Tutorial MID data

January 26, 2024

1 Offline analysis tutorial

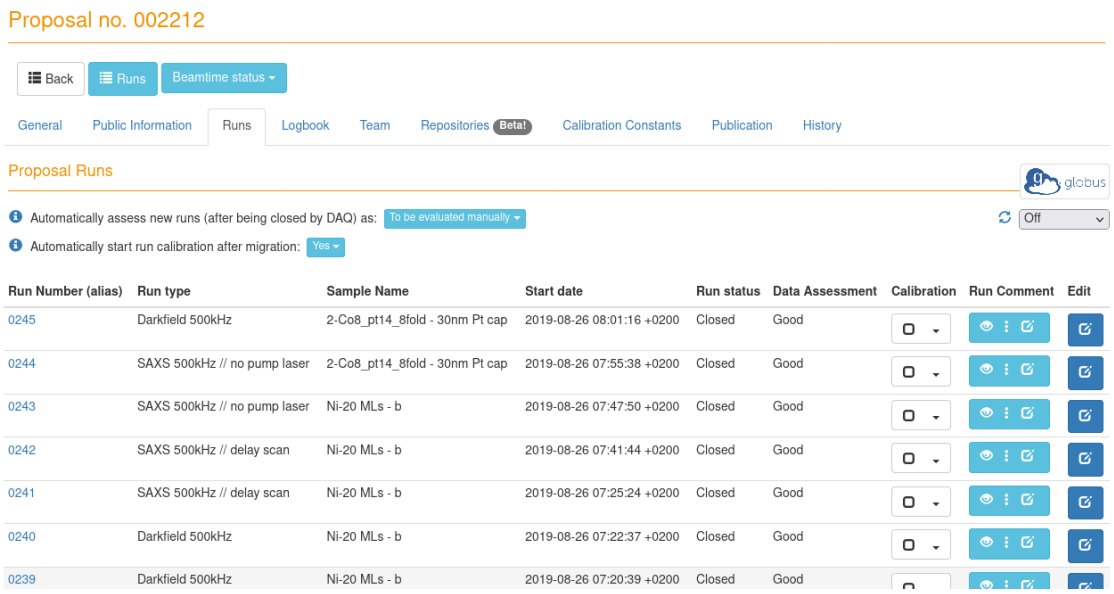
This tutorial is about generic offline analysis. For some types of experiment, such as crystallography, you probably want to feed the data to specific processing software - there's a separate session about that.

If you have an EuXFEL campus account, you can try running this notebook yourself at this link:

https://max-jhub.desy.de/user-redirect/notebooks/GPFS/exfel/exp/XMPL/201750/p700000/usr/Shared/um202
offline-tutorial/Tutorial%20MID%20data.ipynb#

1.1 Exploring data

Look at your proposal & runs in myMdC (metadata catalogue).

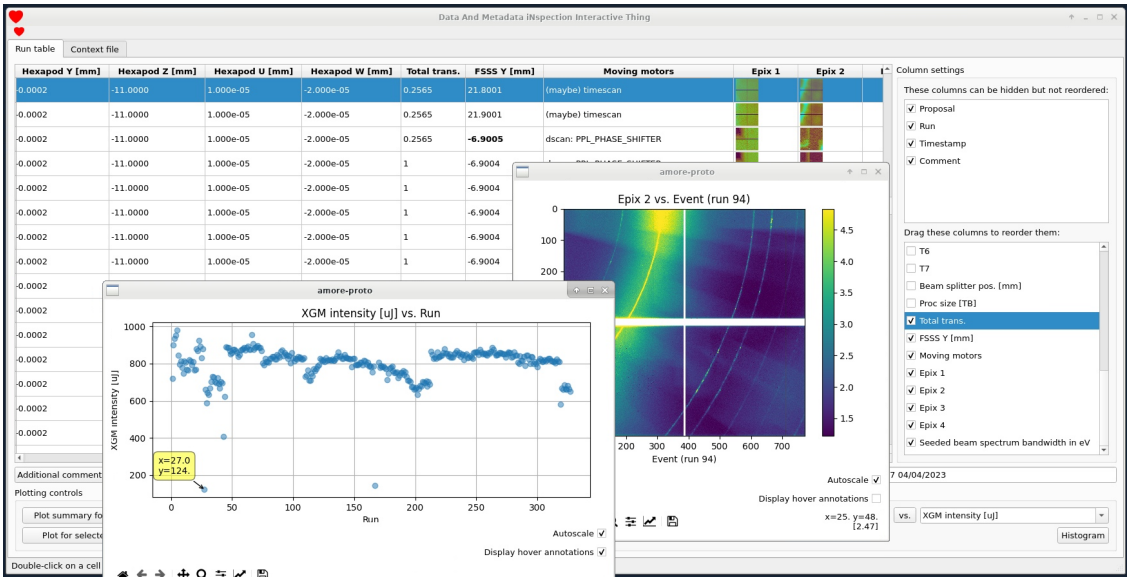


Command line tools:

```
module load exfel cmdxfel exfel-python

# find a run (or proposal) directory
findxfel 2958 54
```

```
# list sources in a run directory
lsxfel /gpfs/exfel/exp/XMPL/201750/p700000/proc/r0027

# see data size for each run in proposal
duxfel 2958
```

You can also work with DAMNIT to make an overview of your experiment.



There's a tutorial on this immediately afterwards in the other parallel session.

2 Data analysis

The examples I'll go through demonstrate:

- Correlating data from different sources
  - The EXtra-data library for accessing EuXFEL data
  - Loading data into memory as a NumPy array
  - Getting aligned data from different sources
  - Plotting
  - Loading large data in chunks
  - Using multiprocessing to load & process chunks in parallel
  - Using EXtra to work with scan data (**NEW**)
- Detector images & azimuthal integration
  - Detector geometry with EXtra-geom & assembling images
  - Using pyFAI for azimuthal integration
  - Using pasha to process data in parallel

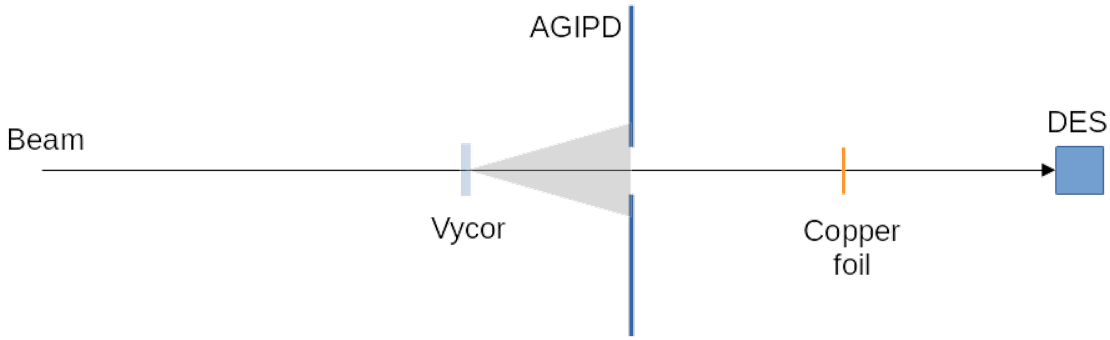
Thanks to Johannes Möller at MID for the example notebooks which formed the basis for this tutorial. Both of these examples use the AGIPD detector:

The AGIPD detector at MID - Image © European XFEL

[1]:
```
%matplotlib inline

import multiprocessing
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

from extra_data import open_run
from extra_data.components import AGIPD1M

mpl.rcParams['font.size'] = 14
mpl.rcParams['figure.figsize'] = (8, 6)
```

2.1 Correlating data

We'll look at a run from MID containing a scan of photon energy, made by adjusting the undulators in SASE 2. A copper foil is placed in the beam, and the diagnostic endstation (DES) is used to measure the X-rays passing through the foil. The ratio of intensity before the foil (measured by AGIPD) and after represents how much the copper foil absorbs X-rays at different photon energies. To compute this, we need to match up data from three different sources.



First, we'll open a run in Python using EXtra-data. Proposal 700000 contains sample data, which should be accessible to anyone on Maxwell. The run we're using here originally comes from MID.

`data='all'` combines corrected detector data with raw data from other sources. **Since 2022, uncorrected data is no longer copied to the `proc` folder,** so you're likely to want this option most of the time.

[2]:
```
run = open_run(proposal=700000, run=33, data='all')
run.info()
```

```
# of trains:    3353
Duration:       1:51:46.3
First train ID: 1032976906
Last train ID:  1033043968

16 XTDF detector modules (MID_DET_AGIPD1M-1)
```

```
  e.g. module MID_DET_AGIPD1M-1 0 : 512 x 128 pixels
  MID_DET_AGIPD1M-1/DET/0CH0:xtdf
  64 frames per train, up to 214592 frames total

5 instrument sources (excluding XTDF detectors):
  - MID_EXP_DES/CAM/CAM1:daqOutput
  - MID_EXP_FASTADC/ADC/DESTEST:channel_2.output
  - MID_RR_SYS/MDL/KARABACON:output
  - MID_XTD6_IMGPI/CAM/BEAMVIEW:daqOutput
  - SA2_XTD1_XGM/XGM/DOOCS:output

40 control sources:
  - MID_AUXT2_ATT/MDL/ATT
  - MID_AUXT2_ATT/MOTOR/ROD1_IN_OUT
  - MID_AUXT2_ATT/MOTOR/ROD2_IN_OUT
  - MID_AUXT2_ATT/MOTOR/ROD3_IN_OUT
  - MID_AUXT2_ATT/MOTOR/ROD4_IN_OUT
  - MID_DET_AGIPD1M/CC/MON_0
  - MID_EXP_AGIPD1M/GAUGE/PG1
  - MID_EXP_AGIPD1M/MOTOR/Q1M1
  - MID_EXP_AGIPD1M/MOTOR/Q1M2
  - MID_EXP_AGIPD1M/MOTOR/Q2M1
  - MID_EXP_AGIPD1M/MOTOR/Q2M2
  - MID_EXP_AGIPD1M/MOTOR/Q3M1
  - MID_EXP_AGIPD1M/MOTOR/Q3M2
  - MID_EXP_AGIPD1M/MOTOR/Q4M1
  - MID_EXP_AGIPD1M/MOTOR/Q4M2
  - MID_EXP_AGIPD1M/PSC/HV
  - MID_EXP_AGIPD1M/TSENS/H1_T_EXTHOUS
  - MID_EXP_AGIPD1M/TSENS/H2_T_EXTHOUS
  - MID_EXP_AGIPD1M/TSENS/Q1_T_BLOCK
  - MID_EXP_AGIPD1M/TSENS/Q2_T_BLOCK
  - MID_EXP_AGIPD1M/TSENS/Q3_T_BLOCK
  - MID_EXP_AGIPD1M/TSENS/Q4_T_BLOCK
  - MID_EXP_AGIPD1M1/CTRL/MC1
  - MID_EXP_AGIPD1M1/CTRL/MC2
  - MID_EXP_AGIPD1M1/FPGA/MASTER_H1
  - MID_EXP_AGIPD1M1/FPGA/MASTER_H2
  - MID_EXP_AGIPD1M1/MDL/FPGA_COMP
  - MID_EXP_DES/CAM/CAM1
  - MID_EXP_DES/PROC/CAM1
  - MID_EXP_DES/TSYS/TRIGGER_CAM1
  - MID_EXP_FASTADC/ADC/DESTEST
  - MID_EXP_SYS/TSYS/UTC-2-S4
  - MID_RR_SYS/MDL/KARABACON
  - MID_RR_SYS/MDL/PULSE_PATTERN_DECODER
  - MID_XTD1_UND/DOOCS/ENERGY
  - MID_XTD6_IMGPI/CAM/BEAMVIEW
```

```
  - MID_XTD6_IMGPI/SPROC/BEAMVIEW
  - MID_XTD6_MONO/MDL/ENERGY_CHANGER
  - SA2_BR_SYS/MDL/MID_REPEATER
  - SA2_XTD1_XGM/XGM/DOOCS
```

[3]:
```
run['MID_EXP_FASTADC/ADC/DESTEST:channel_2.output'].keys()
```

[3]:
```
{'data.baseline',
 'data.peakMean',
 'data.peakStd',
 'data.peaks',
 'data.rawBaseline',
 'data.rawData',
 'data.rawPeaks',
 'data.samplesForBaseline',
 'data.samplesPerPeak',
 'data.trainId'}
```

[4]:
```
des_intens = run['MID_EXP_FASTADC/ADC/DESTEST:channel_2.output', 'data.peaks'].
 ↪ndarray()
des_intens.shape
```

[4]:
```
(3352, 2700)
```

[5]:
```
run['MID_DET_AGIPD1M-1/DET/0CH0:xtdf', 'image.data'].shape
```

[5]:
```
(214464, 512, 128)
```

[6]:
```
214464 / 64
```

[6]:
```
3351.0
```

We have different numbers of measurements, because some trains are missing in each source.

Let's look at only the trains where we have data from the DES, undulators, and all 16 AGIPD modules:

[7]:
```
sel = run.select([
    'MID_EXP_FASTADC/ADC/DESTEST:channel_2.output',  # DES
    'MID_XTD1_UND/DOOCS/ENERGY',                     # Undulators
    'MID_DET_AGIPD1M-1/DET/*CH0:xtdf',               # AGIPD
], require_all=True)
```

[8]:
```
des_intens = sel['MID_EXP_FASTADC/ADC/DESTEST:channel_2.output', 'data.peaks'].
 ↪ndarray()
des_intens.shape
```

[8]:
```
(3351, 2700)
```

[9]:
```
sel['MID_DET_AGIPD1M-1/DET/0CH0:xtdf', 'image.data'].shape
```

[9]:
```
(214464, 512, 128)
```

[10]:
```
214464 / 64
```

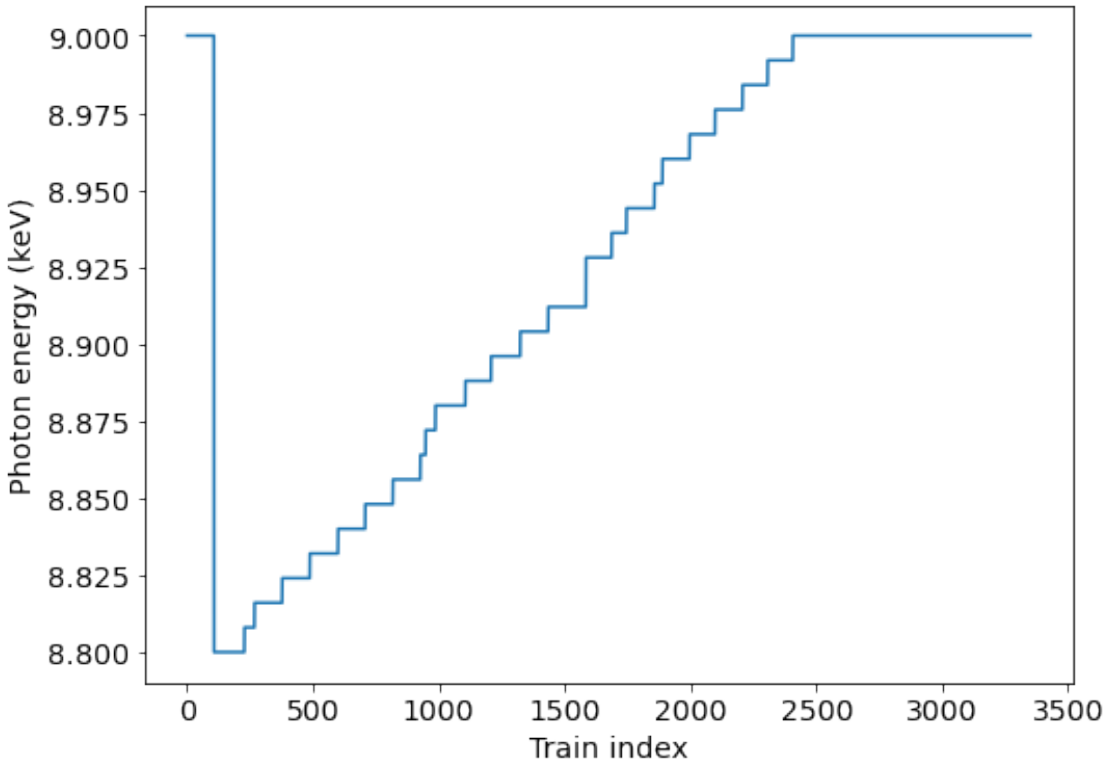
[10]:
```
3351.0
```

We have 2 fewer trains left in `sel` - those are the ones where data was missing for one of these sources.

Here is the scan of pulse energies in this run:

[11]:
```
photon_energy = sel['MID_XTD1_UND/DOOCS/ENERGY','actualPosition'].ndarray()

plt.plot(photon_energy)
plt.xlabel("Train index")
plt.ylabel("Photon energy (keV)")
```

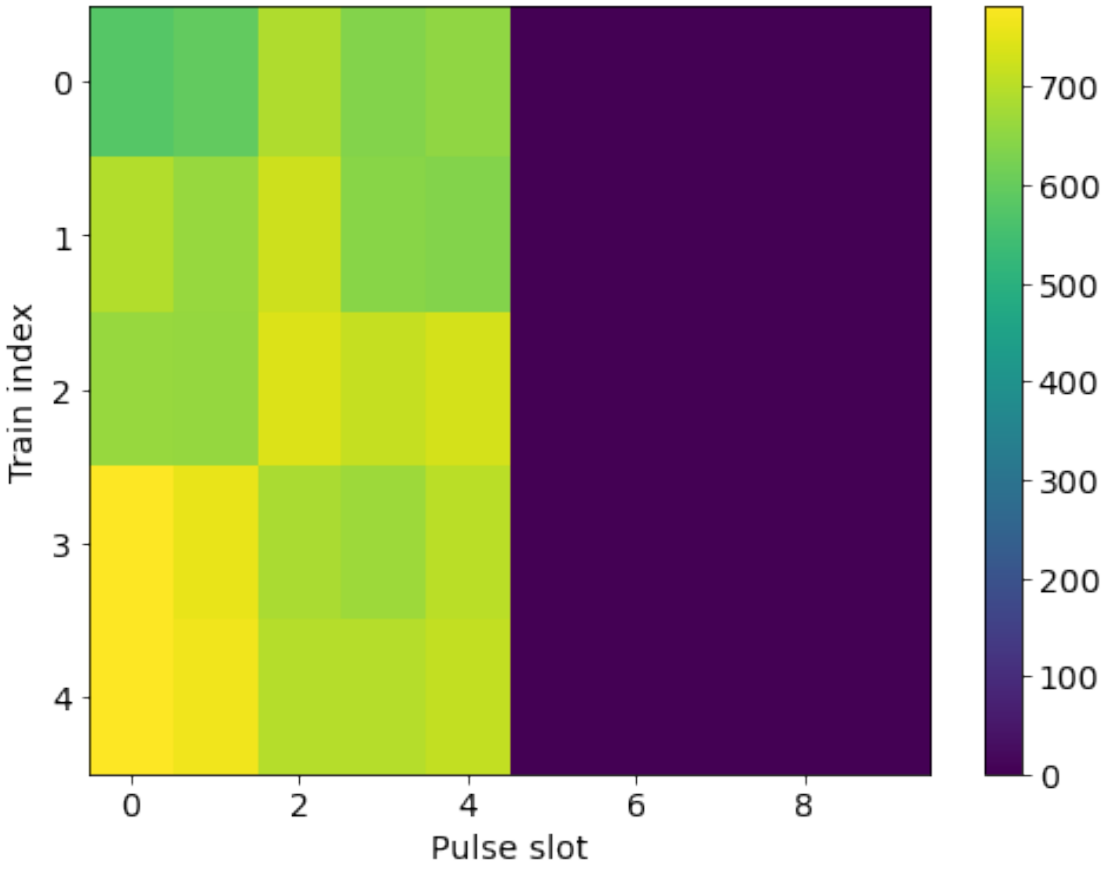
[11]:
```
Text(0, 0.5, 'Photon energy (keV)')
```



Now we can look at the intensity measured by the DES:

[12]:
```
plt.imshow(des_intens[:5, :10], aspect='auto')
plt.xlabel("Pulse slot")
plt.ylabel("Train index")
plt.colorbar()
```

[12]:
```
<matplotlib.colorbar.Colorbar at 0x2b1d8b22d750>
```



And the average value from all pixels of the AGIPD detector:

[13]:
```
agipd = AGIPD1M(sel)
agp_arr  = agipd.select_trains(np.s_[:5])['image.data'].ndarray()
agp_mask = agipd.select_trains(np.s_[:5])['image.mask'].ndarray()
agp_arr[(agp_mask > 0) & (agp_mask < 8)] = np.nan
agp_arr.shape
```

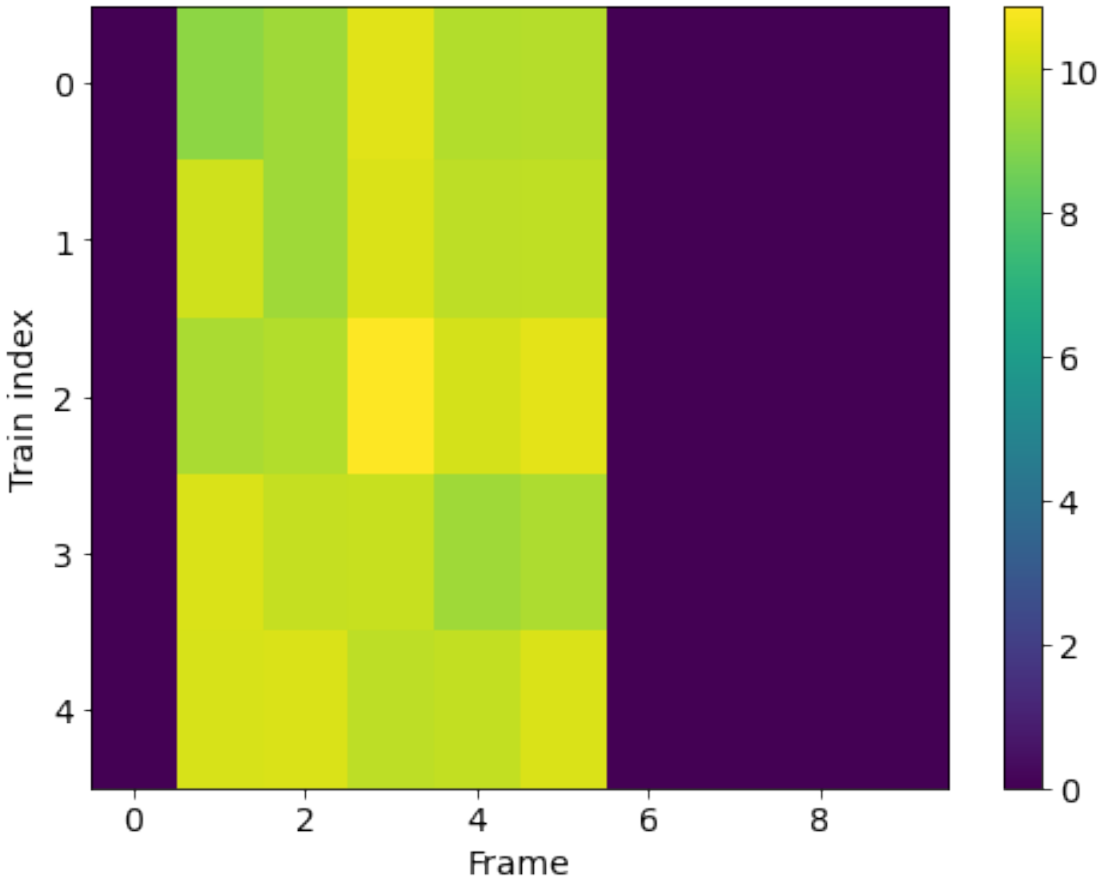
[13]:
```
(16, 320, 512, 128)
```

[14]:
```
# Average each frame, reshape to (trains, pulses)
agp_intens = np.nanmean(agp_arr, axis=(0, -2, -1)).reshape(5, -1)
agp_intens.shape
```

[14]: (5, 64)

[15]:
```
plt.imshow(agp_intens[:, :10], aspect='auto')
plt.xlabel("Frame")
plt.ylabel("Train index")
plt.colorbar()
```

[15]: <matplotlib.colorbar.Colorbar at 0x2b202a6b9330>

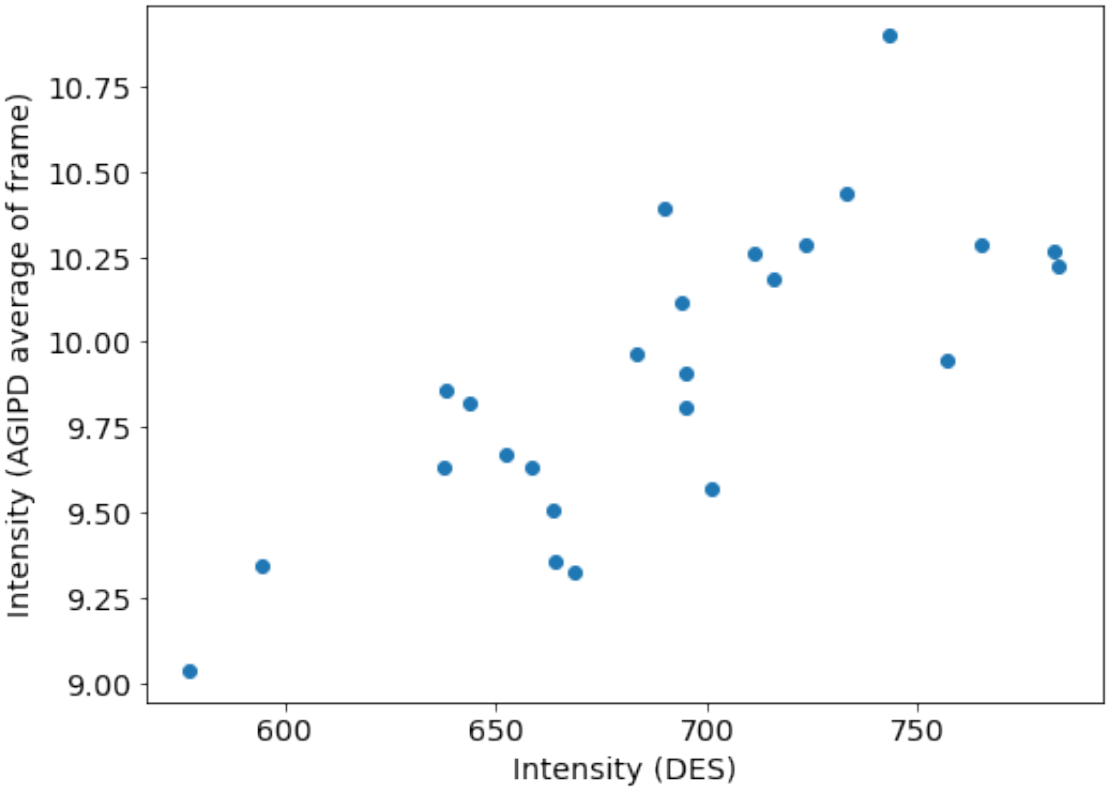


We've only got 5 pulses in each train, and in AGIPD they're starting from the second slot.

We can plot the DES and AGIPD data for the first 5 trains:

[16]:
```
plt.scatter(des_intens[:5, :5].ravel(), agp_intens[:, 1:6].ravel())
plt.xlabel('Intensity (DES)')
plt.ylabel('Intensity (AGIPD average of frame)')
```

[16]: Text(0, 0.5, 'Intensity (AGIPD average of frame)')



This looks like there is a correlation, though it's not yet very clear. Let's look at more data.

2.2 Working with big data

With only 5 frames per train, the AGIPD data in this run is about 70 GB, and the mask data is the same again. In this case, it's possible to load it all into memory at once. But similar experiments could easily have enough data that that's not possible.

Instead, we'll break the data up into chunks, and process them in parallel.

[17]:
```
# pixels  bits mods pulses
512 * 128 * 4 * 16 * 5 / 1e6  # MB / train
```

[17]: 20.97152

[18]:
```
15 * 20 * 64   # 15 trains per chunk, 64 workers
```

[18]: 19200

[19]:
```
[len(chunk.train_ids) for chunk in agipd.split_trains(trains_per_part=15)][:5]
```

[19]: [14, 15, 15, 15, 15]

Maximum memory use depends on `trains_per_part` and the number of worker processes you run (`Pool(64)` below).

[20]:
```
def agipd_frame_avg(agp_chunk):
    # Load only the 5 frames with X-rays in each train:
    agp_arr  = agp_chunk['image.data'].select_pulses(np.s_[1:6]).ndarray()
    agp_mask = agp_chunk['image.mask'].select_pulses(np.s_[1:6]).ndarray()

    # Apply the mask
    agp_arr[(agp_mask > 0) & (agp_mask < 8)] = np.nan

    # Average each frame to a single number
    return np.nanmean(agp_arr, axis=(0, -2, -1))
```

[21]:
```
%%time
with multiprocessing.Pool(64) as pool:
    results = pool.map(
        agipd_frame_avg,
        # Remove the select_trains part to process all the data (takes a few␣
 ↪minutes)
        agipd.select_trains(np.s_[:2000]).split_trains(trains_per_part=15)
    )
```

```
CPU times: user 1.03 s, sys: 967 ms, total: 2 s
Wall time: 1min 33s
```

`results` is a list of chunks, now we need to concatenate them together.

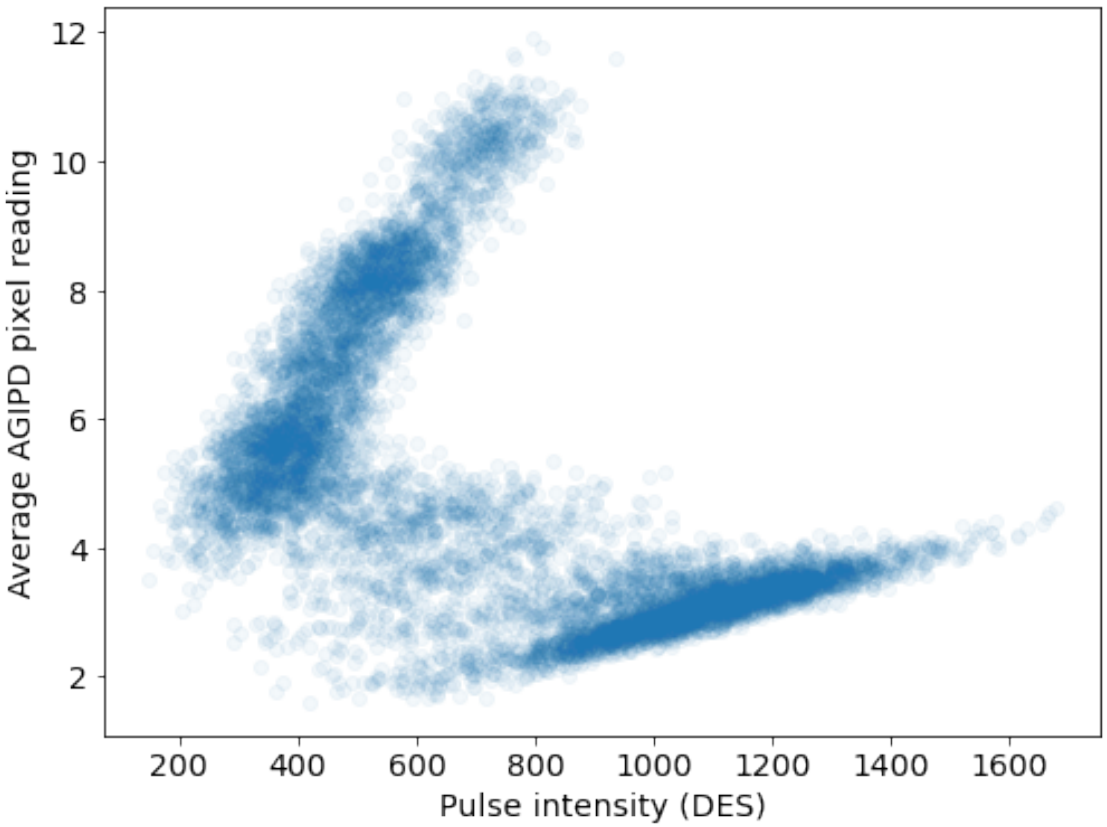
[22]:
```
agp_intens_all = np.concatenate(results)
agp_intens_all.shape
```

[22]:
```
(10000,)
```

Now plotting DES and AGIPD measurements shows a clear pattern, with two separate correlations.

[23]:
```
plt.scatter(des_intens[:2000, :5].ravel(), agp_intens_all, alpha=0.05)
plt.xlabel('Pulse intensity (DES)')
plt.ylabel('Average AGIPD pixel reading')
```

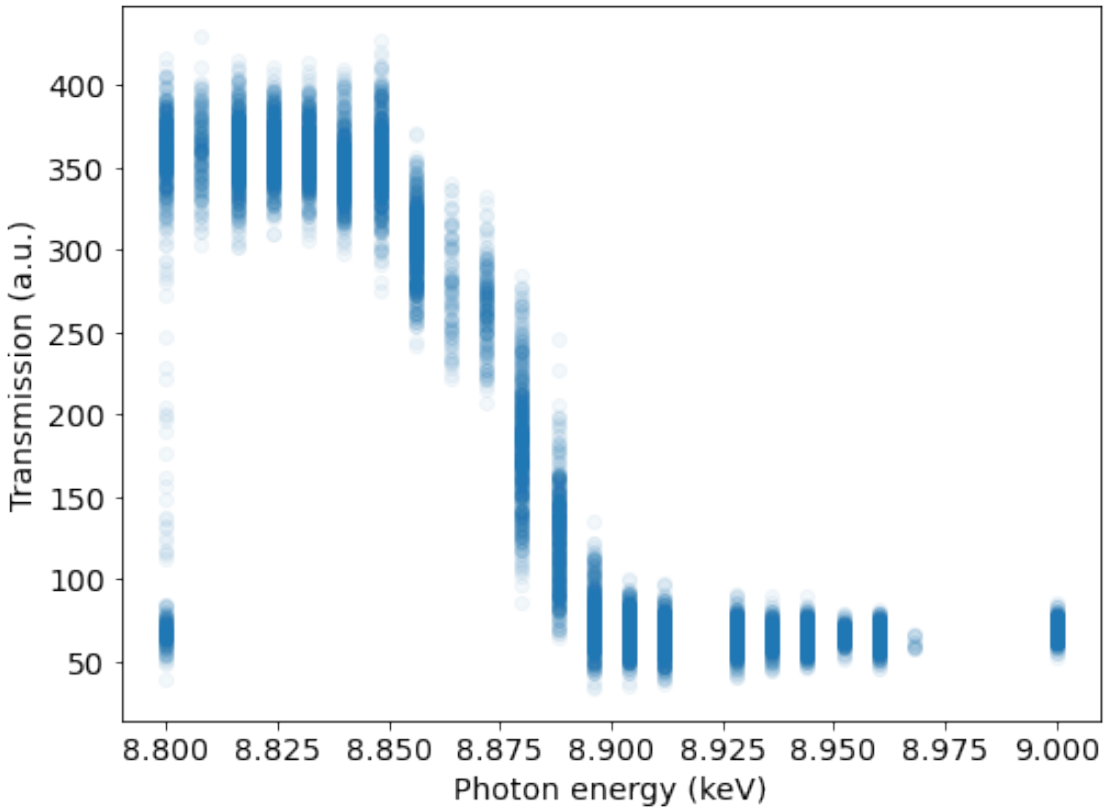
[23]:
```
Text(0, 0.5, 'Average AGIPD pixel reading')
```



The ratio of DES / AGIPD measurements represents X-ray transmission through the copper foil. We can calculate this and plot it against the photon energy.

[24]:
```
transmission = des_intens[:2000, :5].ravel() / agp_intens_all
plt.scatter(photon_energy[:2000].repeat(5), transmission, alpha=0.05)
plt.xlabel("Photon energy (keV)")
plt.ylabel("Transmission (a.u.)")
```

[24]:
```
Text(0, 0.5, 'Transmission (a.u.)')
```



We can see that the copper strongly absorbs X-rays above 8.9 keV.

2.3 EXtra components

NEW: We're building a group of classes to simplify access to common types of data. See https://extra.readthedocs.io/en/latest/components/ for more of these.

[25]:
```
from extra.components import Scantool, Scan
```

[26]:
```
scan_tool = Scantool(run)
scan_tool.info()
```

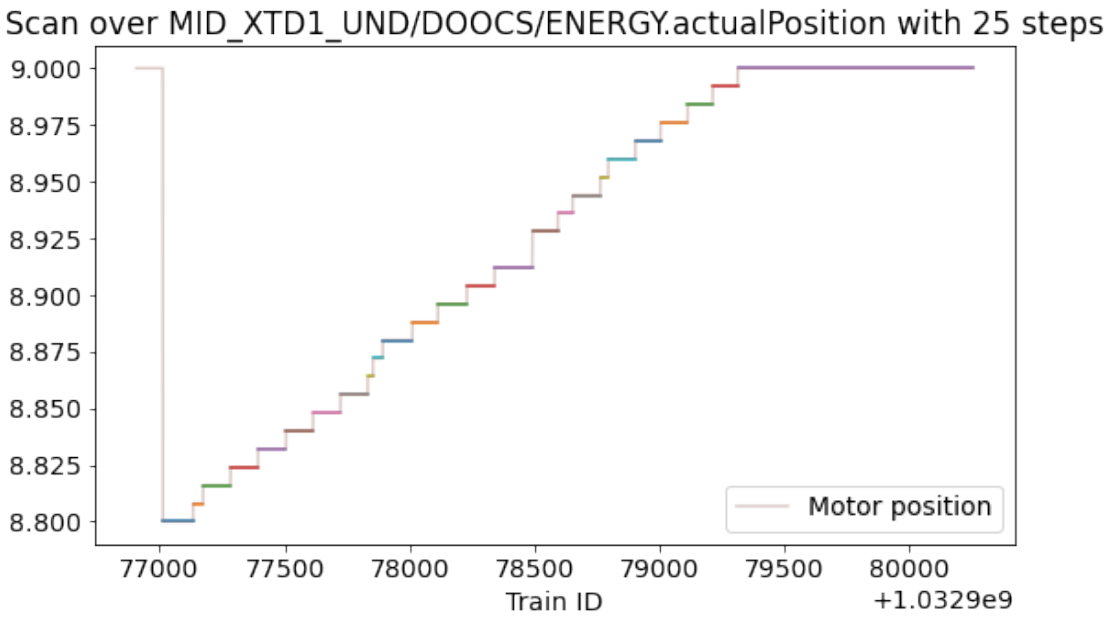
```
Scantool (MID_RR_SYS/MDL/KARABACON) configuration:
  Scan type: ascan
  Acquisition time: 5.0s

Motors:
  ENERGY_UND (MID_XTD1_UND/DOOCS/ENERGY): 8.8 -> 9.0, 25 steps
```

[27]:
```
scan = Scan(sel['MID_XTD1_UND/DOOCS/ENERGY'], min_trains=10)
scan.info()
```

```
Scan over MID_XTD1_UND/DOOCS/ENERGY.actualPosition from 8.8000 to 9.0000:
 Steps: 25
 Scan time: 0:05:24
 Average step length: 129.76 trains (12.98s)
 Average step size: 8.33e-03 [arb. u.]

Detection parameters:
 resolution: 4.00e-03
 min_trains: 10
 intra_step_filtering: 1
```

[28]:
```
scan.plot()
```

[28]:
```
<AxesSubplot:title={'center':'Scan over MID_XTD1_UND/DOOCS/ENERGY.actualPosition
with 25 steps'}, xlabel='Train ID'>
```



[29]:
```
scan.positions
```

[29]:
```
array([8.80000114, 8.80799961, 8.81599998, 8.8239975 , 8.83199978,
       8.84000015, 8.84800053, 8.85600185, 8.86400032, 8.87200165,
       8.87999916, 8.88799953, 8.895998  , 8.90399933, 8.9119997 ,
       8.9279995 , 8.93599987, 8.94399834, 8.95199966, 8.96000004,
       8.96800232, 8.97599888, 8.98400116, 8.99199867, 9.        ])
```

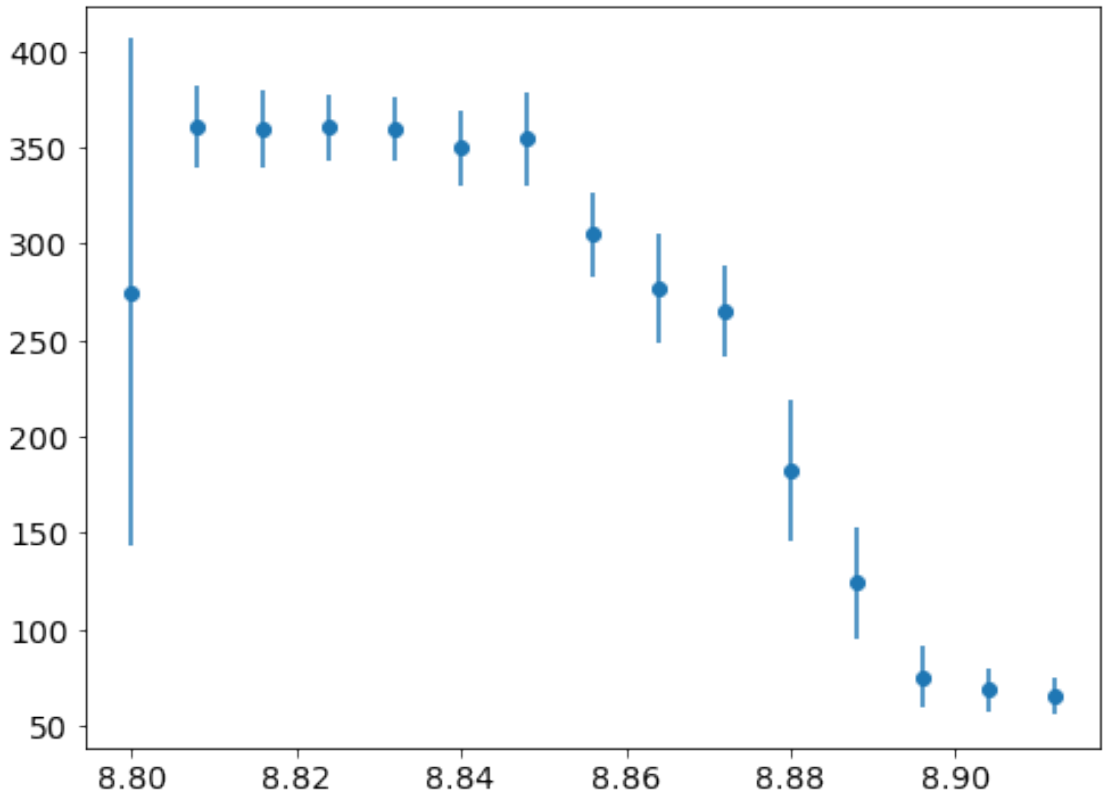
[30]:
```
scan.positions_train_ids[0]
```

[30]:
```
array([1032977013, 1032977014, 1032977015, 1032977016, 1032977017,
       1032977018, 1032977019, 1032977020, 1032977021, 1032977022,
       1032977023, 1032977024, 1032977025, 1032977026, 1032977027,
       1032977028, 1032977029, 1032977030, 1032977031, 1032977032,
       1032977033, 1032977034, 1032977035, 1032977036, 1032977037,
       1032977038, 1032977039, 1032977040, 1032977041, 1032977042,
       1032977043, 1032977044, 1032977045, 1032977046, 1032977047,
       1032977048, 1032977049, 1032977050, 1032977051, 1032977052,
       1032977053, 1032977054, 1032977055, 1032977056, 1032977057,
       1032977058, 1032977059, 1032977060, 1032977061, 1032977062,
       1032977063, 1032977064, 1032977065, 1032977066, 1032977067,
       1032977068, 1032977069, 1032977070, 1032977071, 1032977072,
       1032977073, 1032977074, 1032977075, 1032977076, 1032977077,
       1032977078, 1032977079, 1032977080, 1032977081, 1032977082,
       1032977083, 1032977084, 1032977085, 1032977086, 1032977087,
       1032977088, 1032977089, 1032977090, 1032977091, 1032977092,
       1032977093, 1032977094, 1032977095, 1032977096, 1032977097,
       1032977098, 1032977099, 1032977100, 1032977101, 1032977102,
       1032977103, 1032977104, 1032977105, 1032977106, 1032977107,
       1032977108, 1032977109, 1032977110, 1032977111, 1032977112,
       1032977113, 1032977114, 1032977115, 1032977116, 1032977117,
       1032977118, 1032977119, 1032977120, 1032977121, 1032977122,
       1032977123, 1032977124, 1032977125, 1032977126, 1032977127,
       1032977128, 1032977129, 1032977130, 1032977131, 1032977132,
       1032977133], dtype=uint64)
```

[31]:
```
from xarray import DataArray

# Label our computed transmission with train IDs
transmission_labelled = DataArray(
    transmission.reshape(-1, 5),
    dims=('trainId', 'pulseId'),
    coords={'trainId': sel.train_ids[:2000]}
)

# Bin transmission according to the scan steps
binned_transmission = []
binned_std = []
for trainids in scan.positions_train_ids[:15]:
    bin_vals = transmission_labelled.sel(trainId=trainids)
    binned_transmission.append(bin_vals.mean().item())
    binned_std.append(bin_vals.std().item())
```

[32]:
```
plt.errorbar(scan.positions[:15], binned_transmission, yerr=binned_std, fmt='o')
```

[32]: <ErrorbarContainer object of 3 artists>



2.4 Azimuthal integration

See the `geom-az-int.py` script in this folder for this example.