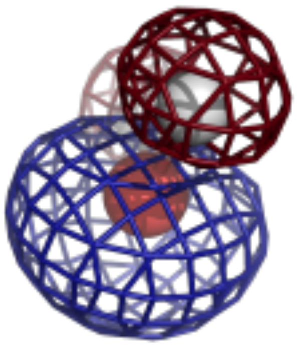
VOLUME TEN



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JULY MMXIX

BETA PEPTIDE LINKS, XFEL GUI, NUMBA, H-BONDS

Table of Contents

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

New Phenix Release

Phenix 1.16 was released prior to the recent change in submission policy by the Protein Data Bank to only accept models solve using X-ray diffraction in the mmCIF format (Adams, P. D. *et al.*, 2019, *Acta Crystallogr. Sect. Struct. Biol.* **75**, 451–454). Changes include a new GUI designed for deposition, *mmtbx.prepare_pdb_deposition*, to create the mmCIF files for deposition into the PDB.

Also, a new tool (CLI and GUI) for getting a validation report from the PDB, *phenix.get_pdb_validation_report*.

Other changes in the addition of sequence checking to Comprehensive Validation for Cryo-EM.

One fundamental change is the inclusion of Amber functionality, by default, in the Phenix installer. This was facilitated by the move to using conda as the installation package manager. A publication is in preparation while the documentation is an ideal source of information.

A new tool, *phenix.hbond*, is available in the nightly and discussed on page 18 of this newletter.

Downloads available at phenix-online.org

Expert advice

Fitting Tip #18 – A subversive kind of misfit "water"

Jane Richardson and Christopher Williams, Duke University

It is common knowledge that a density peak fit as a crystallographic water may not actually be a water molecule. In a previous Fitting Tip (Headd & Richardson 2013) we surveyed examples of four such cases and their separable diagnoses, mostly by the atom type with which they clash: an unidentified ion, part of an unidentified ligand, the start of an unidentified alternate conformation, or a noise peak. Since then, we have documented several other clashing-water situations. Here we show the new case with the most seriously bad impact on the neighboring structure: a water fit into a peak that is really a sidechain atom.

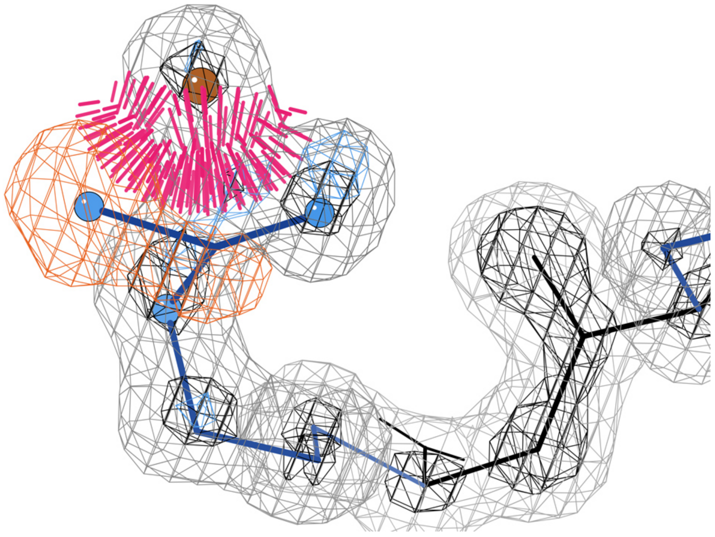


Figure 1: A water (reddish sphere) incorrectly displacing the Nh2 atom of Arg 59 in the 1qLw esterase structure (Bourne 2000). Hotpink spikes flag all-atom clashes >0.4Å, and orange contours represent difference density at -3.5σ. Gray contours are 2mFo-DFc electron density at 1.2σ and black ones at 3σ.

A sidechain can be fit incorrectly for the initial model, usually because of unclear electron density or from a molecular-replacement model with a different rotamer. That produces a difference peak for a real atom that is left outside the model.

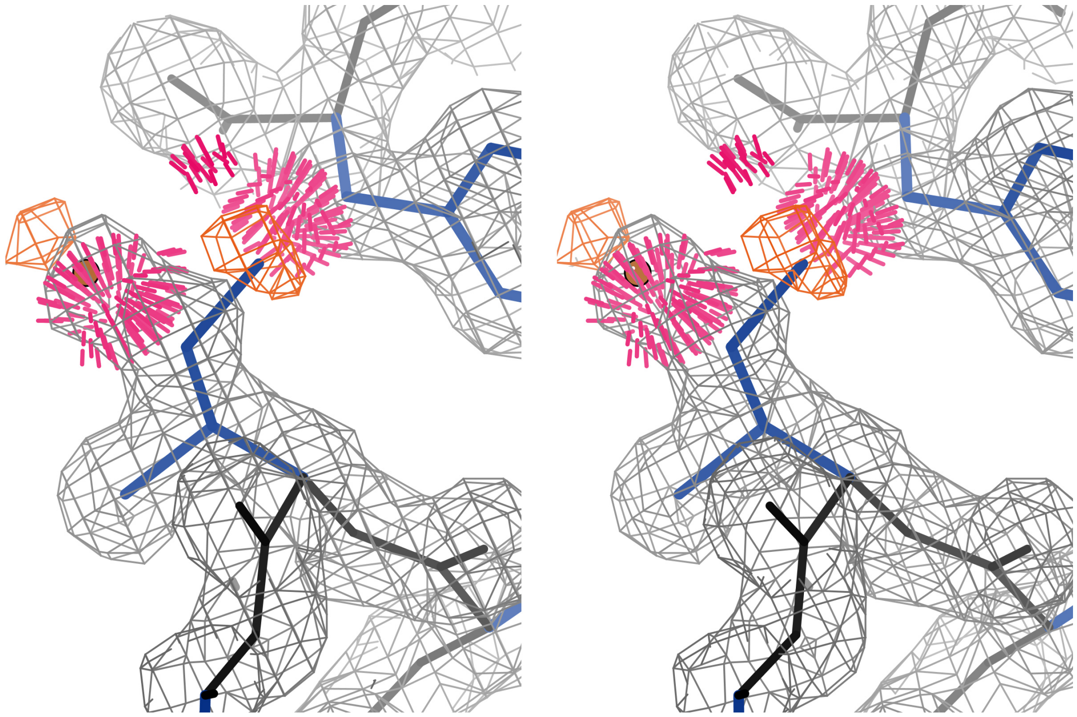


Figure 2: Stereo of a water misfit into the density for the Cd1 atom of Ile 195 in 3js8 (Sagermann 2010). The displaced Cd1 has two bad clashes with other residues. The water (reddish ball) is displaced outward in the density by its unfavorable interaction with Cg, producing another small difference peak.

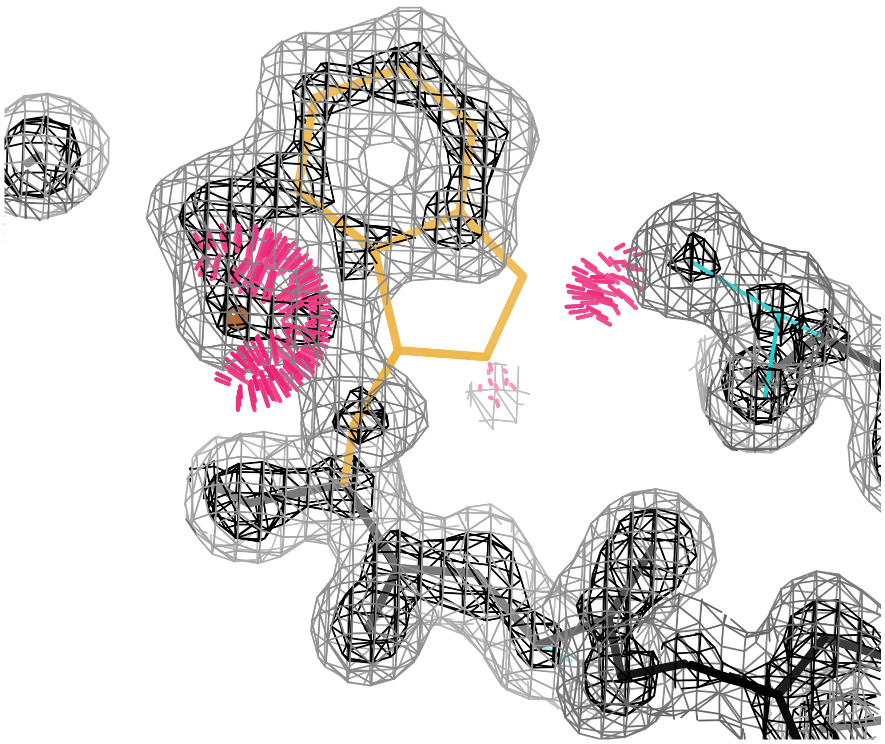


Figure 3: A water (reddish ball) trying to fill the unoccupied electron density created by fitting a Trp sidechain backwards. The water has very large clashes with four atoms of the model, which is a rotamer outlier (flagged in gold), and the incorrect atoms of the Trp also clash, with nearby sidechains. Trp B 170 in 1qw9 at 1.2Å (Hoevel 2003).

Automated or manual water picking will then often place a water in that difference peak. Refinement cannot by itself recover from this type of mistake, but an informed look at severely clashing "waters" can diagnose a correction

Arginine

Arginine, with four χ angles, is prone to an approximately but not correctly placed guanidinium group. Figure 1 shows Arg 59 in 1qLw at 1.09Å, with a water modeled into the peak that actually represents the Nh2 atom. Of course, the water has huge clashes with Ce, Nh1 and Nh2, and the misfit guanidinium produces large negative difference density on the modeled atoms. Arg 32 in 1bkr has a similar problem.

Isoleucine

For isoleucine, it is the Cd1 atom that is displaced by a water, and it usually moves into a different rotamer with Cd entirely out of density and clashing with other residues. The 3js8 cholesterol oxidase structure at 1.54Å has four such cases (Ile 195, 443, 459, and 463), each with a clash overlap >1Å between the water and the Hg12 atom. Figure 2 shows the Ile 195 example, with the large water clash, plus difference density and additional clashes for the displaced Cd1.

Tryptophan

This problem can occasionally occur even for a tryptophan fit backward and non-rotameric, where a water is placed in the density for the Cd1 atom of the sidechain's 5-membered ring. Figure 3 shows Trp 170 in chain B of the 1qLw arabinfuranosidase structure, with the rotamer-outlier sidechain (in gold) obviously backward, with only its 6-membered ring in density. The water has huge clashes with 4 atoms of the model, but it does manage to fill some of the otherwise-unoccupied density. Trp 170 is fit correctly in chain A of 1qw9, but we have twice seen this same startling pattern of a backward Trp in undeposited initial models.

This same structure also has examples of water displacing an atom in leucine and in methionine. Leu A 243 has the water in place of the Cd1 branch, pushing the sidechain aside enough to create a Cβdeviation outlier. In Met A 377 the water occupies the sulfur density of what should be the major alternate conformer. Evidently the modeling of this structure involved early and aggressive water placement.

The bottom line

A water fit in the density of a protein atom causes especially dire consequence for the residue it displaces and clashes with. This happens infrequently, and mostly at about 2Å or higher resolution, but is important and rather easy to avoid. Use the Phenix GUI or the MolProbity multi-chart to search for bad clashes between protein atoms and modeled HOHs, and look at them, where the cases described here are blindingly obvious in Coot or kinemage graphics. Also consult Headd 2013 on other types of water problems to watch for. We are currently working on a tool that will make that process even easier by identifying water clashes and putting them into probable categories to guide their fixup.

References:

Bourne PC, Isupov MN, Littlechild JA (2000) The atomic resolution structure of a novel bacterial esterase, *Struture* **8**: 143-151 [1qLw]

Headd J, Richardson J (2013) "Fitting Tip #5: What's with water?", *Comp Cryst Newsletter* **4**: 2-5

Hoevel K, Shallom D, Niefind K, Belakhov V, Shoham G, Bassov T, Shoham Y, Schomberg D (2003) Crystal structure of a family 51 alpha-L-arabinofuranosidase in complex with 4-nitrophenyl-Ara, *Embo J* **22**: 4922-4932 [1qw9]

Sagermann M, Ohtaki A, Newton K, Doukyu N (2010) Structural characterization of the organic solvent-stable cholesterol oxidase from Chromobacterium sp. DS-1, *J Struct Biol* **170**: 32-40

FAQ

Can I submit my X-ray model to the Protein Data Bank in PDB format?

The answer is no. The PDB has moved away from PDB format in favour of the mmCIF format. Read more at Adams, P. D., Afonine, P. V., Baskaran, K., Berman, H. M., Berrisford, J., Bricogne, G., Brown, D. G., Burley, S. K., Chen, M., Feng, Z., Flensburg, C., Gutmanas, A., Hoch, J. C., Ikegawa, Y., Kengaku, Y., Krissinel, E., Kurisu, G., Liang, Y., Liebschner, D., Mak, L., Markley, J. L., Moriarty, N. W., Murshudov, G. N., Noble, M., Peisach, E., Persikova, I., Poon, B. K., Sobolev, O. V., Ulrich, E. L., Velankar, S., Vonrhein, C., Westbrook, J., Wojdyr, M., Yokochi, M. & Young, J. Y. (2019). *Acta Crystallogr. Sect. Struct. Biol.* **75**, 451–454.

Automatic β-peptide linking in Phenix

Nigel W. Moriarty

Introduction

As a general rule, amino acids polymerise using α–peptide linkages. This is the case for the standard biological amino acids – their amino groups are bonded to the Cα[1] atom (see figure 1A). For β-peptides, the amino group is bound to the Cβ atom (see figure 1B). A concise Wikipedia entry ("Beta-Peptide" 2018), discusses the details including that β-alanine (shown in figure 1B) is the only naturally occurring of the β-peptides.

On 10 June 2019, the Protein Data Bank (Burley et al. 2019) had 40 entries that contain β-alanine (3-letter code BAL) as a polymer and 13 entries as a free ligand. To automatically refine these entries, restraints for the entity are required as well as linking parameters. For the latter, links between the amino acids should ideally contain bonds, angles, dihedrals and planes as needed. For α–peptide linkages, there is one bond, four angles, three dihedral angles and two planes. A single link object can be used on each α–peptide bond with modifications for *cis* conformations. Proline (3-letter code PRO[1]) requires a different set of *cis* and *trans* links that, essentially, replacing the H hydrogen atom with the Cδ carbon atom and adjusting the values appropriately. This proline-specific link is applied to the peptide bond between the proline and the preceding amino acid.

β-peptides require two link records – one for linking to the preceding amino acid and another to link to the following peptide. Modifying the standard peptide links to accommodate the changes was done to produce the skeleton of the links. To obtain suitable values for the bond lengths and angles, a simple LBFGS-B minimization using the SciPy library (Jones, Oliphant, Peterson, et al. 2001) was performed using the highest resolution structure – 4Z0W. The 1.1Å structure contains two chains with four instances of BAL in each. The rmsZ of the link parameters was used as the target.

The resulting values are shown in table 1 and have been added to the GeoStd (Moriarty and Adams, n.d.) shipped with Phenix version 1.16. The mechanism for apply peptide links will use the appropriate link in each situation.

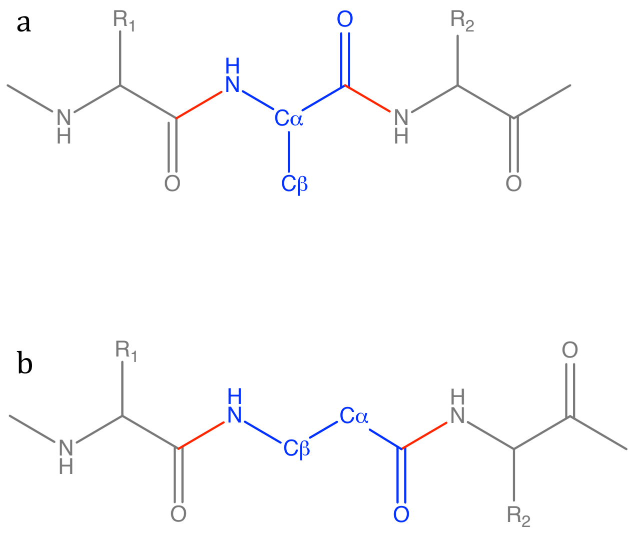


Figure 1: (a) Diagram of α-peptides. That is, the amino groups are bonded to the Cα carbon atom. The middle amino acid (blue) is linked via the red bonds to the preceding "C" carbon atom and the successive "N" nitrogen atom. (b) β-alanine (blue) polymerised with two α-peptides via similar links as in (a).

[1] Greek letters are not subscripted to aid readability and clarity.
[2] Human readable codes (Moriarty, 2016, *CCN*, 26-27) are the norm for this publication but the context makes it clear that the code for proline – PRO – does not contain a zero.

Table 1: Ideal bond lengths and bond angles for pre- and post-β-peptide links.

Pre - β	
Bond (Å)	
C–N	1.335
Angles (°)	
O–C–N	122.7
Cα–C–N	115.7
C–N–Cβ	122.7

Post - β	
Bond (Å)	
C–N	1.346
Angles (°)	
O–C–N	121.3
Cα–C–N	115.9
C–N–Cα	120.8

References

"Beta-Peptide." 2018. Wikipedia. September 6, 2018. https://en.wikipedia.org/w/index.php?title=Beta-peptide&oldid=858336632.

Burley, Stephen K., Helen M. Berman, Charmi Bhikadiya, Chunxiao Bi, Li Chen, Luigi Di Costanzo, Cole Christie, et al. 2019. "Protein Data Bank: The Single Global Archive for 3D Macromolecular Structure Data." *Nucleic Acids Research* 47 (D1): D520–28. https://doi.org/10.1093/nar/gky949.

Jones, E., T. Oliphant, P Peterson, and others. 2001. *SciPy: Open Source Scientific Tools for Python*. www.scipy.org.

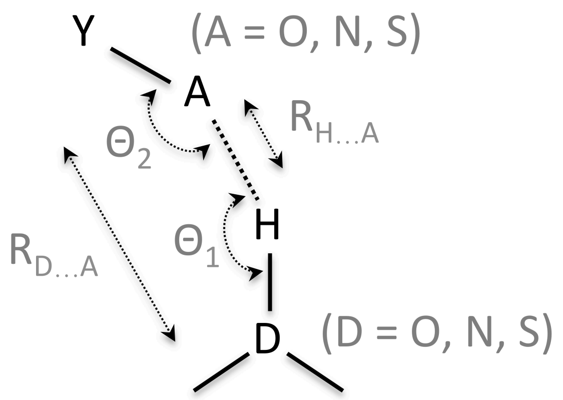
Moriarty, Nigel W., and Paul D. Adams. n.d. *GeoStd*. http://sourceforge.net/projects/geostd.

phenix.hbond: a new tool for annotation hydrogen bonds

Pavel V. Afonine

Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Hydrogen bonds (H-bonds) are non-covalent integrations that are of paramount importance to form and stabilize protein and nucleic acid structure. Secondary structure elements such as helices, sheets and interacting base pairs are held together by H-bonds. In the context of structure solution, the information about H-bonds can be used for validation and refinement. Validation typically focuses on the geometry of H-bonds, such as donor-acceptor distance and angles, as well as overall count of H bonds per structure that is expected to match prior knowledge derived from high-resolution models. Ordered solvent molecules are often validated based upon having plausible hydrogen bond interactions with the macromolecule or/and other solvent molecules. In refinement, restrains on H-bond parameters (length and angles) are particularly important at low resolution when the experimental data isn't sufficient to maintain correct secondary structure geometry (Headd *et al*., 2012). This sets the scene to introduce a new *Phenix* tool called *phenix.hbond* that is designed to annotate hydrogen bonds in atomic models. There are a number of conventions and rules that are used to identify H-bonds, for example see Torshin *et al.* (2002) and Steiner (2002). *phenix.hbond* uses geometric parameters shown in figure 1. Running *phenix.hbond* requires atomic model in PDB or mmCIF format with all hydrogen atoms added, as well as ligand restraint files if the model contains unknown to the library items. Optionally, thresholds for H-bond parameters (figure 1) can be provided that will overwrite the defaults. The program generates two output files. One is a PyMol script that can be used to visualize H-bonds as dashed lines connecting corresponding atoms that form hydrogen bond. The other file defines H-bond restraints as *restraints edits* (*Phenix* parameter file) that are suitable to use in *Phenix* refinement. Output to the log includes a list of all H bonds found that match criteria in figure 1, as well as various statistics such as histograms of H-bond lengths and angles.



$$1.4Å \le R_{H...A} \le 3.0Å \qquad \Theta_1 \ge 120°$$

$$2.5Å \le R_{D...A} \le 4.1Å \qquad 90° \le \Theta_2 \le 180°$$

Figure 1. Hydrogen bond geometry definition used in *phenix.hbond*. $R_{D...A}$ distance is not used with default settings, but can be enabled if needed.

While there is no particular reason why this should not work for all bio-macromolecules, currently *phenix.hbond* is only optimized and tested to work with proteins, which is the limitation that will be removed in future.

Literature

Ivan Y. Torshin, Irene T. Weber, Robert W. Harrison. Protein Engineering, Design and Selection, Volume 15, Issue 5, May 2002, Pages 359–363.

Steiner, T. (2002). Angew. Chem. Int. Ed. 41. 48-76.

Headd JJ, Echols N, Afonine PV, Grosse-Kunstleve RW, Chen VB, Moriarty NW, Richardson DC, Richardson JS, Adams PD. Acta Cryst. D68, 381-390 (2012).

Bytes and Bobs : Accelerating python code with Numba.

Petrus H. Zwart[a,b]

[a] Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

[b] Center for Advanced Mathematics in Energy Research Applications, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

Correspondence email: PHZwart@lbl.gov

This is a lightweight introduction to something I encountered and found useful and interesting. Although the material presented here might be standard knowledge for some of you, it certainly wasn't for me. I provide these insights here in the hope that could be of use to some. The article below is by no means complete, exhaustive or unbiased.

Introduction

Coding in python is great, but one of the major downsides is that is can be rather slow, especially when iterating over large arrays. Have a look at the following example where we compute the sum of a large number of values in an array using a simple python for-loop (panel 1):

```
#Panel 1
import numpy as np
import time

def tst_python(x):
    result = 0
    for xx in x:
        result += xx
    return result

N=int(1e6)
x = np.random.random(int(N))
t0 = time.time()
rp = tst_python(x)
t1 = time.time()
time_python = t1-t0

print time_python, 'seconds'

>
0.206596851349 seconds
```

```
#Panel 2
from scitbx.array_family import
flex
import numpy as np
import time

def tst_flex(x):
    return flex.sum(x)

N=int(1e6)
x = np.random.random( int(N) )
x_as_flex = flex.double( x )
t0 = time.time()
rp = tst_flex(x_as_flex)
t1 = time.time()
time_flex = t1-t0

print time_flex, 'seconds'

>
0.000848054885864 seconds
```

Although a 0.20 second seems decent enough, the cctbx build-in methods available from the scitbx.array_family speed this up dramatically (panel 2).

We see that there is a speedup of a factor of 250 over the plain python code. As the reader might recall, this is accomplished in the following way:

1. Writing a dedicated C++ function that performs the numerical operation, a summation in this case.
2. Writing a C++ wrapper using the boost–python tools that exposes this functionality to python.
3. Recompiling a portion of the CCTBX library.

Although these steps are by no means hard, they can be daunting and cumbersome, especially when you haven't done this for a while. The boost-python mechanism has been the driving force in providing algorithms at acceptable speeds within the CCTBX and PHENIX software frameworks [1], and in the hand of seasoned CCTBX and PHENIX developers, is a marvelous tool to provide code at the highest performance levels.

Numba

The main drawback of boost-python however, especially for the casual, frustrated, or time-constrained developer [2] is the need to dive back into C++ to get stuff done. An alternative approach is however available: the *numba* toolkit. Numba is a just-in-time compiler that translates "python functions into optimized machine code at runtime using the industry standard LLVM compiler" [3].

Although I am sure that the computer science behind the LLVM compiler and its python interface is fascinating (see for instance [4,5]), it is more productive to focus on how to use it and what to expect.

The use of numba is relatively straightforward. By adding a specific numba decorator (*@numba.jit*) to a function, an optimized function is compiled at runtime that is almost just as fast as compiled C++ code. An illustrative example is provided in panel 3.

As you can see, the first function call upon execution is about a factor of 4 slower as compared to native python code. The second identical function call within the same python script (with a fresh set of random numbers) runs in 0.0015 seconds. This is only a factor 2 slower than the optimized C++ code, and is similar to a `numpy.sum()` function call (data not shown).

The bulk of the time upon first execution is spend in the compilation of the numba-decorated python code. At the second function call, this compilation is no longer needed resulting in a very nice performance.

Note that the keyword 'cache=True' can reduce some of the compilation time required at the first function call: only 0.1 seconds was need for the initial compilation when the script was re-executed. Note that the compilation time is independent of the argument provided to the function: if the first function call to `tst_numba` is executed on an array with length 3, the timings are the same (data not shown).

The runtimes are summarized in Figure 1 for all 5 cases.

```
#Panel 3
import numba
import numpy as np
import time

@numba.jit(nopython=True,cache=True)
def tst_numba(x):
    result = 0
    for xx in x:
        result += xx
    return result

N=1e6
x = np.random.random(int(N))

t0 = time.time()
rn = tst_numba(x)
t1 = time.time()
time_numba = t1-t0
print time_numba, 'seconds'

x = np.random.random(int(N))
t0 = time.time()
rn = tst_numba(x)
t1 = time.time()
time_numba = t1-t0
print time_numba, 'seconds'

> First execution
0.822153091431 seconds
0.00154995918274 seconds

> Second execution
0.108898162842 seconds
0.00156705284119 seconds
```

Outlook

Numba supports numpy data types, which makes it very easy to use. Debugging numba functions is relatively straightforward, the documentation and examples on the numba website are fairly instructive. A possible drawback of numba is that it not yet supports all object-oriented features of python, forcing one to write a separate, dedicated numba functions for numerical task that can be called from within a class.

As the toy examples indicate, a C++ implementation seems to get the fastest code possible, albeit at the cost of having to deal with boost python. The use of numba allows one to code in native python, using numpy objects, but without a potential boost-python struggle. If the numba function coded up is called repeatedly, such as a target function and its derivatives in a minimizer, initial compilation costs are small price to pay to strike a balance between run-time efficiency and developer time.

Besides the illustrated accelerations on basic python code, numba also features GPU support for CUDA systems and for AMD ROC GPUS [3].

Numba can be installed with pip (including the required compiler) thus:

```
cctbx.python -m pip install numba
```

or via conda.

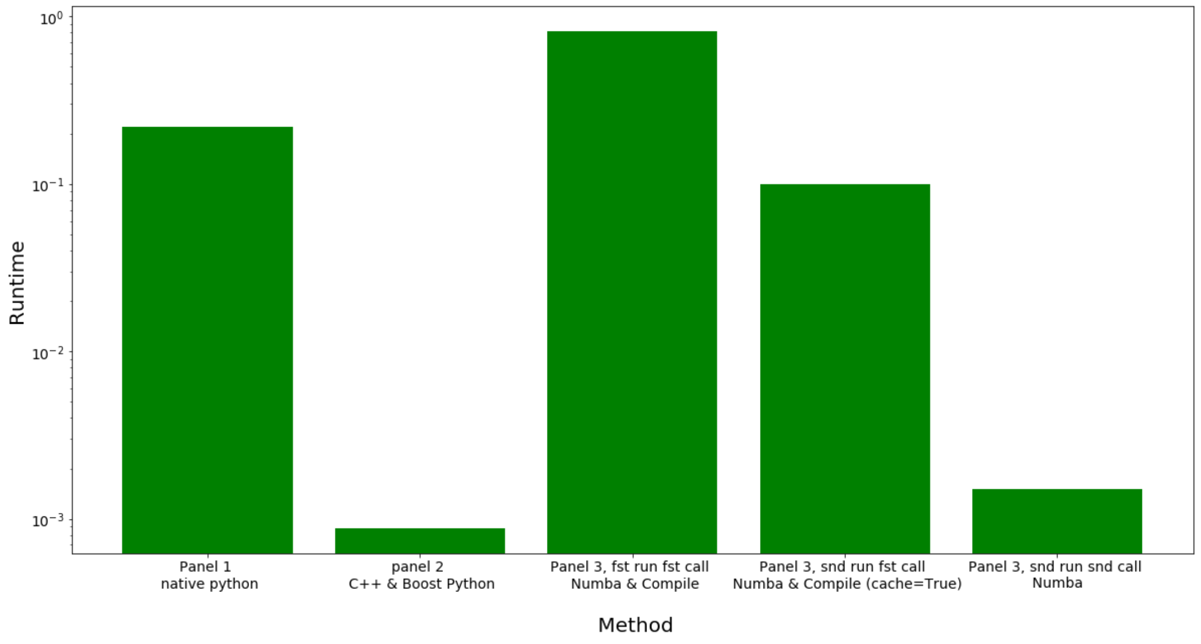


Figure 1: Runtimes for the summation of 1 million random numbers. The vertical axis is on a logarithmic scale.

References

1. https://www.boost.org/doc/libs/1_70_0/libs/python/doc/html/article.html
2. http://biosciences.lbl.gov/profiles/peter-zwart/
3. https://numba.pydata.org/index.html
4. https://llvm.org/
5. https://en.wikipedia.org/wiki/LLVM

Processing serial crystallographic data from XFELs or synchrotrons using the *cctbx.xfel* GUI

Aaron S. Brewster[a], Iris D. Young[a,b], Artem Lyubimov[c], Asmit Bhowmick[a], and Nicholas K. Sauter[a]

[a]*Molecular Biophysics and Integrated Bioimaging Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720*
[b]*Department of Bioengineering and Therapeutic Sciences, University of California, San Francisco, CA 94158*
[c]*Stanford Synchrotron Radiation Laboratory, SLAC National Accelerator Laboratory, Menlo Park, CA 94025*

Correspondence email: asbrewster@lbl.gov, nksauter@lbl.gov

Introduction

The most important question when conducting a serial crystallographic (SX) experiment is "Have I collected enough data?" SX experiments at X-ray free electron lasers (XFELs) can produce millions of images with hundreds of thousands of diffraction patterns. Each pattern needs to be indexed to determine the crystal orientation matrix and unit cell dimensions, and then integrated to produce intensities for the observed Miller reflections. These images are 'stills', meaning they are collected without rotating the crystal in the beam due to short pulse length, usually 10s of femtoseconds long (reviewed in Bergmann *et al.* (2017)). Similarly, serial crystallography experiments at synchrotrons can produce hundreds of thousands images, also all stills, using fixed-target mounting systems on chips or loops and then raster-scanning through the crystals without rotating the sample in the beam (reviewed in Sierra *et al.* (2018)). The amount of data produced by either approach can make it difficult to determine whether sufficient data have been collected.

While estimates of completeness, multiplicity of measurements, signal vs. noise, cross-correlation statistics, and unit cell isomorphism can all give useful insights as to dataset quality, knowing when a dataset answers a particular scientific question can only come from examining the electron density maps. The specifics generally depend on the type of experiment being performed, but they usually rely on examining difference density in the maps, often comparing two time points, two mixing conditions or other such treatments against each other. Time at an XFEL is scarce, therefore feedback as to sample quality and data completeness as determined from electron density maps needs to be available as fast as possible.

Challenges to rapid data processing include ensuring accurate detector calibration, aggregating and visualizing hit rates, crystal quality, automated processing job submission, monitoring the available computing systems, keeping samples and associated metadata organized, and rapidly merging data to create electron density maps. Particularly at XFELs where experimental teams can involve 20+ scientists, results need to be communicated effectively to all parties, including sample preparation teams, beamline scientists, sample injection specialists and data analysts. Finally, processing needs to be automated to allow scientists to spend time studying the data itself rather than focusing on the mechanics of submitting many jobs and monitoring their state.

To solve these challenges we have developed the *cctbx.xfel* GUI (graphical user interface). This program, under active development, allows users to rapidly move through all phases of serial crystallographic data reduction in an organized matter, taking advantage of whatever local computing resources are available. The GUI is open source and part of the *cctbx* and DIALS software packages (Grosse-Kunstleve *et al.*, 2002, Hattne *et al.*, 2014, Winter *et al.*, 2018).

In this article we detail two tutorials for processing XFEL data using the *cctbx.xfel* GUI, including refining the geometry. In the first tutorial we use data from Nakane *et al.* (2016a). This is an iodine derivatized bacteriorhodopsin protein sample (HAD13a) collected on the octal-sensor MPCCD detector at SACLA. This dataset is useful for a tutorial since the initial geometry is good enough to index out of the box and it doesn't require further software libraries. In the second tutorial we process a thermolysin dataset collected on the CSPAD detector at LCLS, following the approaches shown in Brewster *et al.* (2018).

We also show here how the GUI could be used to process data from synchrotrons, where the output files are typically stored in directories filled with single image files. The tutorials here provide a demo designed to run on a small Linux node outside of the facilities of interest, but they can be adapted easily to run at full scale either at these facilities or other computing environments.

SX data processing workflows

We re-state here what we wrote in our previous newsletter article (Brewster *et al.*, 2016), updated to be more general across SX experiments outside of LCLS. It has been our experience that analyzing data collected using serial crystallography (SX) typically requires three distinct processing stages labeled here **calibration**, **discovery**, and **batch.** **Calibration** refers to refining the geometry of the experiment, but also includes some pre-processing steps, such as creating bad pixel masks. Using these inputs, initial parameters are derived that describe the experiment, such as detector distance, any beam correction parameters needed and so forth. During **discovery**, the user examines individual diffraction patterns and searches for appropriate parameters for data reduction, including hitfinding parameters if used, spotfinding parameters, target unit cell dimensions, crystal symmetry and an optimal merging strategy. Finally, when optimal software configuration is established, the user enters **batch** processing mode, endeavoring to maximize the parallel computing options offered and, during live experiments, attempting to provide constructive feedback to beam line operators in as close to real-time as possible. After the experiment, the user will often need to reprocess the runs collected in batch mode. During batch processing, the user will continue to refine processing parameters as the results are evaluated, perhaps even revising initial experimental geometry estimates. Thus the three stages are somewhat fluid as feedback from later stages may call for repeating earlier stages.

Processing at scale using MySQL

Aggregating feedback from SX experiments has often been done by searching through log files or result files and creating plots and tables. As experiments get large, with thousands to millions of images being processed across many datasets, the scale of the data complicates this kind of data scraping. To solve this problem, we used a database system implemented in MySQL to store and retrieve information about every frame processed. The system allows us to use structured queries to quickly sort, aggregate and visualize crystal indexing results and integration quality.

At LCLS, the facility staff has provided a MySQL server for general users. Access is trivial to obtain by emailing the staff. For other facilities we have provided a program, *cctbx.xfel.ui_server* that wraps MySQL and initializes the database from scratch. Users at any facility can run this program, as described below, either locally or on a computer cluster. The *cctbx.xfel* GUI will connect to this server and use it to track jobs, processing parameters and sample quality for rapid feedback. Because MySQL is designed as an enterprise solution for managing large amounts of data, as experiments expand in scope, the backend for managing the large amounts of metadata will scale as well.

Data processing tutorials

The two tutorials presented here describe how to process datasets from SACLA and LCLS. We first explain how to install and configure the *cctbx.xfel* software and how to acquire the tutorial data, then, after initial calibration, we demonstrate how to use the GUI itself to submit and monitor jobs, and visualize the processing results. Merging is described, but this section is under active development and is likely to change after this article is published.

Installation

The *cctbx.xfel* GUI comes with DIALS and Phenix installations and will run natively after installing MySQL. What follows are directions for a standalone (non-LCLS) installation. After that are directions for an LCLS installation that includes *psana*, the package needed to read the LCLS file format (XTC) natively. Both procedures assume the user is on a single node system, without access to the original facility's computers, though queuing support using multiple nodes for large batch processing is also described. These directions are verified to work on Linux Centos 7.

Let $WORKING be a new, empty directory. Note, here $WORKING always refers to the full path to that directory. Also let $NPROC be the number of processors available on your system, for example, 32.

Standalone builds

Download the installation script:

```
wget https://raw.githubusercontent.com/cctbx/cctbx_project/master/xfel/util/standalone_xfelgui_installer.sh
```

Run the script, providing the destination folder:

```
chmod +x standalone_xfelgui_installer.sh
./standalone_xfelgui_installer.sh $WORKING
```

The script will download the latest version of DIALS, install it in the $WORKING folder, install MySQL, and create a setup.sh script that you can use to put DIALS in your path:

```
source $WORKING/setup.sh
```

LCLS builds

The *cctbx.xfel* GUI is available for all users at LCLS at /reg/g/cctbx. However, for the purposes of this tutorial, we assume no access to the LCLS computing systems. To use the *psana* libraries, we need to build the software manually; we cannot use a pre-built DIALS bundle. The installation script does this.

Download the script:

```
wget https://raw.githubusercontent.com/cctbx/cctbx_project/master/xfel/util/lcls_xfelgui_installer.sh
```

Run the script, providing the destination folder and the number of processors:

```
chmod +x lcls_xfelgui_installer.sh
./lcls_xfelgui_installer.sh $WORKING $NPROC
```

The script will download the latest version of DIALS, *psana*, MySQL, and other dependencies, build the software, and create a setup.sh script that you can use to put DIALS in your path:

```
source $WORKING/setup.sh
```

For more information on developer builds of DIALS, see
https://dials.github.io/documentation/installation_developer.html

Download and prepare tutorial data

The *cctbx.xfel* GUI requires a run to have finished being collected before processing begins. Therefore it has multiple modes for monitoring for new data, depending on the facility. LCLS has a webservice that can be used to query if data is available. SACLA uses Cheetah to prepare HDF5 files from the raw data, and indicates it is finished using a status file (Barty *et al.*, 2014, Nakane *et al.*, 2016b). Standalone facilities such as synchrotrons often collect a constant number of files in a single raster. If all else fails, time stamps and file sizes can be monitored.

We download the data from cxi.db (Maia, 2012):

SACLA tutorial data

- cd $WORKING; mkdir -p data/run1; cd data/run1
- Get a run from HAD13a from here: https://www.cxidb.org/data/43/HAD13a/. Here we use the first run which had the fewest hits and is the smallest file size: wget http://portal.nersc.gov/archive/home/projects/cxidb/www/43/HAD13a/run371999-0.h5
- Create a Cheetah status.txt file so the run will be seen by the GUI: echo Status=Finished > status.txt

LCLS tutorial data

These directions assume the user is not working on the LCLS interactive nodes.

- cd $WORKING; mkdir lcls; cd lcls

- LCLS uses a .dat file that maps experiment names to experiment number. Create a .dat file with only the thermolysin experiment in it:
    - mkdir ExpNameDb
    - echo "280 CXI cxi78513" > ExpNameDb/experiment-db.dat
- Download a run from cxi.db, entry 81 (see https://www.cxidb.org/id-81.html):
    - mkdir -p CXI/cxi78513/xtc; cd CXI/cxi78513/xtc
    - wget http://portal.nersc.gov/archive/home/projects/cxidb/www/81/cxi78513/xtc/e280-r0013-s00-c00.xtc
    - Note, this is 81.8 GB! And it's only 1/5th of run 13!
- Build the small data file needed to read the xtc file:
    - mkdir smalldata
    - smldata -f e280-r0013-s00-c00.xtc -o smalldata/e280-r0013-s00-c00.smd.xtc
- Download the calibration folder. This includes the CSPAD geometry file refined in Brewster 2018 (see also https://github.com/phyy-nx/dials_refinement_brewster2018) and the CSPAD dark pedestal files.
    - cd $WORKING/lcls/CXI/cxi78513
    - wget http://portal.nersc.gov/archive/home/projects/cxidb/www/81/cxi78513/calib.tar.gz
    - tar -xvf calib.tar.gz; rm calib.tar.gz
- Export variables instructing *psana* where the data are (note you can add these lines to your setup.sh script from the installation section after the source commands).

```
export SIT_DATA=$WORKING/lcls
export SIT_ROOT=$SIT_DATA
export SIT_PSDM_DATA=$SIT_DATA
```

If the user is running at LCLS on their own data, all of these steps can be skipped as *psana* has defaults that can find the data in /reg/d/psdm, the data's default location.

Start MySQL

The MySQL server is wrapped by the program *cctbx.xfel.ui_server*. This program takes as an argument the directory in which the database will be initialized, which directory must be empty the first time the program is ran. The first time the program is ran, a root password will be requested which will be used for the root database account that the program will create and set up. This should not be your system root password. Subsequently, the program can be run on a cluster or locally, as needed.

- cd $WORKING
- cctbx.xfel.ui_server db.port=3307 db.server.basedir=$WORKING/MySQL db.user=guidemo2019 db.name=guidemo2019
    - Note, the db.user and db.name fields create a MySQL user and a MySQL database within the MySQL/data folder. A password can also be provided but it would be stored as unencrypted text in the GUI settings file so this is not recommended. Here we leave the password blank.
- Provide a root password and wait until "Raising max connections" appears
- Background the process (CTRL-Z, then type bg)
- Note, when done processing, shut the server down using fg followed by CTRL-C

This step can be skipped when running at LCLS itself, provided that facility staff has granted access to psdb-user.slac.stanford.edu.

Initial calibration and masking

In the case of HAD13a, the initial geometry is sufficient for indexing. If it were not, the initial detector position could be determined using a powder pattern from a known sample, such as silver behenate

(AgBeh). To do this, using the averaging commands below on the AgBeh dataset, run *dials.image_viewer* and use the Actions: show unit cell tool to determine the new beam center and detector distance by fitting the overlaid rings.

Given a good initial geometry, follow these steps to create an untrusted pixel mask using an average image. This is optional because for the HAD13a dataset, the beamstop is a simple circle in the middle of the image and it can be masked out using a low-resolution filter during processing. However, you can use *dials.image_viewer* tool to create a custom mask if needed.

- cd $WORKING
- mkdir averages; cd averages
- dxtbx.image_average ../data/run1/run371999-0.h5 -v -n $NPROC
- dials.image_viewer *.cbf
- Page to avg.cbf if it's not already displayed
- Actions: show mask tool
- The goal is to make a low resolution mask around the beamstop. If this were a single panel image, the circle tool would work, but because it is multipanel, each inner tile needs its own mask. Use the polygon tool four times. When done, click save mask. A pixels.mask file will be created.
- Test the mask: dials.image_viewer *.cbf mask=pixels.mask. Click the show mask button in the settings dialog. The masked pixels will turn red.

For the LCLS thermolysin data, the detector has already been calibrated (see Brewster *et al.* (2018)). Follow the instructions at https://github.com/phyy-nx/dials_refinement_brewster2018/wiki/Averages-and-masking to generate an untrusted pixel mask.

Run and configure the *cctbx.xfel* GUI

To start the *cctbx.xfel* GUI, on the command line, run:

- cd $WORKING
- cctbx.xfel

At this point several settings dialogs will be used to configure the processing environment. In these examples, local processing is used (single node), but alternatives are discussed below including multi-node clusters.

Standalone GUI (HAD13a example)

Use the following settings for the HAD13a dataset:

- Login window:
    - Experiment Tag: common. The experiment tag is a way to group processing results together. We tend to use 'common' to indicate a set of processing results that are available for all contributors to an experiment, but any string of characters can be used.
    - Facility: Standalone
    - Output: $WORKING/results
- DB Credentials window:
    - DB Host name: 127.0.0.1
    - DB Port number: 3307
    - DB name: guidemo2019
    - DB user name: guidemo2019
- Facility options window:
    - Folder to monitor: $WORKING/data
    - Monitor for: folders

    - File matching template: run*.h5
    - Check the 'Files are composite' box
- Advanced settings:
    - Multiprocessing: local
    - Total number of processors: $NPROC
    - Processing back end: cctbx.xfel (standalone mode). Note there are other options here, including the possibility of running other programs. Contact the authors if interested.
- Note, the DB host name will vary depending on what system the MySQL database is running on. For this tutorial, running on a local node, we use the IP address of 'localhost'.
- Tip: the GUI saves your settings to ~/.cctbx.xfel/settings.phil, which will be automatically used the next time you run the GUI.

Note, if running the GUI at SACLA during an experiment, set the multiprocessing system to PBS, then use the blX-occupancy queue (where X is the beamline number) and specify the appropriate number of cores per node (currently 28 cores) and the total number of cores desired. If that number is >28, multiple nodes will be used per job.

Multiple users can run the GUI at the same time using the same experiment tag and database. They will see the same set of processing results as the single MySQL backend is queried. However, it is advised to only monitor for new runs and submit jobs from a single GUI instance.

<u>LCLS GUI (thermolysin example)</u>

- Login window:
    - Experiment Tag: common.
    - Facility: LCLS
    - Experiment: cxi78513
    - Output: $WORKING/results
- DB Credentials window:
    - DB Host name: 127.0.0.1
    - DB Port number: 3307
    - DB name: guidemo2019
    - DB user name: guidemo2019
- Facility options window:
    - LCLS user name: <Leave blank or get from staff>
    - LCLS password: <Leave blank or get from staff>
    - These credentials are for the LCLS web service for monitoring for new runs. The service does not use the same credentials as the facility's Unix accounts. If the credentials are not provided, the GUI instead looks for runs in the xtc folder for the experiment.
- Advanced settings:
    - Multiprocessing: local
    - Total number of processors: $NPROC
    - Processing back end: cctbx.xfel (LCLS mode).

Note these instructions are for running outside of LCLS. If using the LCLS provided MySQL server, specify the DB host name as psdb-user.slac.stanford.edu, leave the port blank (it will default to 3306), and specify the DB name and DB user name provided to you by the facility staff. Additionally, for multiprocessing select LSF and then select either the psanaq (offline processing) or a high priority queue if processing during an experiment.

GUI tab: Runs

The GUI is organized into a series of tabs. The first tab, runs, shows the runs discovered by the GUI. Click the 'Watch for new runs' button and after a moment the available runs will appear. A run represents a continuous time of data collection where all the parameters will the same. At a synchrotron, this will often represent a single raster scan. The GUI will continue to monitor for new data every few seconds. You can disable the monitoring by clicking the 'Watch for new runs' button again.

We found it difficult to keep track of which run numbers correspond to which sample and the sample conditions. We typically log metadata in a spreadsheet during the beamtime, but we wanted to be able to sort runs by this metadata within the GUi. To this end, runs can be tagged with descriptive terms, such as 'thermolysin', 'batch 5', 'timepoint 3', and so forth. Click 'Manage tags' to create, rename, and remove tags. Click a run to tag it, or click 'Change tags on multiple runs' to work with many tags at once. During data collection, use the 'Manage persistent tags' feature to automatically tag new runs with a tag set as they arrive. Use these tags to group runs together using words appropriate to your experiment. You can use these tags in the subsequent plots to quickly switch between which data is being examined.

GUI tab: Trials

The list of parameters available to the core processing program *dials.stills_process* is extensive, but most of the time only a few defaults need to be changed. We have found that during an experiment the same data needs to be re-processed several times as it calibrated and explored, changing geometry, spotfinding and indexing parameters. We group processing attempts that change parameters that generally refer to the crystal together into trials. For example, trial 0 may be our initial indexing trial based on the published unit cell, but after examining the results, we see the unit cell is slightly different, so we resubmit the jobs into a new trial 1 with a better unit cell estimate. The output folders are organized by runs and trials to keep the data organized.

Further, we have often found that sequential groups of runs tend to have the same set of detector-specific parameters, such as geometry. Therefore we create run groups (or run blocks) that identify sets of runs that should be similarly behaved. Thus, a trial has sample and crystal-specific parameters, and a trial comprises a set of run groups. Note that while tags and run groups both organize runs into logical groupings, they are used differently. Tags use user-defined metadata specific to the sample and experiment while run groups organize runs into sets with similar processing parameters. Use the Trials tab to manage these features:

Had13 Trial 0

- Go to trials tab and click new trial. Settings:
  - Comment: as you like, for example 'HAD13a, initial indexing'
  - Change resolution in bottom right to 1.7. This resolution is only used in aggregating the results, not in indexing or integrating the data.
  - In the central window, use these parameters:

    ```
    spotfinder {
     filter {
     d_max = 19
     min_spot_size = 2
     }
    }
    indexing {
     known_symmetry {
     space_group = C2221
     unit_cell = 46.2, 103.0, 128.7, 90, 90, 90
     }
    }
    prediction.d_max=19
    ```

  - Note prediction.d_max and spotfinder.filter.d_max can be removed if you created a static low resolution mask above.

  - Choosing good spotfinding parameters is critical. In this case the gain is close to 1 so spotfinder.threshold.dispersion.gain is not modified (unlike for the CSPAD, see below). The DIALS tutorials online have further guidance for choosing these parameters using *dials.image_viewer* (see https://dials.github.io/documentation/tutorials).
  - If the unit cell and symmetry are unknown, the indexing parameters can be omitted in which case the P1 will be used. Clustering tools such as those in Zeldin *et al.* (2015) and Gildea and Winter (2018) can be used to determine the unit cell and symmetry.
- Click ok.
- Click new block. If you made a mask, put the path to it in the "Untrusted pixel mask" box (should be $WORKING/averages/pixels.mask). Click ok.
- Check the Active Trial box

Additional run groups can be created as needed, and can be added or removed from a trial using the 'Select blocks' button. Note, by default, a set of processing results will be created for each image. To save on file system usage, these can be combined using output.composite_output=True in the trial parameters.

Thermolysin Trial 0

For the CSPAD thermolysin data, use the above procedure with these trial parameters:

```
spotfinder {
 filter.min_spot_size=2
 threshold.dispersion.gain=25
 threshold.dispersion.global_threshold=100
}
indexing {
 known_symmetry {
 space_group = P6122
 unit_cell = 92.9 92.9 130.4 90 90 120
 }
 refinement_protocol.d_min_start=1.7
}
```

Here, a global threshold of 100 is used to remove noisy pixels. Note, the LCLS version by default writes CBF versions of each indexed image to assist debugging the raw XTC streams. To save on file system usage, this can be disabled with dispatch.dump_indexed=False. Also, for LCLS processing, composite_output=True is the default.

For the run group, use CxiDs1.0:Cspad.0 for the detector address, 580.119 for DetZ, and provide the path to the untrusted pixel mask if you created one.

GUI tab: Jobs

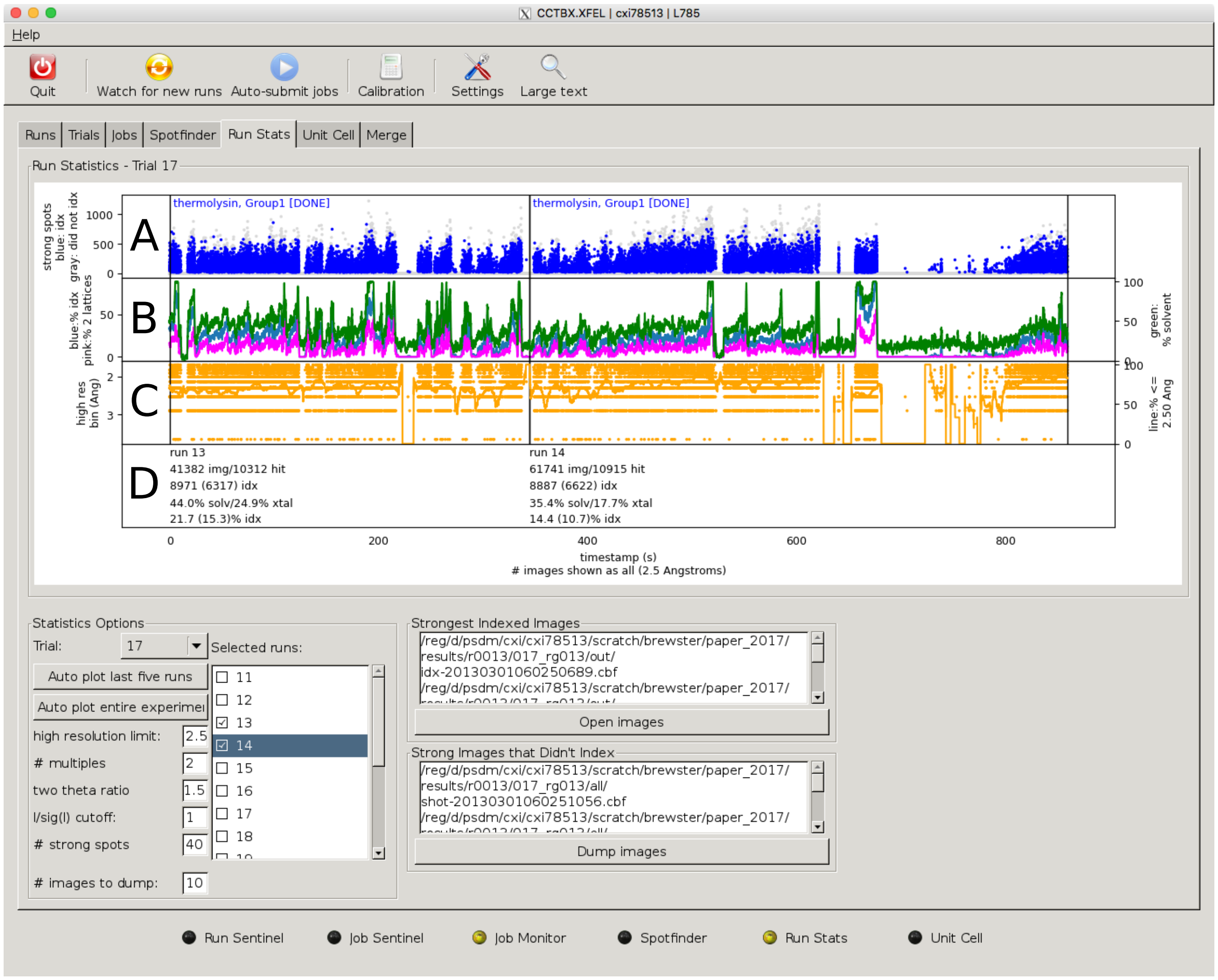
The jobs tab displays processing job information. A job consists of the output from a single processing attempt, and is associated with a trial number, a run group number, and a run. Click the 'Auto-submit jobs` button. For local processing mode, each job not yet processed is ran in sequence. For LSF, PBS, or other queuing systems, all jobs not yet processed are submitted to the queue. As new data arrives, they are automatically submitted. Under the hood, the program *cxi.mpi_submit* is used to submit the jobs (see Brewster *et al.* (2016)).

Processing results are logged to the MySQL database, and are created as DIALS reflection tables and experiment files in the output folder $WORKING/results, as configured above. The full set of commands and processing parameters used for the job are copied to this folder, in job-specific subfolders, for archival purposes.

Jobs can be terminated, deleted, and restarted using the jobs tab. Deleting a job deletes all the results from the MySQL database and from the file system in the results folder.

GUI tab: Run Stats

The Run Stats tab displays a variety of hit rate



Run Statistics - Trial 17

run 13
41382 img/10312 hit
8971 (6317) idx
44.0% solv/24.9% xtal
21.7 (15.3)% idx

run 14
61741 img/10915 hit
8887 (6622) idx
35.4% solv/17.7% xtal
14.4 (10.7)% idx

images shown as all (2.5 Angstroms)

Statistics Options
Trial: 17
Selected runs: ☐ 11 ☐ 12 ☑ 13 ☑ 14 ☐ 15 ☐ 16 ☐ 17 ☐ 18
Auto plot last five runs
Auto plot entire experimen
high resolution limit: 2.5
# multiples: 2
two theta ratio: 1.5
I/sig(I) cutoff: 1
# strong spots: 40
# images to dump: 10

Strongest Indexed Images
/reg/d/psdm/cxi/cxi78513/scratch/brewster/paper_2017/results/r0013/017_rg013/out/idx-20130301060250689.cbf
/reg/d/psdm/cxi/cxi78513/scratch/brewster/paper_2017/

Strong Images that Didn't Index
/reg/d/psdm/cxi/cxi78513/scratch/brewster/paper_2017/results/r0013/017_rg013/all/shot-20130301060251056.cbf
/reg/d/psdm/cxi/cxi78513/scratch/brewster/paper_2017/

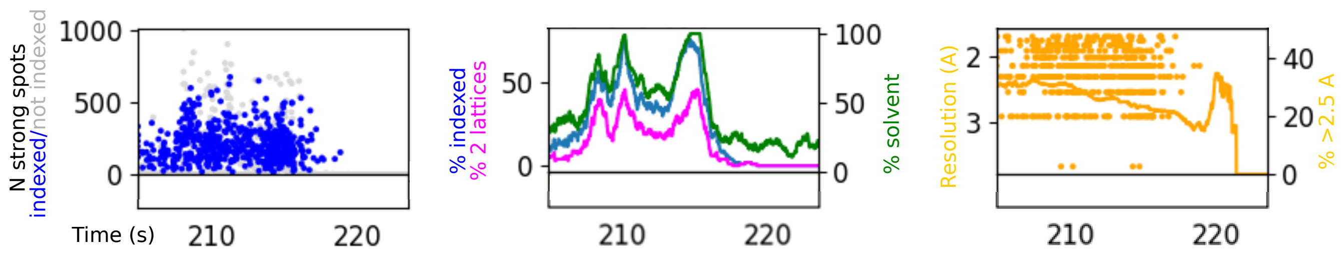


Figure 1: Run Stats tab showing two thermolysin runs. Note, the tutorial uses only 1/5th of run 13, whereas here we show all of run 13 plus run 14. Top: XFEL GUI. Bottom row: zooms of three sections of the main plot.

information. Select a trial and a run, and the plot will be generated. See figure 1 for a thermolysin example.

Every part of the hit rate plot is a direct result of issues encountered during SX experiments. We need to know a variety of information at a glance, such as whether we are hitting the sample with the beam, are there crystals, are there multiple lattices, can we index them (if not, why), and what

information. Select a trial and a run, and the plot will be generated. See figure 1 for a thermolysin example.

Every part of the hit rate plot is a direct result of issues encountered during SX experiments. We need to know a variety of information at a glance, such as whether we are hitting the sample with the beam, are there crystals, are there multiple lattices, can we index them (if not, why), and what

is their quality. Further, we need the information updated live as the experiment progressed, but we also need to be able to quickly compare runs to each other, even if they were collected days earlier. To this end, the database is queried and hit rates displayed as you select specific runs, auto-plot the last five runs, or plot the entire experiment (all runs at once).

The top panel is split into four horizontal plots, labeled in figure 1 as A-D. In the hit rate plot (A), every image has one dot. The height of the dot is how many reflections were found during spotfinding on the image. The dot is blue if it indexed and gray if it did not.

In the indexing plot (B), moving averages are computed for the solvent hit rate (green, right-hand Y-axis), the indexing rate (blue, left-hand Y-Axis) and the multiple lattice rate (magenta, left-hand Y-Axis). These rates are the percentage of the total number of shots in the moving window that contained solvent, indexed successfully, or had multiple lattices present, respectively.

The solvent rate is computed by examining ratio of the water ring intensity to the background. This is done by specifying two 2θ values in the run group, a high value and a low value. The defaults are 22.8° for the water ring and 12.5° for the background ring (note these are wavelength dependent following Bragg's law). Radial average values are computed at these two positions and if their ratio is higher than the default of 1.0, then the shot is considered having solvent.[1]

The diffraction quality plot (C) shows one dot per image in yellow, where the height of the dot (left Y-axis) is the resolution the image diffracted to, using a mean I/σ ratio cutoff of 1.0 by default. The yellow line is a moving average of the percentage of indexed images that diffract to at least 2.5Å (right Y-axis). This is the high-quality rate. Note that the resolution estimates are usually optimistic. After scaling and post-refinement, many reflections have reduced intensity due to partiality correction, since most reflections are partial. Therefore the user may wish to adjust the I/σ ratio cutoff to something more stringent to get a more realistic resolution estimate.

The statistics bar (D) shows per run statistics. In the case of thermolysin run 13, 10312 shots out of 41382 images were considered hits, where a hit has at least 40 reflections. 8971 images indexed, 6317 of which were high quality. 44.0% of shots had solvent, and 24.9% of shots had crystals (this is the hitrate). 21.7% of shots indexed, and 15.3% of shots indexed to 2.5Å. The high quality rate (not shown) would be (6317/8971) = 70.4% for this run.

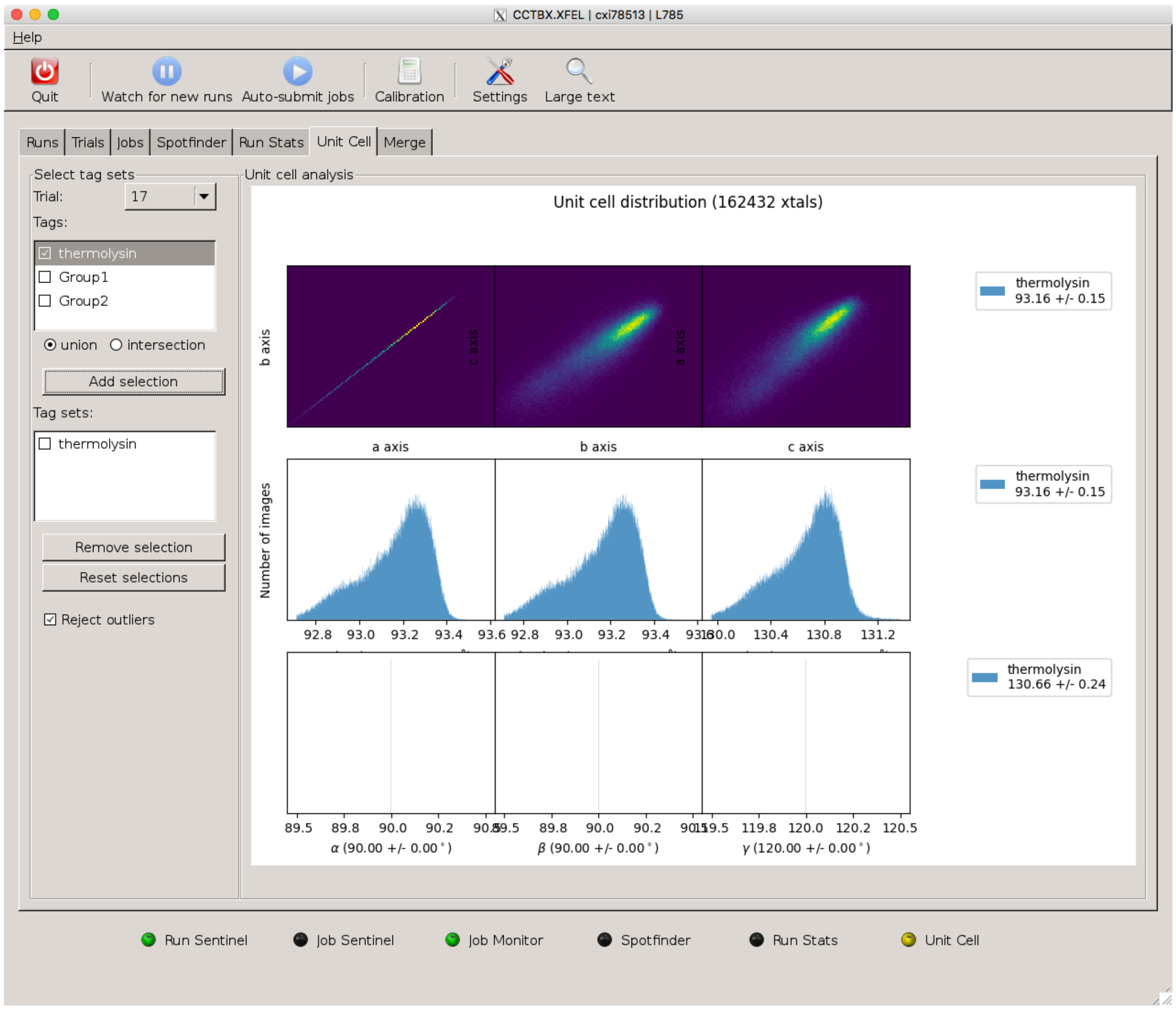
Most of the parameters listed above, such as I/σ ratio cutoff, are configurable in the lower-left hand corner of the Run Stats tab. The two image display options, 'Strongest indexed images' and 'Strongest images that didn't index' open the DIALS image viewer and allow users to look at their best data and their most problematic data, respectively. (Note this feature is only available for the LCLS facility, but is in development for extant facilities).

Some tips on usage. A flat line in the hitrates bar (A) indicates none of the shots contain Bragg diffraction. This could mean there is no data but it could also mean the spot finding thresholds are too stringent. A mix of blue and gray dots indicates many images are not indexing. The unit cell parameters or geometry might not be well optimized. Many gray dots consistently higher than the blue dots could indicate multiple lattices, as indexing tends to fall off if too many crystals are hit in a shot.

Combining A and B, if the solvent rate is zero and the hitrate is zero, then there is no solvent in the beam, indicating the jet is missing or the raster is missing, etc. If the solvent rate is high but the number of spots is low, the crystals concentration is too low. If the solvent rate and number of spots are both high but the indexing rate is low, the indexing parameters or experimental geometry is wrong, or the spotfinding parameters are picking up noisy reflections.

The spotfinder should find few reflections on images considered misses, but poor spotfinding parameters can result in noisy mis-identified reflections, especially near a beam stop or other shadowing. If the background number of spots per image is too high (around 20-40), consider first

[1]The background ring default 2θ is configured for kapton tape scatter as typically seen in SX experiments used with drop on tape methods (see Fuller 2017), and as such the default ratio of 1.0 is too low for injection or raster experiments without a kapton signal. In this case the ratio should be set to a higher values, such as 1.5 in this example, where the thermolysin was



Trial: 17

Tags:
- ☑ thermolysin
- ☐ Group1
- ☐ Group2

(● union) ○ intersection

Tag sets:
- ☐ thermolysin

☑ Reject outliers

thermolysin 93.16 +/- 0.15

thermolysin 93.16 +/- 0.15

thermolysin 130.66 +/- 0.24

α (90.00 +/- 0.00 °)

β (90.00 +/- 0.00 °)

γ (120.00 +/- 0.00 °)

Figure 2: Unit cell plot for thermolysin. Note, this is for all the thermolysin data from (Brewster *et al.*, 2018).

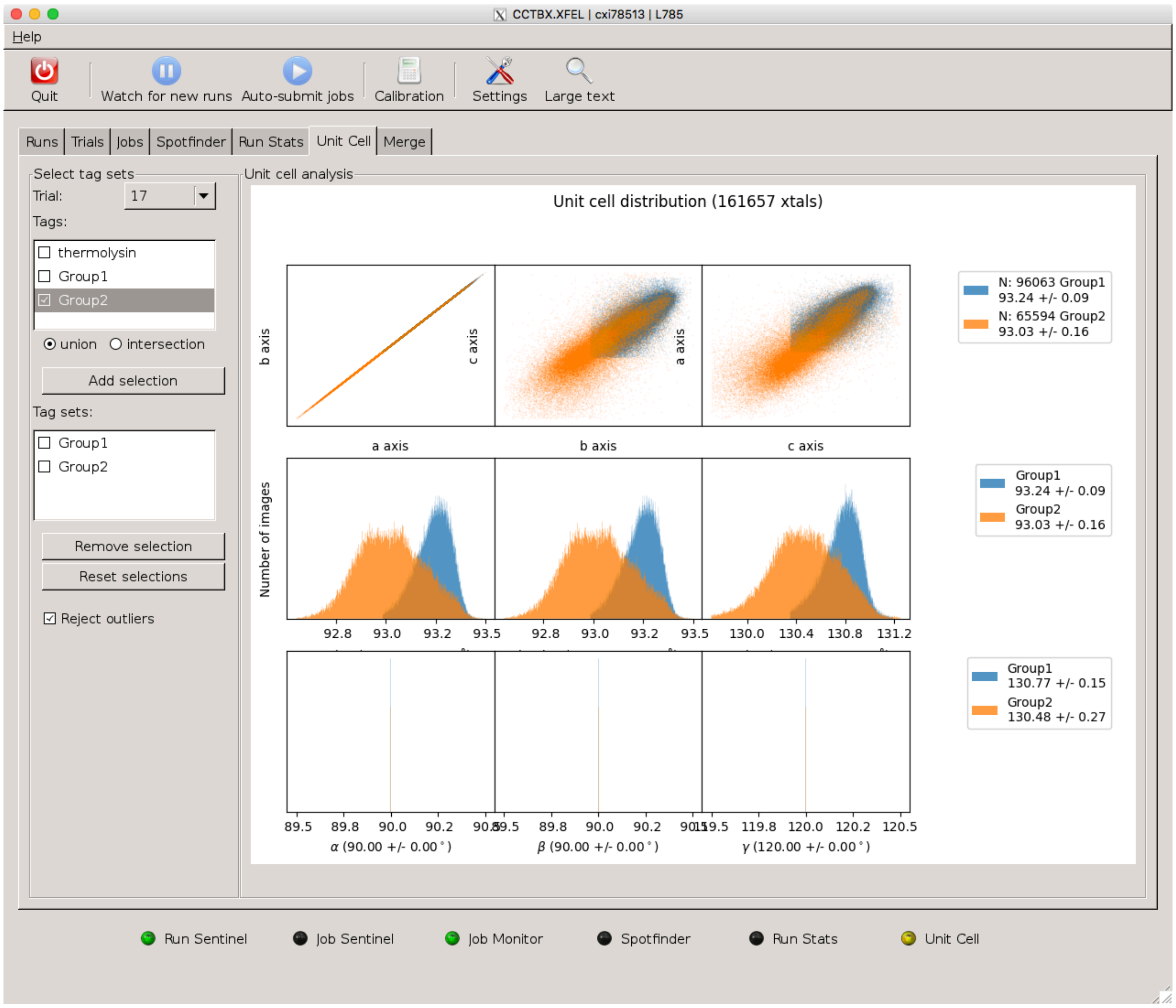
adjusting the spotfinding parameters and if that isn't working, setting dispatch.hit_finder.minimum_number_of_reflections to a number slightly higher than the background of your spotfinding results. This parameter will skip indexing obvious misses, saving processing time.

GUI tab: Unit Cell

The Unit Cell tab uses tag sets to display unit cell distributions in 2D (figure 2 top) and 1D (figure 2 middle and bottom). Runs must already be tagged to be displayed in this plot. Select a trial, then select a tag or tags and click 'Add selection'. The tag set will be shown in the Tag sets window and can be removed from the plot using the 'Remove selection' button.

A tag set is one or more collections of runs all tagged with tags in the set. The set can be a union or an intersection, meaning display any images from runs tagged with any of these tags (union), or display only images from runs with all these tags (intersection).

For example, in figure 3, Runs 11-22 are tagged as 'Group1', and runs 26-29 are tagged as 'Group2'. In this case it can be seen that groups 1 and 2 do not overlap even though they are both measurements of the same protein. In Brewster 2018, the geometry for each run is refined separately, which



Trial: 17

☐ thermolysin
☐ Group1
☑ Group2

(●) union ( ) intersection

☐ Group1
☐ Group2

☑ Reject outliers

🟦 N: 96063 Group1 93.24 +/- 0.09
🟧 N: 65594 Group2 93.03 +/- 0.16

🟦 Group1 93.24 +/- 0.09
🟧 Group2 93.03 +/- 0.16

🟦 Group1 130.77 +/- 0.15
🟧 Group2 130.48 +/- 0.27

Figure 3: Unit cell plot for two groups of runs for thermolysin. Note, this is for all the thermolysin data from (Brewster *et al.*, 2018).

helps to correct this pattern by slightly adjusting the detector distance in a time-dependent refinement, causing these populations to better overlap (not shown).

Geometry refinement and re-indexing

These instructions, shown here for the HAD13a data but generally applicable, are similar to those given here: http://cci.lbl.gov/xfel/index.php/2017_dials.stills_process. For the CSPAD data, follow the directions at https://github.com/phyy-nx/dials_refinement_brewster2018/wiki, under the metrology refinement and metrology evaluation sections (see also Brewster *et al.* (2016) and Brewster *et al.* (2018)).

Recall that the DIALS file formats are twofold: a set of experimental models linked together in an experiment list and a list of reflections in a reflection table. An experiment represents a single diffraction result from a single crystal, and for still images includes a detector model (with position and orientation for each panel), a beam model (including beam direction and wavelength), and a crystal model (comprising the unit cell and crystal orientation as well as mosaic parameters. In an experiment list file, each experiment can refer to the same detector, so in the first step of geometry refinement, we create a list of experiments and reflections such that each experiment points to the same detector model. We will then jointly refine all the models, such that all reflections will inform the position of the detector while the position of the detector will affect each crystal model (Waterman *et al.*, 2016, Brewster *et al.*, 2018).

Also note that the octal MPCCD detector at SACLA has 8 panels. We group these into hierarchy levels, where level 0 represents the detector as a whole, and level 1 represents each panel individually. A detector can have arbitrarily many hierarchy levels (for example, the CSPAD has 4: the detector, quadrants, 2x1 modules and each ASIC).

- cd $WORKING; mkdir -p metrology/t000; cd metrology/t000
- Combine and filter experiments:
  - dials.combine_experiments ../../results/run1_run371999-0/000_rg001/out/*indexed.refl ../../results/run1_run371999-0/000_rg001/out/*refined.expt reference_from_experiment.detector=0
  - This will create combined.refl (a set of indexed reflections from all indexed images) and combined.expt (a set of crystallographic models including 1 detector model, N beam models and N crystal models, where N is the number of indexed image. In the test for this article, there were 92 images indexed).
  - cctbx.xfel.filter_experiments_by_rmsd combined.*
  - This will remove any images that have a particularly bad RMSD, where RMSD is the root mean squared deviation of the observations from the predictions. In this case, 3 images were removed with this command.
- Refine the detector as a block (level 0):
  - dials.refine refine_level0.phil filtered.* output.experiments=refined_level0.expt output.reflections=refined_level0.refl
  - Where the file refine_level0.phil contains:

```
refinement {
 parameterisation {
 auto_reduction {
 min_nref_per_parameter = 3
 action = fail fix *remove
 }
 beam {
 fix = *all in_spindle_plane out_spindle_plane wavelength
 }
 detector {
 fix_list = Tau
 }
 }
 refinery {
 engine = SimpleLBFGS LBFGScurvs GaussNewton LevMar *SparseLevMar
 }
 reflections {
 outlier {
 algorithm = null auto mcd tukey *sauter_poon
 separate_experiments = False
 separate_panels = True
 }
 }
}
```

  - These parameters adjust the refiner for SX experiments as opposed to the defaults, which are geared towards traditional rotation experiments at synchrotrons. Description of the parameters:
    - auto_reduction: for stills, there are few reflections per shot, and thus few reflections per parameter. We lower the cutoff to 3 of how many reflections per parameter to use compared to the default of 5 and we allow experiments with too few reflections to be removed.

  - We fix the beam so it does not refine. As the beam, the detector distance, and the unit cell dimensions are correlated, one parameter must be used as a ruler, and we assume the beamline facility has correctly calibrated the wavelength.
  - We fix the rotation of the detector around its normal and around its fast and slow axes. The user should feel encouraged to try other refinement strategies including letting the detector tilt refine. For more, see Brewster *et al.* (2018).
  - We use the sparse LevMar refiner which uses sparse algebra to handle the large matrix of parameters needed for refinement. For more, see Brewster 2018.
  - We use the Sauter/Poon approach for outlier rejection (Sauter & Poon, 2010). To determine outliers, for each panel we consider all the reflections on that panel across all experiments.
- In the refinement summary you'll see the RMSD_X, RMSD_Y and RMSD_DeltaPsi will all have decreased.

- Refine the individual panels (level 1):
  - dials.refine refine_level1.phil refined_level0.* output.experiments=refined_level1.expt output.reflections=refined_level1.refl
  - Where the file refine_level1.phil contains (difference from level 0 highlighted in red):

```
refinement {
 parameterisation {
 auto_reduction {
 min_nref_per_parameter = 3
 action = fail fix *remove
 }
 beam {
 fix = *all in_spindle_plane out_spindle_plane wavelength
 }
 detector {
 fix_list = Group1Tau1,Tau2,Tau3
 hierarchy_level = 1
 }
 }
 refinery {
 engine = SimpleLBFGS LBFGScurvs GaussNewton LevMar *SparseLevMar
 }
 reflections {
 outlier {
 algorithm = null auto mcd tukey *sauter_poon
 separate_experiments = False
 separate_panels = True
 }
 }
}
```

  - Note, here we allow the panel to tilt around their normal axes (Tau1), but still fix the other tilt axes. One panel has to be fixed to limit the degrees of freedom, hence Group1Tau1 is still fixed (see Brewster *et al.* (2018)).
  - RMSDs will fall even further after this refinement.
- To visualize refinement results:
  - cctbx.xfel.detector_residuals hierarchy=1 plot_max=0.3 tag=Filtered filtered.* &
  - cctbx.xfel.detector_residuals hierarchy=1 plot_max=0.3 tag=Level0 refined_level0.* &
  - cctbx.xfel.detector_residuals hierarchy=1 plot_max=0.3 tag=Level1 refined_level1.* &

From the text output of *cctbx.xfel.detector_residuals*:

```
Filtered:
RMSD (microns) 116.314413579
Overall radial RMSD (microns) 82.32334385
Overall transverse RMSD (microns) 82.1700058634
```

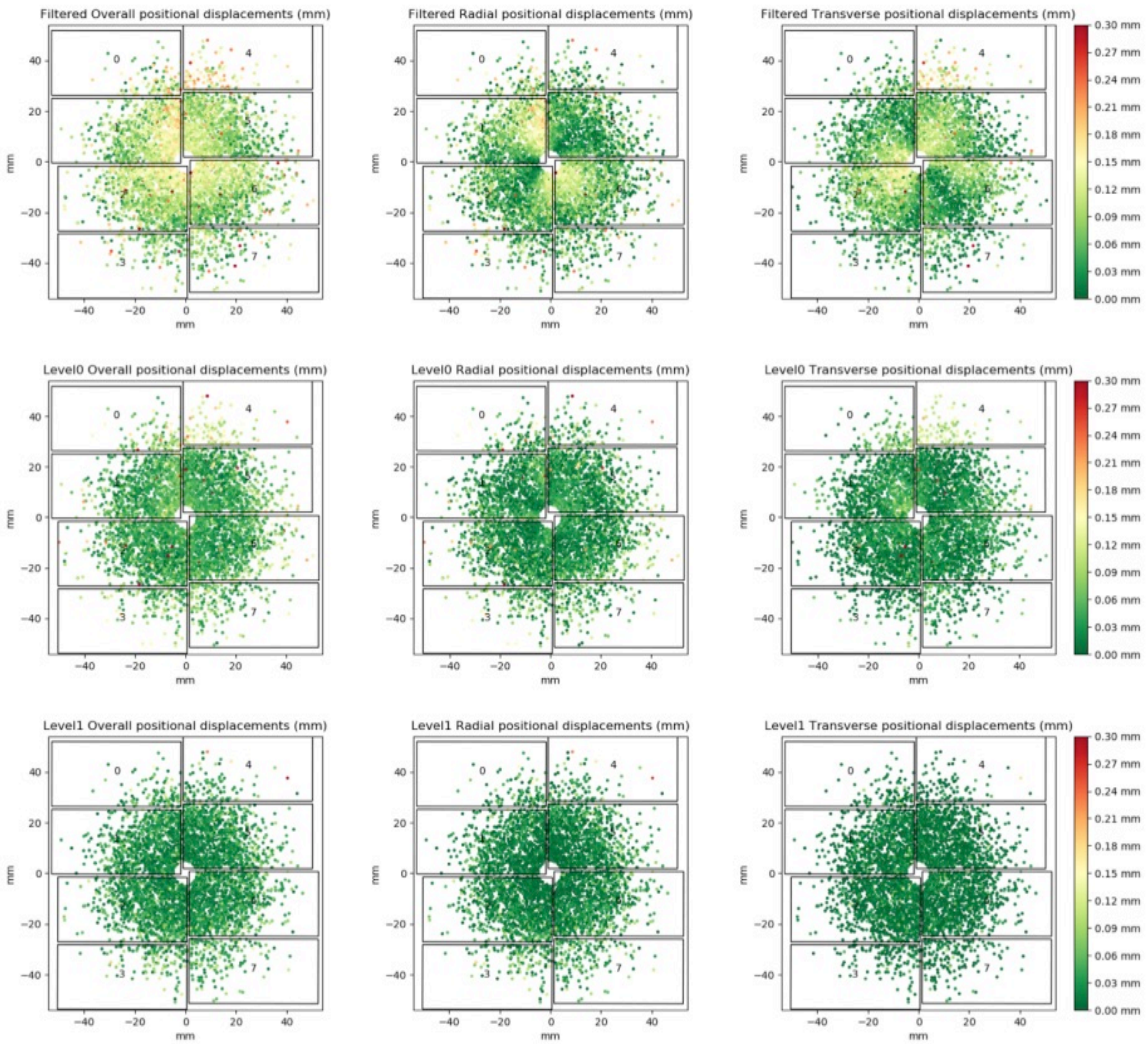


Figure 4: Positional displacement plots for the octal MPCCD detector at SACLA before and after refinement.

```
Refined level 0:
RMSD (microns) 66.0757037266
Overall radial RMSD (microns) 48.6571169644
Overall transverse RMSD (microns) 44.7044023747

Refined level 1:
RMSD (microns) 41.3719691901
Overall radial RMSD (microns) 33.8076040347
Overall transverse RMSD (microns) 23.8471328277
```

Displacement plots from *cctbx.xfel.detector_residuals* for the 3 stages are in figure 4. Top row: filtered, middle row: refined level 0, bottom row: refined level 1. Each dot is a reflection. The color is the displacement between the observed and predicted reflection centroids. Left column: overall displacement. Middle column: radial displacement (along vector from the reflection to the beam center). Right column: transverse displacement (along vector normal to beam vector and radial vector).

How much did the detector move? We can use the command cspad.detector_shifts filtered.expt filtered.refl refined_level1.expt refined_level1.refl to evaluate this (note, the legacy name *cspad.detector_shifts* will be renamed at some point). In the 'Detector shifts summary' table, level 0 moved 116.8 microns in XY and 142.5 in Z. Level 1, where each panel moved individually, moved on average 92.3+/-46.9 microns in XY and -208.6+/-43.5 microns in Z.

From these results we have the following observations:

- The cross-hatching in the unrefined data usually means a bad beam center.
- Comparing RMSDs between the unrefined and refined datasets is not completely fair due to outlier rejection. See Brewster 2018 for a discussion.
- After level 1 refinement, the upper right panel gets much better.
- During level 0 refinement, the detector moves in Z +93.3 microns, but during level 1 refinement, the panels moved in Z back -208.6 microns. This has been observed before (see Brewster *et al.* (2018) for a discussion).
- The refined detector will have slightly different Z values for each panel, which might not be physically realistic. An alternative refinement approach would constrain all the panels to move the same amount during level 1 refinement by adding detector.constraints.parameter=Dist.

To prepare the new metrology for use by indexing:

- mkdir split; cd split
- dials.split_experiments ../refined_level1*
- cd ..
- mv split/split_00.expt .
- rm -rf split/

This takes the file refined_level1.expt and removes all but one of the experiments, creating split_00.ext. W use this instead of the full set from the combined experiments list. We can then reprocess the HAD13a data using the new metrology:

- Back in the GUI tab click on new trial, change the resolution to 1.7, change the comment to 'HAD13a, refined metrology', and click OK
- Click new block, in the extra phil block add input.reference_geometry=$WORKING/metrology/t000/split_00.expt
- Click Active trial and the new job will be submitted
- Using the new metrology increased the number of images indexed from 93 to 114

GUI tab: Merging

As stated in the Introduction, the most important question to answer during an SX experiment is "Have I collected enough data?" Therefore, merging data rapidly to create datasets for scientists to examine is vital. In this tab, tag sets can be selected to list the output folders for each run in the tag set. This can be copied into merging scripts for merging independently from the GUI. Currently, merging directly from the GUI isn't supported, but functionality for this will be implemented by Oct 2019. For details on merging the thermolysin data, see https://github.com/phyy-nx/dials_refinement_brewster2018/wiki/Merging .

Integrating the GUI into new facilities

The *cctbx.xfel* GUI can be run standalone using SX data in most environments as currently implemented. For example, at a synchrotron, the user may want to monitor a folder where new rastering runs will appear as new directories filled with images. Simply configure the GUI to monitor this folder in the GUI setup screens. Facility-specific features, including areas where additions can be made trivially are listed here:

- DIALS natively supports raw data from most facilities. This includes XTC (LCLS), PAL HDF5, SACLA Cheetah HDF5, NeXus (Eiger, EuXFEL and SwissFEL), and any single-image format, such as CBF and SMV, currently supported by the *dxtbx* data modeling library that is part of *cctbx* (Parkhurst *et al.*, 2014). Adding support for new beamlines is straightforward.
- New data monitors. At LCLS this is done by querying a webservice and at SACLA this is done by monitoring for a Cheetah status.txt

file. Other standalone methods include looking for a certain number of files or watching for time stamps and/or file sizes to be unchanging. Implementing new monitors specific to a facility is straightforward and can be done with a few lines of Python.

- Support for clustering systems. In addition to local mode, where simple Python multiprocessing is done on the current node, LSF and PBS queuing systems are available through the GUI. *cxi.mpi_submit*, the program used by the GUI to submit jobs, additionally supports SLURM and custom queuing systems, such as the NESRC shifter technology, which will be made available in the GUI in the near feature.

Please contact the authors if you need additional support in any of these areas.

Conclusions and outlook

The *cctbx.xfel* GUI has been successfully used live during experiments at LCLS, SACLA and PAL, and will be made available at DLS, EuXFEL, and SwissFEL in the coming year. The GUI can and has been used remotely through X-windows forwarding or virtual systems such as NoMachine or VNC.

It may be that facilities wish to integrate components and features of the *cctbx.xfel* GUI into their pipelines without using the GUI itself, for example the MySQL database logging and the Run Stats graphics. These programs are all based on Python object-oriented design and have documented APIs that can be merged into existing frameworks. Again, please contact the authors if there is interest.

Finally, as XFELs and synchrotrons move SX into the kilohertz and megahertz regimes, data processing solutions designed from the ground up to scale with the experimental size will be vital. The MySQL and multiprocessing approaches detailed here are designed exactly with the scaling problems in mind. We hope to work with beamline scientists and facilities to incorporate these methods into their systems, ensuring the fast-feedback and monitoring needed during precious SX beamtime to enable answering that most important question as fast as possible, “Have I collected enough data to answer my scientific questions?”

Acknowledgements

This research was supported by NIH grant GM117126 for data-processing methods. Portions of this research were carried out at LCLS at the SLAC National Accelerator Laboratory, supported by the US DOE, Office of Science, OBES under Contract No. DE-AC02-76SF00515. Data processing was performed in part at the National Energy Research Scientific Computing Center, supported by the DOE Office of Science, Contract No. DE-AC02-05CH11231.

References

Barty, A., Kirian, R.A., Maia, F.R.N.C., Hantke, M., Yoon, C.H., White, T.A. & Chapman, H. (2014). **Cheetah: software for high-throughput reduction and analysis of serial femtosecond X-ray diffraction data**. *J. Appl. Crystallogr.* **47**, 1118-1131.

Bergmann, U., Yachandra, V. & Yano, J. (2017). *X-Ray Free Electron Lasers*. London: The Royal Society of Chemistry.

Brewster, A.S., Waterman, D.G., Parkhurst, J.M., Gildea, R.J., Michels-Clark, T., Young, I.D., Bernstein, H.J., Winter, G., Evans, G. & Sauter, N.K. (2016). **Processing XFEL data with cctbx.xfel and DIALS**. *Computational Crystallography Newsletter* **7**, 32-53.

Brewster, A.S., Waterman, D.G., Parkhurst, J.M., Gildea, R.J., Young, I.D., O'Riordan, L.J., Yano, J., Winter, G., Evans, G. & Sauter, N.K. (2018). **Improving signal strength in serial crystallography with DIALS geometry refinement**. *Acta Crystallographica Section D Structural Biology* **74**, 877-894.

Gildea, R.J. & Winter, G. (2018). **Determination of Patterson group symmetry from sparse multi-crystal data sets in the presence of an indexing ambiguity**. *Acta Crystallographica Section D Structural Biology* **74**, 405-410.

Grosse-Kunstleve, R.W., Sauter, N.K., Moriarty, N.W. & Adams, P.D. (2002). **The *Computational Crystallography Toolbox*: crystallographic algorithms in a reusable software framework**. *J. Appl. Crystallogr.* **35**, 126-136.

Hattne, J., Echols, N., Tran, R., Kern, J., Gildea, R.J., Brewster, A.S., Alonso-Mori, R., Glockner, C., Hellmich, J., Laksmono, H., Sierra, R.G., Lassalle-Kaiser, B., Lampe, A., Han, G., Gul, S., DiFiore, D., Milathianaki, D., Fry, A.R., Miahnahri, A., White, W.E., Schafer, D.W., Seibert, M.M., Koglin, J.E., Sokaras, D., Weng, T.C., Sellberg, J., Latimer, M.J., Glatzel, P., Zwart, P.H., Grosse-Kunstleve, R.W., Bogan, M.J., Messerschmidt, M., Williams, G.J., Boutet, S., Messinger, J., Zouni, A., Yano, J., Bergmann, U., Yachandra, V.K., Adams, P.D. & Sauter, N.K. (2014). **Accurate macromolecular structures using minimal measurements from X-ray free-electron lasers**. *Nat. Methods* **11**, 545-548.

Maia, F.R.N.C. (2012). **The Coherent X-ray Imaging Data Bank**. *Nat. Methods* **9**, 854-855.

Nakane, T., Hanashima, S., Suzuki, M., Saiki, H., Hayashi, T., Kakinouchi, K., Sugiyama, S., Kawatake, S., Matsuoka, S., Matsumori, N., Nango, E., Kobayashi, J., Shimamura, T., Kimura, K., Mori, C., Kunishima, N., Sugahara, M., Takakyu, Y., Inoue, S., Masuda, T., Hosaka, T., Tono, K., Joti, Y., Kameshima, T., Hatsui, T., Yabashi, M., Inoue, T., Nureki, O., Iwata, S., Murata, M. & Mizohata, E. (2016a). **Membrane protein structure determination by SAD, SIR, or SIRAS phasing in serial femtosecond crystallography using an iododetergent**. *Proceedings of the National Academy of Sciences* **113**, 13039-13044.

Nakane, T., Joti, Y., Tono, K., Yabashi, M., Nango, E., Iwata, S., Ishitani, R. & Nureki, O. (2016b). **Data processing pipeline for serial femtosecond crystallography at SACLA**. *J. Appl. Crystallogr.* **49**, 1035-1041.

Parkhurst, J.M., Brewster, A.S., Fuentes-Montero, L., Waterman, D.G., Hattne, J., Ashton, A.W., Echols, N., Evans, G., Sauter, N.K. & Winter, G. (2014). **dxtbx: the diffraction experiment toolbox**. *J. Appl. Crystallogr.* **47**, 1459-1465.

Sauter, N.K. & Poon, B.K. (2010). **Autoindexing with outlier rejection and identification of superimposed lattices**. *J. Appl. Crystallogr.* **43**, 611-616.

Sierra, R.G., Weierstall, U., Oberthuer, D., Sugahara, M., Nango, E., Iwata, S. & Meents, A. (2018). *X-ray Free Electron Lasers*, pp. 109-184.

Waterman, D.G., Winter, G., Gildea, R.J., Parkhurst, J.M., Brewster, A.S., Sauter, N.K. & Evans, G. (2016). **Diffraction-geometry refinement in the DIALS framework**. *Acta crystallographica. Section D, Structural biology* **72**, 558-575.

Winter, G., Waterman, D.G., Parkhurst, J.M., Brewster, A.S., Gildea, R.J., Gerstel, M., Fuentes-Montero, L., Vollmar, M., Michels-Clark, T., Young, I.D., Sauter, N.K. & Evans, G. (2018). **DIALS: implementation and evaluation of a new integration package**. *Acta crystallographica. Section D, Structural biology* **74**, 85-97.

Zeldin, O.B., Brewster, A.S., Hattne, J., Uervirojnangkoorn, M., Lyubimov, A.Y., Zhou, Q., Zhao, M., Weis, W.I., Sauter, N.K. & Brunger, A.T. (2015). **Data Exploration Toolkit for serial diffraction experiments**. *Acta Crystallogr. D Biol. Crystallogr.* **71**, 352-356.