**

Peak picking: extract peak position from image

- **Steepest ascent**
  - Starting from a seed, search for the nearest (local) maximum
  - Sub-pixel precision obtained from second order expansion: $\delta pos = -H^{-1} \times \nabla$
    - **Methods to calculate the hessian and gradient, noise issue**
- **Massif algorithm**
  - Groups of peaks are extracted from a difference with a blurred image
  - Unique parameter: width of the gaussian for blurring
  - Uses binning and un-binning to keep processing time reasonable
- **Blob-detection: systematic peak search**
  - Difference of subsequent blurs. Search for all
  - Octave changing with binning

→ see Aurore's presentaton

- **Monte-Carlo sampling**
  - Peak selection when region is known
  - Based on random search + steepest ascent

Calibrants: provide opening of cones

- **PyFAI ships 10 reference sample (decreasing 2θ of first ring):**
  - Au: Gold
  - CeO2: Ceria
  - Si: Silicon
  - alpha_Al2O3: Corundum
  - Cr2O3: Chromium oxide
  - CrOx: Undefined chromium oxide formerly used on MX beamlines
  - LaB6: Lantanide hexaboride
  - PBBA: Para Bromo Benzoic Acid
  - C14H30O: tetradecanol
  - AgBh: Silver Behenate
- **But you can provide your d-spacing file if you prefer:**
  - Ascii text files with d-spacing written in Angstrom (like FIT2D)
  - Use the American Minaralogist database:
    - **http://rruff.geo.arizona.edu/AMS/amcsd.php**

Strength of the geometry

- **Can accommodate highly tilted detector**
  - See tutorial on LaB6
- **Can import/export geometry of FIT2D**
  - getFit2D/setFit2D method from geometry/azimuthalIntegrator
- **Stable to binning:**
  - LaB6 at 0.1m from detector (60µm, 2048*2048), orthogonal geometry
  - Calibration performed after binning: precision ~1/10 of a pixel

Binning	Pixel Size (µm)	Poni1 (m)	Poni2 (m)	Delta1 (µm)	Delta2 (µm)
1	60	6.00E-002	6.00E-002	4.820	1.630
2	120	6.00E-002	6.00E-002	-8.530	-4.020
4	240	6.00E-002	6.00E-002	-2.700	-4.210
8	480	6.00E-002	6.01E-002	45.410	91.440

Tutorial on calibration

- **Few examples from tests:**
  - HalfCCD FReLoN image of LaB6 from ID11 taken at λ=0.27Å
  - FReLoN image of Si from ID22 taken at λ=0.42Å

- **Extra tutorial: See in folder with examples**
  - Calibration of images taken with the detector on a 2θ-arm
    - **LaB6 images taken with the Titan detector at λ=1Å**
  - Saxs data with a pixel detector (dealing with gaps in detector)
    - **AgBh images taken with a Pilatus1M at λ=1Å**

Additional tools:

Extra scripts which can make your life easier

List of scripts in pyFAI:

- **Pre-processing tools:**
  - drawMask_pymca: tool for drawing a mask
  - pyFAI-average: tool for averaging/median/... filtering images
- **Calibration tools:**
  - pyFAI-calib: Initial calibration tool
  - pyFAI-recalib: Calibration refinement tool (Obsolete)
  - MX-calibrate: Calibrate automatically a set of images
  - check_calib: checks the calibration of an image at the sub-pixel level
- **Azimuthal integration tools:**
  - pyFAI-integrate: graphical interface for integration
  - pyFAI-waxs: command line interface for integration
  - pyFAI-saxs: command line interface for integration
  - diff_tomo: diffraction mapping&tomography tool

Calibration tool: MX-calibrate

- **Tool to calibrate a set of diffraction images taken at various distances (especially MX beamlines)**
  - Expect distance to be in the filename for linear regression
  - Can read the wavelength from file header (or from command line)

- **Example of use:**

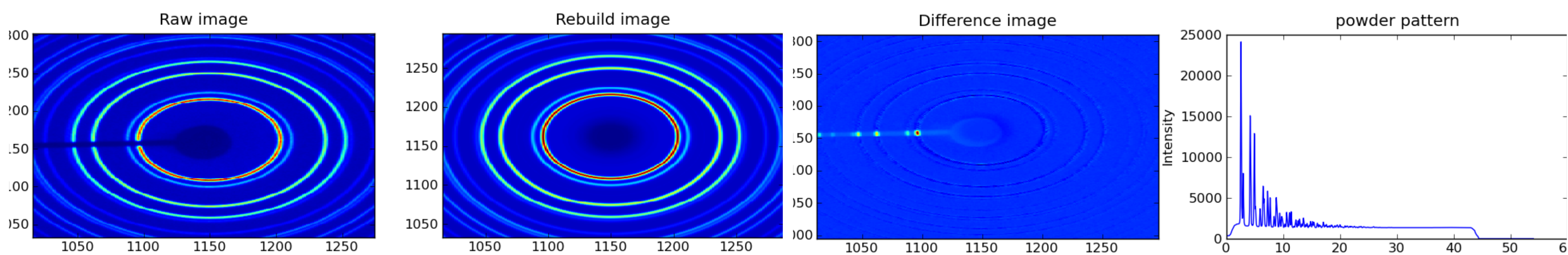
  MX-calibrate -D Pilatus6M -c CeO2 ceria_*.cbf

- **Returns: Linear regression of parameters vs detector distance**

```
dist = 0.00100143591695 * dist_mm + -0.00112366879462    R= 0.999999965812   stderr= 1.17109288577e-07
poni1 = -3.41861282886e-06 * dist_mm + 0.215648741605    R= -0.996810176908  stderr= 1.2240640473e-07
poni2 = 1.64908246321e-06 * dist_mm + 0.212896001186     R= 0.935403434948   stderr= 2.78771824164e-07
rot1 = 1.17701119402e-06 * dist_mm + 0.00689408012249    R= 0.676514144642   stderr= 5.72993860345e-07
rot2 = -1.56845501269e-06 * dist_mm + 0.00502610867189   R= -0.895473163114  stderr= 3.48664084642e-07
rot3 = 3.11858790559e-12 * dist_mm + -1.13933614082e-09  R= 0.60959568452    stderr= 1.81362213889e-12
direct = 1.00147388349 * dist_mm + -1.12406896453        R= 0.99999996682    stderr= 0.000115373737414
tilt = 9.14175326104e-06 * dist_mm + 0.487856144005  R= 0.118853078555   stderr= 3.41542499598e-05
trp = 0.0128891131751 * dist_mm + 143.975004565          R= 0.936685043656   stderr= 0.00215489991677
centerX = -0.0350625303913 * dist_mm + 1238.48253482 R= -0.99989808711   stderr= 0.000223882943102
centerY = 0.00427419793753 * dist_mm + 1254.36314238 R= 0.975374520063   stderr= 0.000432230388786
```

Calibration tool: check_calib

- **Tool to validate the quality of a calibration:**
  - Perform the azimuthal integration
  - Project back in the averaged pattern on the original image
  - Shows the difference map, highlighting:
    - **Distortion spline file errors**
    - **Defects in the dark measurment**
    - **Inhomogeneities in the detector response (flat)**
  - Calculate the offset between the two images using phase correlation
    - **Precision better then 0.1pixel (here $\delta x = +0.07$ & $\delta y = -0.001$)**



→ proper mask is mandatory

Detector calibration and distortion correction

- **PyFAI can deal with distortion correction for detectors:**
  - Spline-file generated from FIT2D
  - Displacement maps for Pilatus detectors (in percent of pixel)
- **They are obtained from a regular grid**
  - There is no tool (yet) to generate displacement maps in pyFAI
  - Can be added if needed → 2 weeks of work
- **PyFAI can re-sample data on a regular grid if needed:**
  - Similar to azimuthal integration (full pixel splitting)

```
import pyFAI, fabio, pyFAI.distortion
det = pyFAI.detectors.FReLoN('splinefile.spline')
dis = pyFAI.distortion.Distortion(det)
cor_img = dis.correct(raw_img)
```

PyFAI as a plugin ...

- **EDNA:**
  - Generic plugin to perform azimuthal integration
  - Optimized plugin for BM29 BioSaxs (10 Mpixel/s in production)
- **LImA and ProcessLib :**
  - Azimuthal integration performed online (→ BM01)
  - Distortion correction
  - Demo available …
- **PyMca**
  - Under construction
- **Plugins based on a worker (pyFAI.worker.Worker instance):**
  - Azimuthal integrator + information about input and output shapes
  - Contains dark/flat/...
  - Communicates using JSON strings
  - Should be initialized for best performances
  - Output using a writer: pyFAI.io.Writer instance offering EDF, HDF5, …

Part V

Appendix

About good software engineering practices …

- **Coding standards**
  - Comments: every class/method has a documentation
  - Keep the code and the API simple for developers & users
- **Code development**
  - Most functions are internally tested to prevent regression
    - **However we are lacking functional tests on scripts**
  - Continuous integration:
    - **run test daily to prevent regression**

      → **The *master* branch is always useable**

- **Question about versionning:**
  - Would you like version number like 2014.06 ?
  - Version 1.0 is likely to never arrive

Local installation and bootstrapping

- **Test are always run from a local install of the library obtained from:**

  $ python setup.py build

  - **They are located in build/lib*/pyFAI**

- **One can use this build to test programs thanks to F.Picca:**

  $ ./bootstrap ipython

  In [1]: import pyFAI

  In [2]: pyFAI

  Out[2]: <module 'pyFAI' from '/users/kieffer/workspace-ssd/pyFAI/build/lib.linux-x86_64-2.6/pyFAI/__init__.py'>

About the project pyFAI → June 2014

- **1200 commits since it started in July 2011**
  - 11 contributors from 5 synchrotrons world-wide + academic labs
    - **ESRF, Soleil, Petra3, APS, Sesame + CEA, …**
- **Languages used:**
  - Python 23k lines
    - **including 8k of Cython → generates 400.000 lines of C**
    - **Including 3k for the tests**
  - OpenCL: 2k lines for parallel kernels
- **Mantra: K.I.S.S**
  - Simple is easier to understand
  - Simple is easier to maintains
  - Try to be self contained: easier to deploy !

Thanks to all contributors:

- **ESRF**
  - Jonathan Wright
  - Manuel Sanchez del Rio
  - V. Armando Solé
  - Aurore Deschildre
  - Amund Hov
  - Peter Boesecke
- **LinkSCEEM:**
  - Dimitris Karkoulis
  - Zubair Nawaz
  - Giannis Ashiotis
- **Synchrotron Soleil:**
  - Frédéric Emmanuel Picca
- **Advanced Photon Source:**
  - Clemens Prescher
- **Petra3:**
  - Gunthard Benecke
  - Gero Flucke

ESRF | The European Synchrotron