# Introduction and Principles

This is a guide how to index CHESS datasets using minimal computational resources. This example will show you how to index a 3scan (so called “regular anitascan”) dataset of the sample C221110b\_20uc obtained at 200K and 16keV.

In general, indexing means finding the orientation of the crystal system relative to the lab frame and then using this orientation (encoded in the UB matrix) to process the detector images to populate a 3D reciprocal space with the experimental data. Importantly, this process relies on a priori information about the crystal type. Peaks in the experimental data are assigned based on this crystal type. If there are spurious or double peaks using an a priori crystal type to index the data can be quite difficult.

A minimizer tries to assign the peaks in the raw data to reflections of the crystal. The minimization is best when the peaks in the raw data land on integer numbers of the reciprocal space lattice.

In general, the workflow for indexing a dataset goes as follows

1. Stack the data
2. Find peaks in the data
3. Using these peaks, find a rotation and stretching matrix (UBmatrix) which transforms these peaks to the reciprocal space of the system you are indexing to.
4. Using the good UBmatrix process every image to populate the reciprocal space grid.

The guiding principle is to use as few computational resources as possible. The problems are

1. The data take up alot of diskspace (64GB for the 3scan C221110b\_20uc dataset). It takes several hours to copy 3600 cbfs from the classe to the singer server. We cant use the classe system when we dont have beamtime.
2. Finding the correct UB matrix by guessing is a minimization problem which requires alot of CPU power (time). The smaller and more accurate your peak list is the faster and better the UB matrix finding process will be.
3. Processing 3600 .cbf images each with a size of ~10MB is computationally expensive and will take ~2 hours on the singer server.

So to summarize the computer things which slow us down are

1. Copying the data
2. Running the minimizer to find a good UB matrix
3. Actually indexing the data

# Strategy

To reduce the computational resource needed to process the dataset we will make use of several tricks

1. Stack the metadata instead of the data. Stacks used to be a list of images with their metadata. Now they contain the metadata only so that they can be made in <1min. Stacks are created on the singergroup server
2. Find the peaks on the classe server. This way we dont have to copy the images over to find a UB matrix.
3. Identify suitable peaks by powderizing the data. Looking at a |Q| vs intensity plot of the data in one of the three scans allows us to get an idea of which peaks are due to the substrate and hence which peaks are good for feeding the UB matrix finding minimizer.
4. Only once a good UB matrix has been identified do you copy over the images.
5. Index the data on the singerlab server

# Making the Stacks

The stack making code is run on the singerlab server

You need to copy the files over from classe server

sudo screen rsync -avz --verbose USERNAME@lnx201.classe.cornell.edu:/nfs/chess/id4baux/2022-3/singer-3456-a/FILEPATH

The code is located at /home/jg2364/chess\_oct2022/new\_indexing/MakeStacking.py

You might have to set up your conda to install the packages for this work. Otherwise you could copy the code to a jupyter notebook or change your python path.

You also need to give the following to the stack program

PONI of the experiment:  
/home/jg2364/chess\_oct2022/calibration/ceria16keV.poni

Mask of the experiment

/home/jg2364/chess\_oct2022/calibration/mask\_16keV.edf

Specfile: /home/jg2364/chess\_oct2022/new\_indexing/specfiles/C221110b\_20uc

The scan numbers: 16, 17, 18

The sample name

The stack will be produced by the script

# Making the Peaklists

Peaklists are made on the classe server.

You need to ssh into the classe server

ssh jg2364@lnx201.classe.cornell.edu

Two factor authentication needs to be enabled.

Every user has a directory where they can run programs and store data.

Its located at /nfs/chess/user/USERNAME/

You will have to install a virtual python environment to run the peaklist finder. The instructions are here:

https://wiki.classe.cornell.edu/Computing/LinuxSoftwareDevelopment#Using\_a\_Python\_Virtual\_Environment\_44\_with\_Python\_3.6.8

You will also need to install the relevant packages (e.g. nexusformat and fabio)

the script to run is either make\_peaklist.py or make\_peaklist\_streak.py. The latter corrects for streaks. For this dataset a ran make\_peaklist.py without the streak corrector.

The codes are here: /home/jg2364/chess\_oct2022/new\_indexing you will need to rsync them to your user directory at classe.

You set the threshold, any pixel with intensity above the threshold will be considered a peak. Here i used 1e4. Im working on displaying an average of the first few pics to be output before the threshold is input

The peaklist is saved in your user directory at classe

# Making the UB matrix

A jupyter notebook is used to make the UBmatrix.

The jupyter notebook for these scans is located at

/home/jg2364/chess\_oct2022/new\_indexing/new\_approach\_STO.ipynb

You will need to give the notebook the located of the stack and the poni file

A threshold will also be needed here.

You also set the system to index to here.

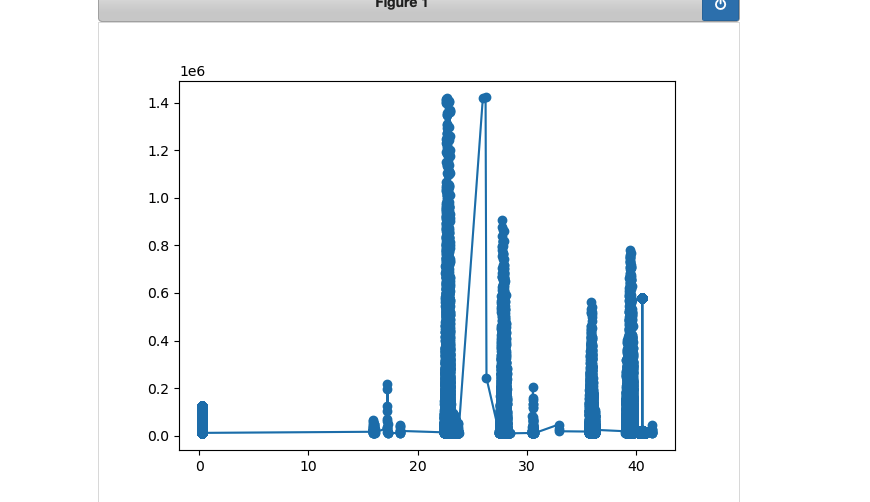
Input is in the line system = xu.materials.Crystal("system", xu.materials.SGLattice( space group, params) ) from xray utils

Input should be in the following way:

Parameters of the system you are indexing to. First enter space group number. Then, depending on the space group number also include 1 to 6 other parameters. Cubic: a. Hexagonal: a and c. Tetragonal: a and c. Orthorhombic: a, b, and c. Monoclinic: a,b, c, and beta. Triclinic: a,b,c, alpha, beta, gamma. So for instance, for space group 221 which is cubic, enter '221, a'. Distances in Angstrom and angles in degrees please.

For STO, the input is 221, 3.905 as the spacegroup of STO is 221 and its cubic and a =3.905 A (right?)

The program then makes a plot of |Q| vs intensity and you have to pick the peaks manually.



For this one i picked the peaks at |Q| of 22.666, 27.749, 35.923, 39.489, always picking around the max value.

Pick the peaks (enter |Q| value of the peaks in the array called the\_good) and then it will run the brute UB matrix minimzer.

You also have to set the Euler angles. For a thin film phi goes from 0 to 360 and, theta is 180 to 181 and chi is 0 to 1. Enter the values in phii, thetaa , and chii respectively.

If you dont get a good chsq immediately, change up the peaks. If the peaks dont land on an integer in rspace if they were obtained at the top of a peak, something is wrong. The peak may not be from the substrate or the stack, poni, mask etc may not be what you think it is. A big problem are the rings because its tough to tell the rings and the substrate peaks apart in the powderized picture.

Then a basin hopping minimizer is run with the best outcome of the brute minimizer. Make sure the bounds are correct.

The UB matrix is output. In addition, the location of the chosen peaks in reciprocal space is shown. These must be very close to integer <0.05 away and the chisq must be less than 0.05 to continue to indexing. Also you can add or delete peaks to the peaklist. Change the peak where the offset from an integer is largest.

# Making the Indexed Object

You have to copy over the images to the singerlab server with rsync before indexing. This can take several hours to a day, depending on how many images are in the scan.

The indexed object is made with a python script. A template can be found at:

/home/jg2364/chess\_oct2022/new\_indexing/MakeIndexing\_anitascan\_STO.py

For the script i used to index the STO substrate data.

This script takes several things

The stack directory

The UB matrix (make sure to put U.T here!!!!)

The image directory (local in singerlab2 only)

The outpath

The range and resolution in HKL space in which you want to populate values

In line 126 enter the directories of the stacks are you indexing.

In line130, make sure the names of the stacks are before %d

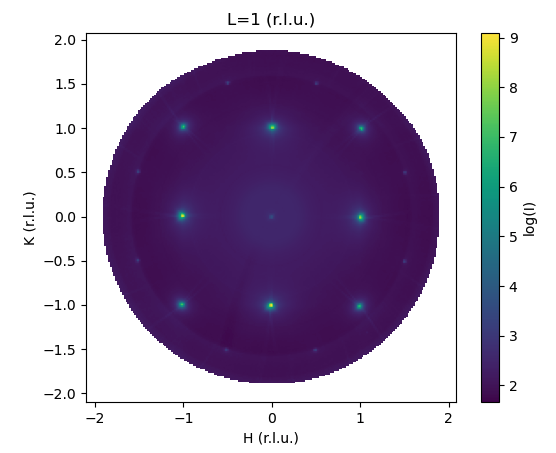
In line 131 subtract from scan\_num the start scan value

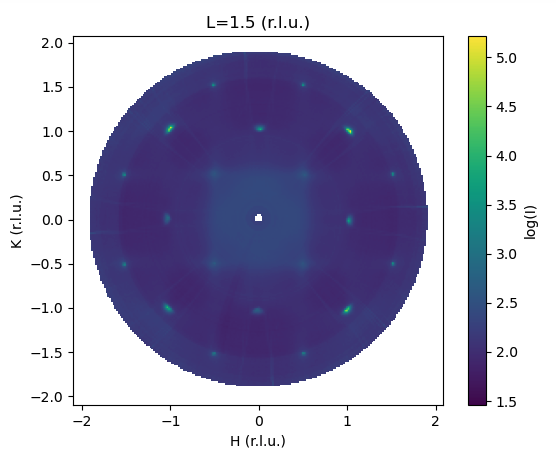
Run the python script, your indexed object will be output

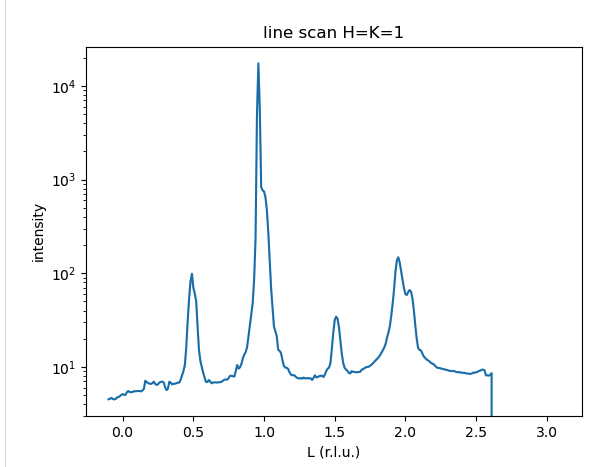
# Visualizing an Indexed Object

The indexed object can be visualized with another jupyter notebook called visualize in new indexing. Further analysis like peak fitting can then be done in jupyter notebooks like this.

It just does some numpy array stuff to look at color plots and linescans like this:







Wow you can see substrate peaks and film peaks and half order film peaks: the dream.