Tutorial on

py3DXRD - data analysis workflow

for the high-energy grain-resolved 3D XRD method

at P21.2 beamline, PETRA III

Repositories:

https://github.com/agshabalin/py3DXRD

/asap3/petra3/gpfs/common/p21.2/scripts/py3DXRD/   (for Maxwell users at DESY)

Dependencies:

py3DXRD is written in Python 3 using tools from

ImageD11 (https://github.com/FABLE-3DXRD/ImageD11)

PolyXSim (https://github.com/FABLE-3DXRD/PolyXSim)

and

HEXRD (https://github.com/HEXRD/hexrd)

and some other packages (like *polarTransform*) that can be installed using pip.

The easiest way to install all of them at once by duplicating my conda environment:

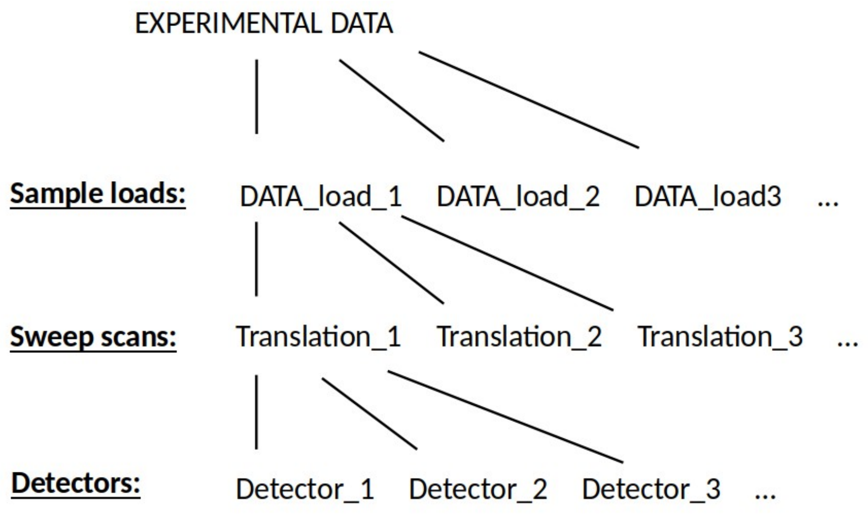
```
conda env create -f environment.yml
```

where /environment.yml is the settings file, you can find it in the "./example/" directory.

After creating this environment and activating it, you can try to run the example.py file in which you should modify the paths and other parameters to match your dataset.

<u>The py3DXRD structure and logic:</u>

Data structure:



Data analysis schematic:

Process images => Find peaks => Index => Filter gvectors => Find grains => Simulate diffraction

Principles:

1) Compatibility with both FABLE and HEXRD
2) Reusing debugged tools from FABLE and HEXRD
3) Object-oriented – analysis if divided by independent block
4) Flexibility – each block should be easy to modify for new requests
5) Logging every operation
6) Space efficient
7) Transparency in the code and output files, not a black box.

Programming realization:

The central idea behind py3DXRD is to use well debugged functionalities of ImageD11 and HEXRD, aim for compatibility with them, but retain flexibility, because the datasets and the ways to analyze them can be vastly different. For that, py3DXRD is programmed in object-oriented approach, where the objects are separate analysis blocks. They include the following stages:

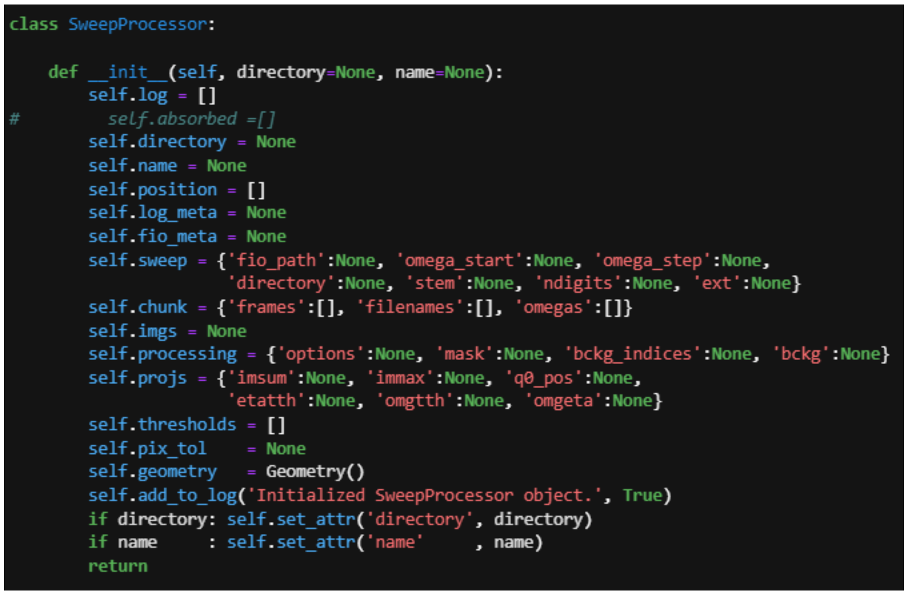
1) class **SweepProcessor.py** (here is all related to evaluating images and sweepscan metadata)
   a) Defining metadata:

*.fio or *.log file for scan parameters
sample positions, values of pressure or temperature
defining output paths, filenames

b) Loading selected set of images
c) Processing images: image orientation, computing and subtracting the background
d) Initializing or loading geometry parameters from *.par or *.yml files
e) Computing and plotting projections:
   sum and maxima of all images,
   omega-2theta, eta-2theta, omega-eta
f) Computing the optimal set of thresholds for peaksearch
g) Running the peaksearch for the set of thresholds
h) Merging the results, so that peaks of different intensities were evaluated similarly
i) Applying spline if needed
j) Saving output files:
   *.npz – data (archive thresholded images for HEXRD),
   *.tif -projections,
   *.flt and *.spt - results of peaksearch,
   *.yml and *.par - geometry files,
   *.p - object itself
k) Based on all the information create the object for the next step (indexing peaks)

One object for each set of images (1 detector in 1 sweep).

This stage is resource- and time- consuming. But once it is completed, the output data is rather compact and can be straightforwardly used for further analysis using ImageD11 or HEXRD.



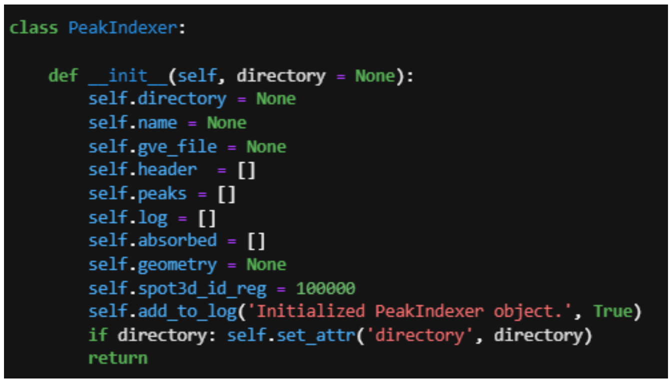
```
class SweepProcessor:

    def __init__(self, directory=None, name=None):
        self.log = []
#         self.absorbed =[]
        self.directory = None
        self.name = None
        self.position = []
        self.log_meta = None
        self.fio_meta = None
        self.sweep = {'fio_path':None, 'omega_start':None, 'omega_step':None,
                      'directory':None, 'stem':None, 'ndigits':None, 'ext':None}
        self.chunk = {'frames':[], 'filenames':[], 'omegas':[]}
        self.imgs = None
        self.processing = {'options':None, 'mask':None, 'bckg_indices':None, 'bckg':None}
        self.projs = {'imsum':None, 'immax':None, 'q0_pos':None,
                      'etatth':None, 'omgtth':None, 'omgeta':None}
        self.thresholds = []
        self.pix_tol     = None
        self.geometry    = Geometry()
        self.add_to_log('Initialized SweepProcessor object.', True)
        if directory: self.set_attr('directory', directory)
        if name     : self.set_attr('name'     , name)
        return
```

Methods:
    add_to_log(str_to_add, also_print)
    set_attr(attr, value))

add_to_attr(attr, value))
print(also_log)
load_sweep(omega_start, omega_step, directory, stem, ndigits, ext, frames)
process_imgs(options, mask, bckg)
calculate_projections(q0_pos, rad_ranges)
plot()
calculate_thresholds()
export_data(thr)
save_peaksearch_yaml(thresholds, pix_tol, spline_path)
run_peaksearch(yaml_file, use_imgs, use_temp_tifs, del_temp_tifs)
merge_peaks(yaml_file)
crop_imgs(roi)
save_tifs()
delete_tifs()
generate_PeakIndexer(directory, name)
calculate_bckg(imgs, indices)
apply_spline_to_fltfile(flt_file_in, flt_file_out, spline, sc_dim)

2) class **PeakIndexer.py** (here is all related to indexing, geometry, and calibration)
   a) Loading peaks from *.flt files, loading geometry from *.par or *.yml files
   b) Indexing
   c) Filtering
   d) "Moving" detector if needed
   e) Merging several *.flt files for this detector is needed.
   f) Saving output files:
      *.par, *.yml – geometry files,
      *.gve – gvectors (result of indexing),
      *.p - object itself
   g) Based on all the information create the object for the next step (evaluation of gvectors)

One object for each set of images (1 detector in 1 sweep).

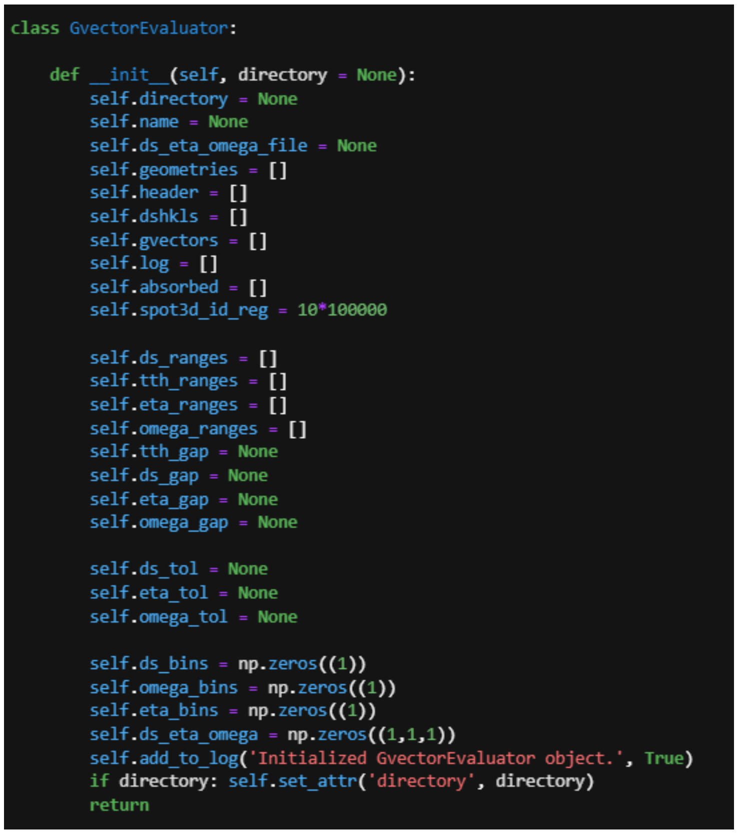

```
class PeakIndexer:

    def __init__(self, directory = None):
        self.directory = None
        self.name = None
        self.gve_file = None
        self.header  = []
        self.peaks = []
        self.log = []
        self.absorbed = []
        self.geometry = None
        self.spot3d_id_reg = 100000
        self.add_to_log('Initialized PeakIndexer object.', True)
        if directory: self.set_attr('directory', directory)
        return
```

Methods:
add_to_log(str_to_add, also_print)
set_attr(attr, value))

add_to_attr(attr, value))
print(also_log)
load_flt(directory, flt_file)
save_flt(directory, flt_file, overwrite)
remove_not_in_range(int_range)
absorb(x, spot3d_id_reg)
run_indexer(directory, par_file, gve_file)
generate_GvectorEvaluator(directory, name)

3) class **GvectorEvaluator.py** (here is all related to gvectors)
   a) Loading gvectors from *.gve files
   b) Filtering and grouping (if needed)
   c) Determination of eta, omega, 2theta ranges
   d) Calculate and plot histograms eta-2theta, omega-2theta
   e) Merging several *.gve files if multiple detectors were used.
   f) Saving output files:
      *.gve – gvectors,
      *.p - object itself
   g) Based on all the information create the object for the next step (grain spotting)

One object for 1 sweep (but multiple detectors).

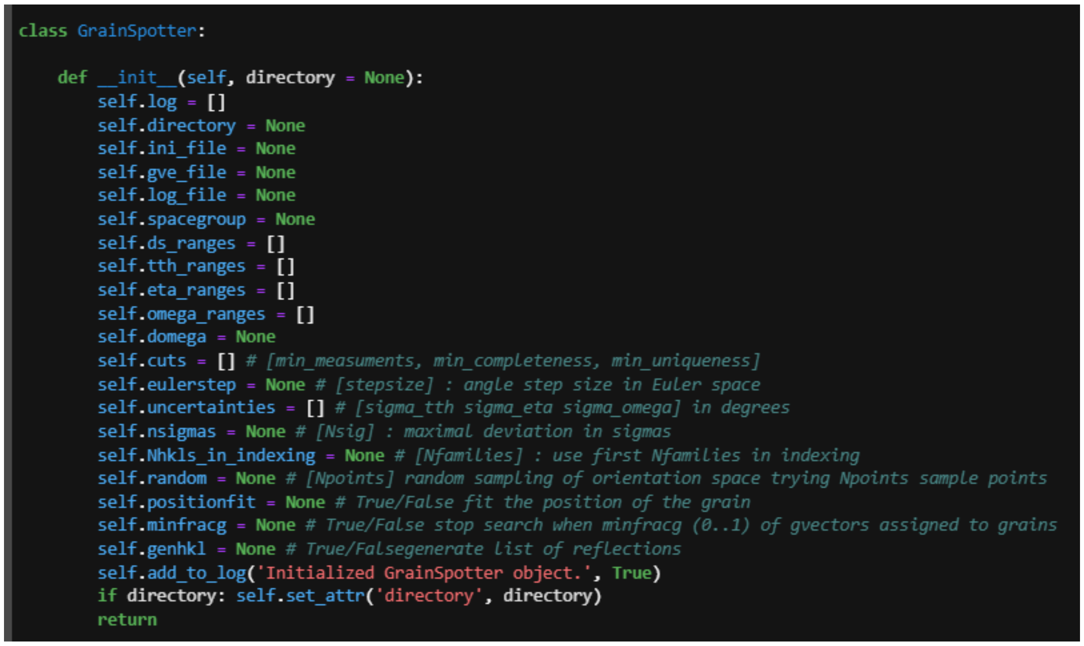

```
class GvectorEvaluator:

    def __init__(self, directory = None):
        self.directory = None
        self.name = None
        self.ds_eta_omega_file = None
        self.geometries = []
        self.header = []
        self.dshkls = []
        self.gvectors = []
        self.log = []
        self.absorbed = []
        self.spot3d_id_reg = 10*100000

        self.ds_ranges = []
        self.tth_ranges = []
        self.eta_ranges = []
        self.omega_ranges = []
        self.tth_gap = None
        self.ds_gap = None
        self.eta_gap = None
        self.omega_gap = None

        self.ds_tol = None
        self.eta_tol = None
        self.omega_tol = None

        self.ds_bins = np.zeros((1))
        self.omega_bins = np.zeros((1))
        self.eta_bins = np.zeros((1))
        self.ds_eta_omega = np.zeros((1,1,1))
        self.add_to_log('Initialized GvectorEvaluator object.', True)
        if directory: self.set_attr('directory', directory)
        return
```

Methods:
add_to_log(str_to_add, also_print)

set_attr(attr, value))
add_to_attr(attr, value))
print(also_log)
load_gve(directory, gve_file)
save_gve(directory, gve_file, overwrite)
calculate_ranges(tth_gap, ds_gap, eta_gap, omega_gap)
remove_not_ranges(ds_ranges, tth_ranges, omega_ranges, eta_ranges)
group_gvectors(ds_tol, eta_tol, omega_tol)
remove_not_inrings()
calc_histo(omega_pixsize, eta_pixsize, ds_eta_omega_file, plot, save_arrays)
absorb(x, spot3d_id_reg)

4) class **GrainSpotter.py** (here is all related to grain spotting)
   a) Loading gvectors from *.gve files
   b) Initializing or loading grainsearch parameters from *.ini file
   c) Running grainspotter.py
   d) Saving output files:
      *.gff, *.log – results grainspotter.py,
      *.p - object itself
   e) Based on all the information create the object for the next step (simulation for each grain)

One object for 1 sweep (but multiple detectors).

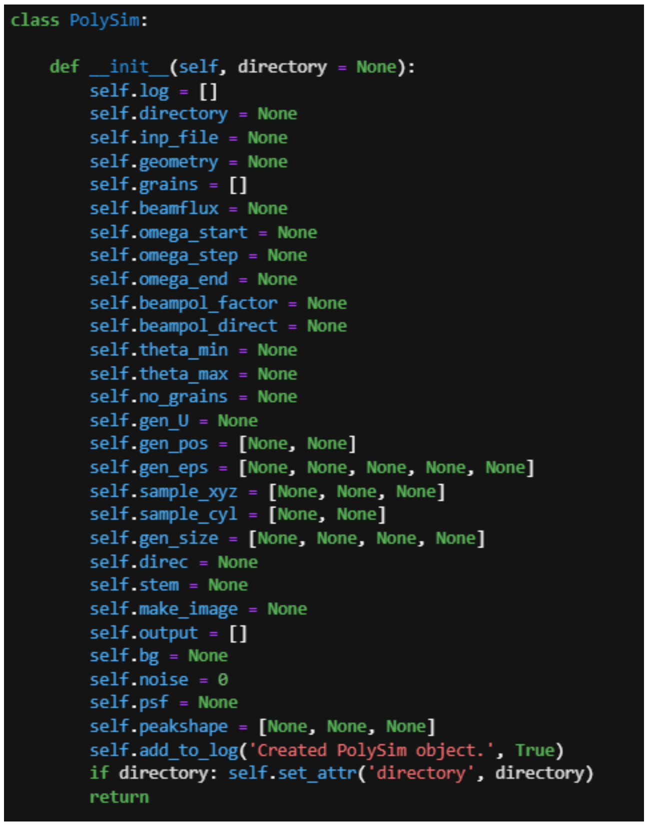


```python
class GrainSpotter:

    def __init__(self, directory = None):
        self.log = []
        self.directory = None
        self.ini_file = None
        self.gve_file = None
        self.log_file = None
        self.spacegroup = None
        self.ds_ranges = []
        self.tth_ranges = []
        self.eta_ranges = []
        self.omega_ranges = []
        self.domega = None
        self.cuts = [] # [min_measuments, min_completeness, min_uniqueness]
        self.eulerstep = None # [stepsize] : angle step size in Euler space
        self.uncertainties = [] # [sigma_tth sigma_eta sigma_omega] in degrees
        self.nsigmas = None # [Nsig] : maximal deviation in sigmas
        self.Nhkls_in_indexing = None # [Nfamilies] : use first Nfamilies in indexing
        self.random = None # [Npoints] random sampling of orientation space trying Npoints sample points
        self.positionfit = None # True/False fit the position of the grain
        self.minfracg = None # True/False stop search when minfracg (0..1) of gvectors assigned to grains
        self.genhkl = None # True/Falsegenerate list of reflections
        self.add_to_log('Initialized GrainSpotter object.', True)
        if directory: self.set_attr('directory', directory)
        return
```

Methods:
add_to_log(str_to_add, also_print)
set_attr(attr, value))
add_to_attr(attr, value))
print(also_log)
load_ini(directory, ini_file)
save_ini(directory, ini_file, overwrite)

run_grainspotter(directory, ini_file, gve_file, log_file)

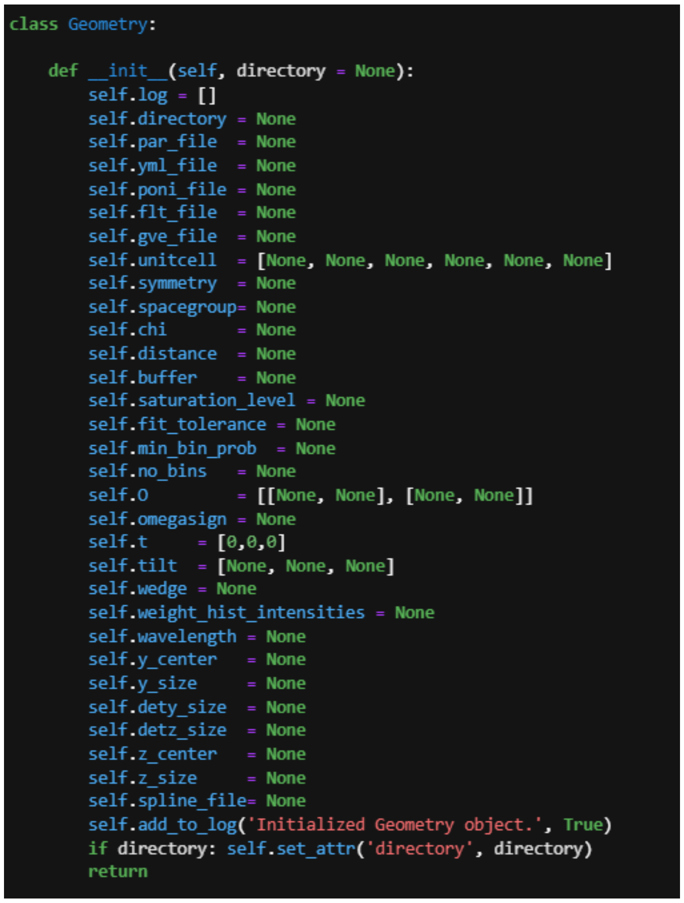
5) class **PolySim.py** (here is all related to simulation for individual grains)
   a) Loading grains from *.gff or *.log files
   b) Initializing or loading PolyXSim parameters from *.inp file
   c) Running PolyXSim for each grain
   d) Register expected peaks (from simulations)
   e) Plot measured vs expected peaks
   f) Saving output files:
      *.tif images if needed
      *.gve, *.par – results of PolyXSim.py,
      *.p - object itself



```
class PolySim:

    def __init__(self, directory = None):
        self.log = []
        self.directory = None
        self.inp_file = None
        self.geometry = None
        self.grains = []
        self.beamflux = None
        self.omega_start = None
        self.omega_step = None
        self.omega_end = None
        self.beampol_factor = None
        self.beampol_direct = None
        self.theta_min = None
        self.theta_max = None
        self.no_grains = None
        self.gen_U = None
        self.gen_pos = [None, None]
        self.gen_eps = [None, None, None, None, None]
        self.sample_xyz = [None, None, None]
        self.sample_cyl = [None, None]
        self.gen_size = [None, None, None, None]
        self.direc = None
        self.stem = None
        self.make_image = None
        self.output = []
        self.bg = None
        self.noise = 0
        self.psf = None
        self.peakshape = [None, None, None]
        self.add_to_log('Created PolySim object.', True)
        if directory: self.set_attr('directory', directory)
        return
```

Methods:
    add_to_log(str_to_add, also_print)
    set_attr(attr, value))
    add_to_attr(attr, value))
    print(also_log)
    load_inp(directory, inp_file)
    save_inp(directory, inp_file, overwrite)
    run_PolyXsim(directory, inp_file)

6) class **Geometry.py** (here is all geometry parameters)



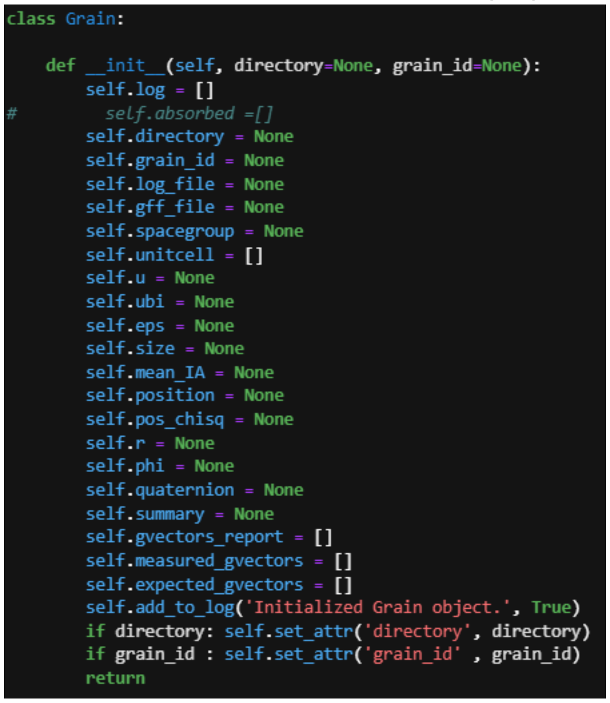
```
class Geometry:

    def __init__(self, directory = None):
        self.log = []
        self.directory = None
        self.par_file  = None
        self.yml_file  = None
        self.poni_file = None
        self.flt_file  = None
        self.gve_file  = None
        self.unitcell  = [None, None, None, None, None, None]
        self.symmetry  = None
        self.spacegroup= None
        self.chi       = None
        self.distance  = None
        self.buffer    = None
        self.saturation_level = None
        self.fit_tolerance = None
        self.min_bin_prob  = None
        self.no_bins   = None
        self.O         = [[None, None], [None, None]]
        self.omegasign = None
        self.t     = [0,0,0]
        self.tilt  = [None, None, None]
        self.wedge = None
        self.weight_hist_intensities = None
        self.wavelength = None
        self.y_center   = None
        self.y_size     = None
        self.dety_size  = None
        self.detz_size  = None
        self.z_center   = None
        self.z_size     = None
        self.spline_file= None
        self.add_to_log('Initialized Geometry object.', True)
        if directory: self.set_attr('directory', directory)
        return
```

Methods:

add_to_log(str_to_add, also_print)
set_attr(attr, value))
add_to_attr(attr, value))
print(also_log)
load_par(directory, par_file)
save_par(directory, par_file, overwrite)
run_PolyXsim(directory, inp_file)

from_hexrd_definitions(pars, det_num = 1)
into_hexrd_definitions(det_num = 1)
move_detector(shift_xyz_in_mm)
load_yml(directory, yml_file, det_num = 1)
save_yml(directory, yml_file, overwrite)
save_geometries_as_yml(list_of_geometries, directory, yml_file, overwrite)
load_poni(directory, poni_file)
save_poni(directory, poni_file, overwrite)
average_geometries(list_of_geometries)

7) class **Grain.py** (here is all related to a single grain)

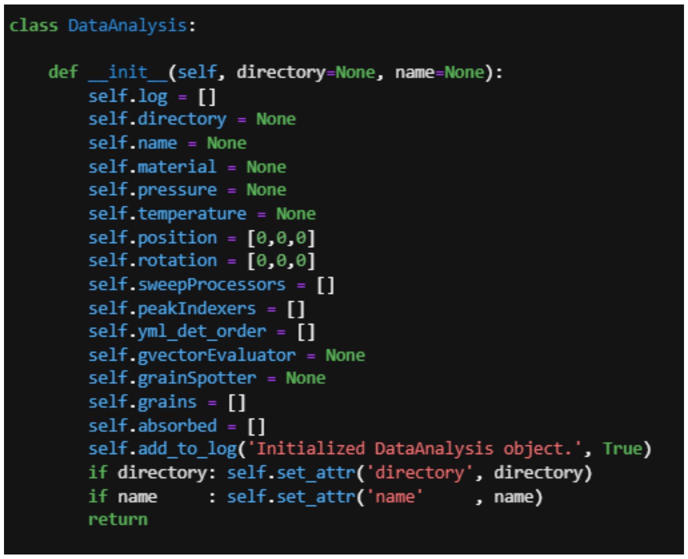


```
class Grain:

    def __init__(self, directory=None, grain_id=None):
        self.log = []
#         self.absorbed =[]
        self.directory = None
        self.grain_id = None
        self.log_file = None
        self.gff_file = None
        self.spacegroup = None
        self.unitcell = []
        self.u = None
        self.ubi = None
        self.eps = None
        self.size = None
        self.mean_IA = None
        self.position = None
        self.pos_chisq = None
        self.r = None
        self.phi = None
        self.quaternion = None
        self.summary = None
        self.gvectors_report = []
        self.measured_gvectors = []
        self.expected_gvectors = []
        self.add_to_log('Initialized Grain object.', True)
        if directory: self.set_attr('directory', directory)
        if grain_id : self.set_attr('grain_id' , grain_id)
        return
```

Methods:

add_to_log(str_to_add, also_print)
set_attr(attr, value))
add_to_attr(attr, value))
print(also_log)
load_log(directory, log_file)
load_gff(directory, gff_file)
save_gff(directory, gff_file, list_of_grains, overwrite)
identify_measured(gvectors)
simulate_gvectors(geometry, omega_range, tth_range, beamflux, bckg, psf, peakshape)
plot_measured_vs_expected()
load_poni(directory, poni_file)
save_poni(directory, poni_file, overwrite)
average_geometries(list_of_geometries)

8) class **DataAnalysis.py** (here is the place where all the blocks are used)

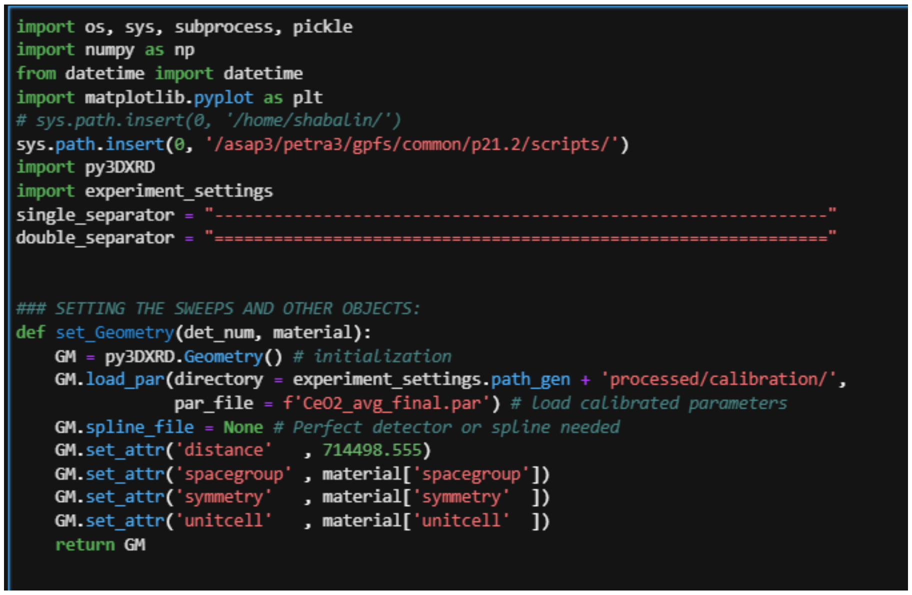


```
class DataAnalysis:

    def __init__(self, directory=None, name=None):
        self.log = []
        self.directory = None
        self.name = None
        self.material = None
        self.pressure = None
        self.temperature = None
        self.position = [0,0,0]
        self.rotation = [0,0,0]
        self.sweepProcessors = []
        self.peakIndexers = []
        self.yml_det_order = []
        self.gvectorEvaluator = None
        self.grainSpotter = None
        self.grains = []
        self.absorbed = []
        self.add_to_log('Initialized DataAnalysis object.', True)
        if directory: self.set_attr('directory', directory)
        if name     : self.set_attr('name'     , name)
        return
```

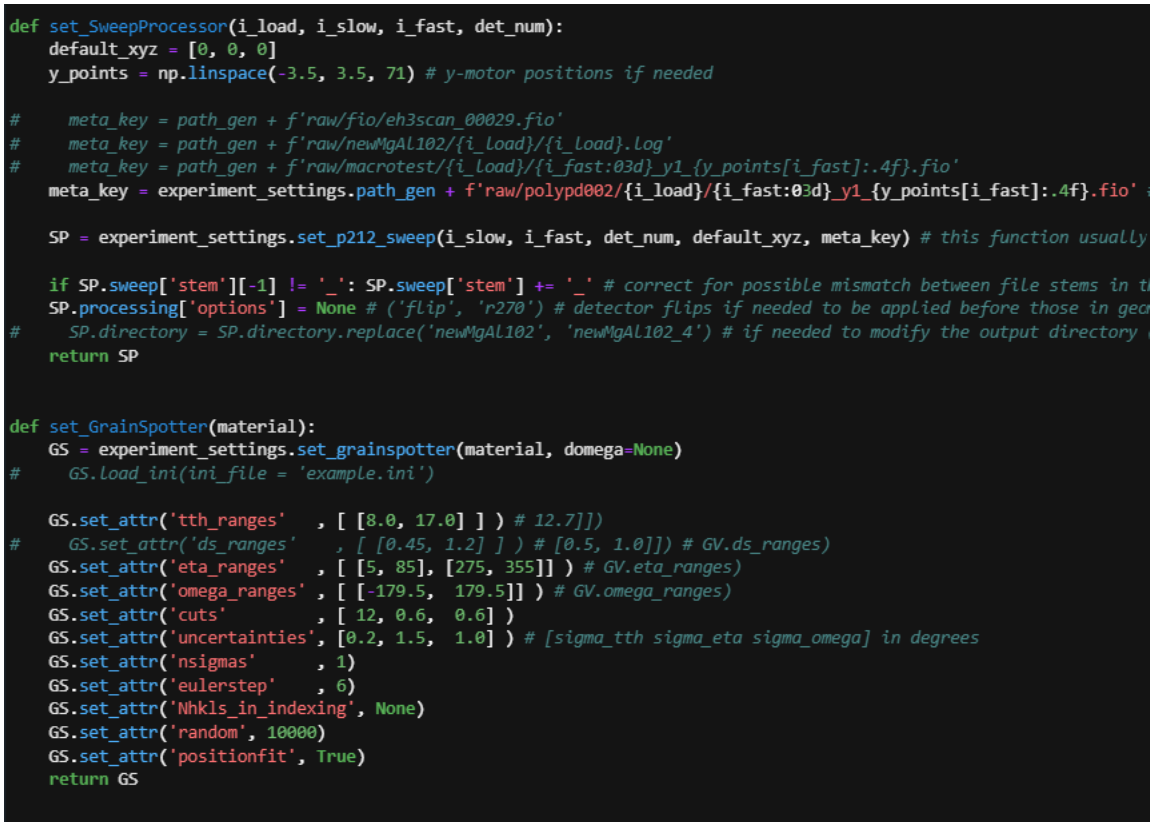
Methods:

add_to_log(str_to_add, also_print)
set_attr(attr, value))
add_to_attr(attr, value))
print(also_log)
save_geometries_as_yml(yml_det_order)
process_images(frames, save_tifs, q0_pos, rad_ranges, thr)
peaksearch(peaksearch_thresholds, peakmerge_thresholds, min_peak_dist, use_temp_tifs, del_temp_tifs)
index(move_det_xyz_mm)
evaluateGvectors(tth_gap, ds_gap, eta_gap, omega_gap, to_plot, save_arrays)
searchGrains(grainSpotter)
runPolyXSim(polyxsim)
mark_peaks()
plot_sinogram(list_DATApaths)
set_MultiDATA(list_DATApaths, num_0)

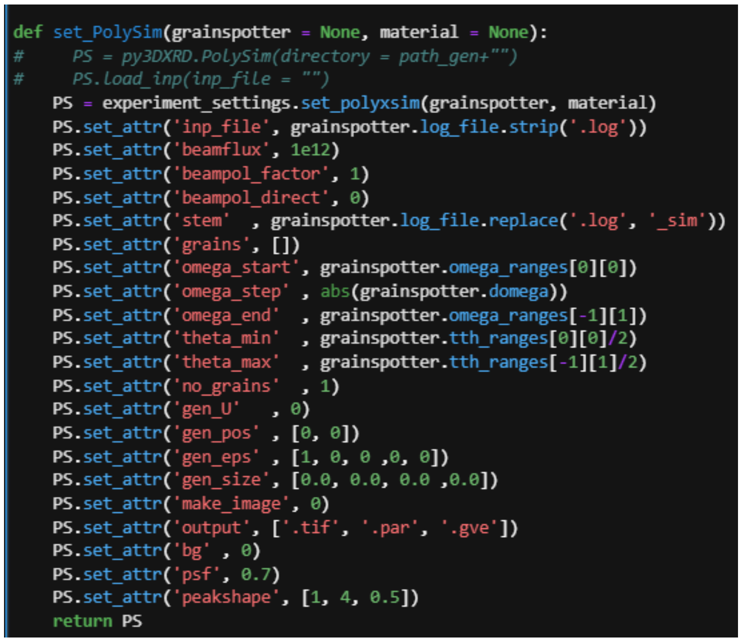
Example of usage:



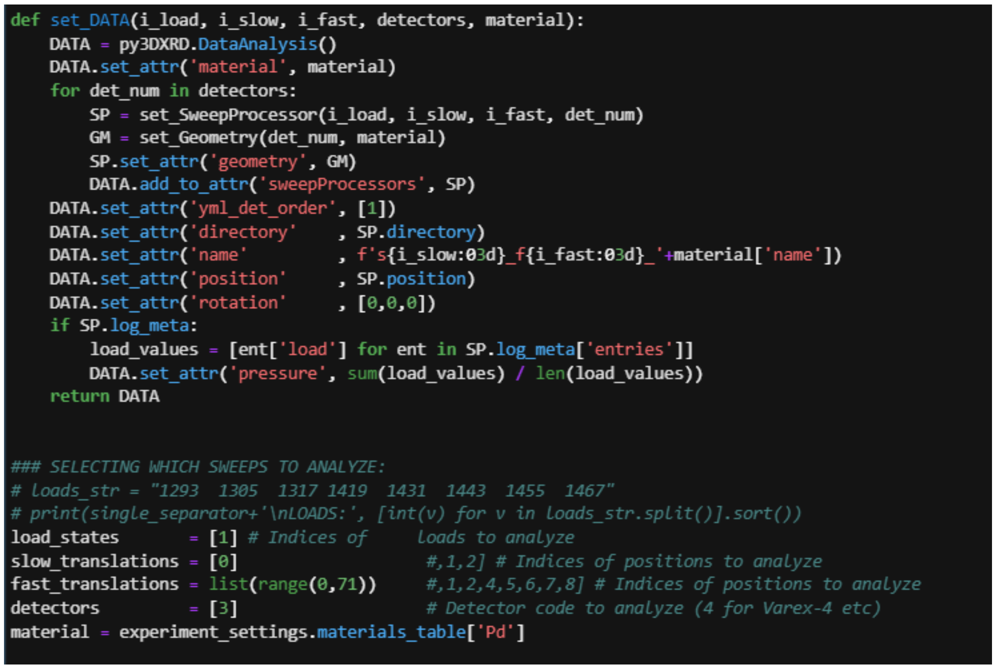
```python
import os, sys, subprocess, pickle
import numpy as np
from datetime import datetime
import matplotlib.pyplot as plt
# sys.path.insert(0, '/home/shabalin/')
sys.path.insert(0, '/asap3/petra3/gpfs/common/p21.2/scripts/')
import py3DXRD
import experiment_settings
single_separator = "--------------------------------------------------------------"
double_separator = "=============================================================="


### SETTING THE SWEEPS AND OTHER OBJECTS:
def set_Geometry(det_num, material):
    GM = py3DXRD.Geometry() # initialization
    GM.load_par(directory = experiment_settings.path_gen + 'processed/calibration/',
                par_file = f'CeO2_avg_final.par') # load calibrated parameters
    GM.spline_file = None # Perfect detector or spline needed
    GM.set_attr('distance'    , 714498.555)
    GM.set_attr('spacegroup' , material['spacegroup'])
    GM.set_attr('symmetry'    , material['symmetry'  ])
    GM.set_attr('unitcell'    , material['unitcell'  ])
    return GM
```



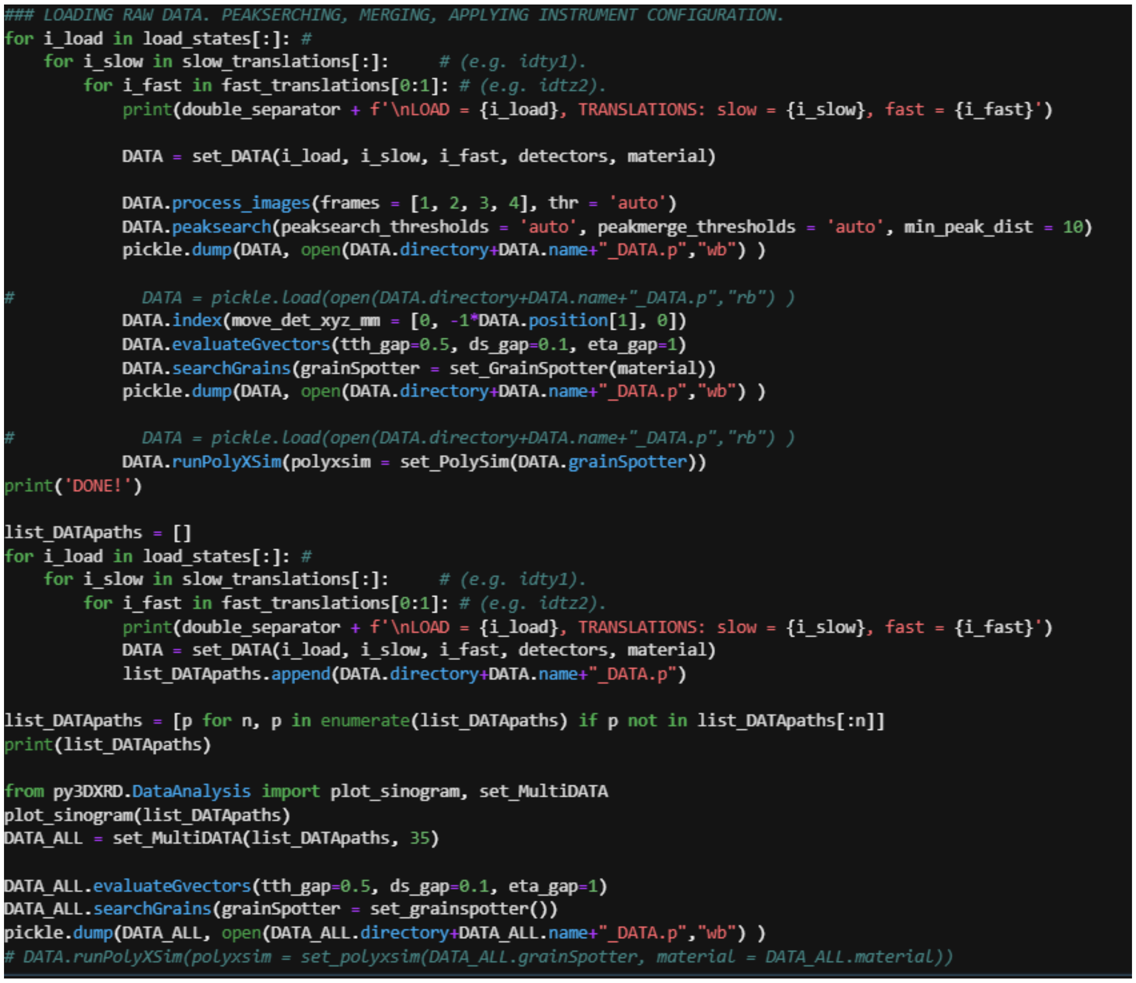
```python
def set_SweepProcessor(i_load, i_slow, i_fast, det_num):
    default_xyz = [0, 0, 0]
    y_points = np.linspace(-3.5, 3.5, 71) # y-motor positions if needed

#     meta_key = path_gen + f'raw/fio/eh3scan_00029.fio'
#     meta_key = path_gen + f'raw/newMgAl102/{i_load}/{i_load}.log'
#     meta_key = path_gen + f'raw/macrotest/{i_load}/{i_fast:03d}_y1_{y_points[i_fast]:.4f}.fio'
    meta_key = experiment_settings.path_gen + f'raw/polypd002/{i_load}/{i_fast:03d}_y1_{y_points[i_fast]:.4f}.fio' 

    SP = experiment_settings.set_p212_sweep(i_slow, i_fast, det_num, default_xyz, meta_key) # this function usually

    if SP.sweep['stem'][-1] != '_': SP.sweep['stem'] += '_' # correct for possible mismatch between file stems in t
    SP.processing['options'] = None # ('flip', 'r270') # detector flips if needed to be applied before those in geo
#     SP.directory = SP.directory.replace('newMgAl102', 'newMgAl102_4') # if needed to modify the output directory 
    return SP


def set_GrainSpotter(material):
    GS = experiment_settings.set_grainspotter(material, domega=None)
#     GS.load_ini(ini_file = 'example.ini')

    GS.set_attr('tth_ranges'    , [ [8.0, 17.0] ] ) # 12.7]])
#     GS.set_attr('ds_ranges'     , [ [0.45, 1.2] ] ) # [0.5, 1.0]]) # GV.ds_ranges)
    GS.set_attr('eta_ranges'    , [ [5, 85], [275, 355]] ) # GV.eta_ranges)
    GS.set_attr('omega_ranges' , [ [-179.5,  179.5]] ) # GV.omega_ranges)
    GS.set_attr('cuts'          , [ 12, 0.6,  0.6] )
    GS.set_attr('uncertainties', [0.2, 1.5,  1.0] ) # [sigma_tth sigma_eta sigma_omega] in degrees
    GS.set_attr('nsigmas'       , 1)
    GS.set_attr('eulerstep'     , 6)
    GS.set_attr('Nhkls_in_indexing', None)
    GS.set_attr('random', 10000)
    GS.set_attr('positionfit', True)
    return GS
```



```python
def set_PolySim(grainspotter = None, material = None):
#     PS = py3DXRD.PolySim(directory = path_gen+"")
#     PS.load_inp(inp_file = "")
    PS = experiment_settings.set_polyxsim(grainspotter, material)
    PS.set_attr('inp_file', grainspotter.log_file.strip('.log'))
    PS.set_attr('beamflux', 1e12)
    PS.set_attr('beampol_factor', 1)
    PS.set_attr('beampol_direct', 0)
    PS.set_attr('stem'   , grainspotter.log_file.replace('.log', '_sim'))
    PS.set_attr('grains', [])
    PS.set_attr('omega_start', grainspotter.omega_ranges[0][0])
    PS.set_attr('omega_step' , abs(grainspotter.domega))
    PS.set_attr('omega_end'  , grainspotter.omega_ranges[-1][1])
    PS.set_attr('theta_min'  , grainspotter.tth_ranges[0][0]/2)
    PS.set_attr('theta_max'  , grainspotter.tth_ranges[-1][1]/2)
    PS.set_attr('no_grains'  , 1)
    PS.set_attr('gen_U'    , 0)
    PS.set_attr('gen_pos' , [0, 0])
    PS.set_attr('gen_eps' , [1, 0, 0 ,0, 0])
    PS.set_attr('gen_size', [0.0, 0.0, 0.0 ,0.0])
    PS.set_attr('make_image', 0)
    PS.set_attr('output', ['.tif', '.par', '.gve'])
    PS.set_attr('bg' , 0)
    PS.set_attr('psf', 0.7)
    PS.set_attr('peakshape', [1, 4, 0.5])
    return PS
```

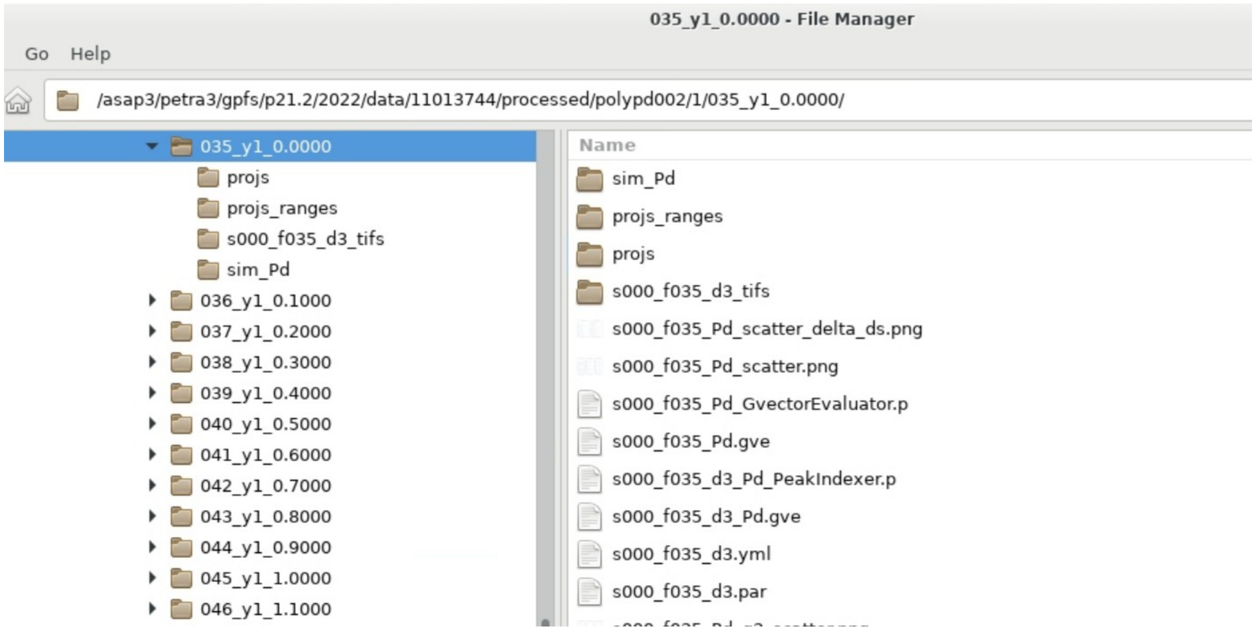


```python
def set_DATA(i_load, i_slow, i_fast, detectors, material):
    DATA = py3DXRD.DataAnalysis()
    DATA.set_attr('material', material)
    for det_num in detectors:
        SP = set_SweepProcessor(i_load, i_slow, i_fast, det_num)
        GM = set_Geometry(det_num, material)
        SP.set_attr('geometry', GM)
        DATA.add_to_attr('sweepProcessors', SP)
    DATA.set_attr('yml_det_order', [1])
    DATA.set_attr('directory'    , SP.directory)
    DATA.set_attr('name'         , f's{i_slow:03d}_f{i_fast:03d}_'+material['name'])
    DATA.set_attr('position'     , SP.position)
    DATA.set_attr('rotation'     , [0,0,0])
    if SP.log_meta:
        load_values = [ent['load'] for ent in SP.log_meta['entries']]
        DATA.set_attr('pressure', sum(load_values) / len(load_values))
    return DATA


### SELECTING WHICH SWEEPS TO ANALYZE:
# loads_str = "1293  1305  1317 1419  1431  1443  1455  1467"
# print(single_separator+'\nLOADS:', [int(v) for v in loads_str.split()].sort())
load_states       = [1] # Indices of     loads to analyze
slow_translations = [0]                  #,1,2] # Indices of positions to analyze
fast_translations = list(range(0,71))    #,1,2,4,5,6,7,8] # Indices of positions to analyze
detectors         = [3]                  # Detector code to analyze (4 for Varex-4 etc)
material = experiment_settings.materials_table['Pd']
```



```python
### LOADING RAW DATA. PEAKSERCHING, MERGING, APPLYING INSTRUMENT CONFIGURATION.
for i_load in load_states[:]: #
    for i_slow in slow_translations[:]:     # (e.g. idty1).
        for i_fast in fast_translations[0:1]: # (e.g. idtz2).
            print(double_separator + f'\nLOAD = {i_load}, TRANSLATIONS: slow = {i_slow}, fast = {i_fast}')

            DATA = set_DATA(i_load, i_slow, i_fast, detectors, material)

            DATA.process_images(frames = [1, 2, 3, 4], thr = 'auto')
            DATA.peaksearch(peaksearch_thresholds = 'auto', peakmerge_thresholds = 'auto', min_peak_dist = 10)
            pickle.dump(DATA, open(DATA.directory+DATA.name+"_DATA.p","wb") )

#             DATA = pickle.load(open(DATA.directory+DATA.name+"_DATA.p","rb") )
            DATA.index(move_det_xyz_mm = [0, -1*DATA.position[1], 0])
            DATA.evaluateGvectors(tth_gap=0.5, ds_gap=0.1, eta_gap=1)
            DATA.searchGrains(grainSpotter = set_GrainSpotter(material))
            pickle.dump(DATA, open(DATA.directory+DATA.name+"_DATA.p","wb") )

#             DATA = pickle.load(open(DATA.directory+DATA.name+"_DATA.p","rb") )
            DATA.runPolyXSim(polyxsim = set_PolySim(DATA.grainSpotter))
print('DONE!')

list_DATApaths = []
for i_load in load_states[:]: #
    for i_slow in slow_translations[:]:     # (e.g. idty1).
        for i_fast in fast_translations[0:1]: # (e.g. idtz2).
            print(double_separator + f'\nLOAD = {i_load}, TRANSLATIONS: slow = {i_slow}, fast = {i_fast}')
            DATA = set_DATA(i_load, i_slow, i_fast, detectors, material)
            list_DATApaths.append(DATA.directory+DATA.name+"_DATA.p")

list_DATApaths = [p for n, p in enumerate(list_DATApaths) if p not in list_DATApaths[:n]]
print(list_DATApaths)

from py3DXRD.DataAnalysis import plot_sinogram, set_MultiDATA
plot_sinogram(list_DATApaths)
DATA_ALL = set_MultiDATA(list_DATApaths, 35)

DATA_ALL.evaluateGvectors(tth_gap=0.5, ds_gap=0.1, eta_gap=1)
DATA_ALL.searchGrains(grainSpotter = set_grainspotter())
pickle.dump(DATA_ALL, open(DATA_ALL.directory+DATA_ALL.name+"_DATA.p","wb") )
# DATA.runPolyXSim(polyxsim = set_polyxsim(DATA_ALL.grainSpotter, material = DATA_ALL.material))
```

Output:



```
035_y1_0.0000 - File Manager

/asap3/petra3/gpfs/p21.2/2022/data/11013744/processed/polypd002/1/035_y1_0.0000/

035_y1_0.0000                 Name
    projs                     sim_Pd
    projs_ranges              projs_ranges
    s000_f035_d3_tifs         projs
    sim_Pd                    s000_f035_d3_tifs
036_y1_0.1000                 s000_f035_Pd_scatter_delta_ds.png
037_y1_0.2000                 s000_f035_Pd_scatter.png
038_y1_0.3000                 s000_f035_Pd_GvectorEvaluator.p
039_y1_0.4000                 s000_f035_Pd.gve
040_y1_0.5000                 s000_f035_d3_Pd_PeakIndexer.p
041_y1_0.6000                 s000_f035_d3_Pd.gve
042_y1_0.7000                 s000_f035_d3.yml
043_y1_0.8000                 s000_f035_d3.par
044_y1_0.9000
045_y1_1.0000
046_y1_1.1000
```