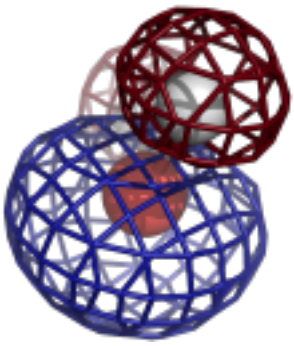
VOLUME TWELVE



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JANUARY MMXXI

PDB is 50, CSD Lessons II, HKLviewer

Table of Contents

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

Phenix News

Announcements

New Phenix Release

Highlights for the 1.19.2 version of *Phenix* released in Februray 2021 include:

- ✓ *phenix.real_space_refine*
  - Improved rotamer fitting – multiprocessing, NCS constraints work on one copy only and propagate changes to all related copies
  - Improved map to restraints weight calculation
  - Morphing can now use multiprocessing
- ✓ New methods
  - *phenix.local_resolution*: calculates a local resolution map
  - *phenix.local_aniso_sharpen*: optimizes a map taking into account local resolution-dependence and anisotropy of the map and its errors
- ✓ High-level scriptable Python tools are now available for map & model analyses, manipulation and model-building
- ✓ Restraints
  - Adjusting the "positions" of atom names is (pseudo-)symmetric amino acid side chains is now the default
  - Improved restraints for Arginine allows more flexibility of the Cγ atom
- ✓ Amber
  - Automatic creation of Amber files in *phenix.refine* GUI
  - Added AmberPrep GUI

- ✓ Restraints
  - Restraints added for FeS
  - Metal coordination library is default to Zn+2 and FeS clusters
- ✓ Rama-Z: New global Ramachandran score for identifying protein structures with unlikely stereochemistry in Command Line Interface (*phenix.rama_z, mmtbx.rama_z*) and GUI (validation reports).
- ✓ Density modification for cryo-EM
  - Includes model-based density modification with automatic model generation
  - Optimized defaults and additional documentation
- ✓ Real-space refinement
  - Hydrogen atoms no longer included in map target function improving fit
  - Add NQH flip option (enabled by default)
- ✓ Other
  - New tool for ordered solvent picking in cryo-EM maps (*phenix.douse*)
  - New map and model superposition tool (*phenix.match_maps*)
  - New FindProgram tool to find any *Phenix* program with a text search
  - Project details now has a button for opening the README file for tutorials

Please note that this new publication should be used to cite the use of *Phenix*:

Macromolecular structure determination using X-rays, neutrons and electrons: recent developments in *Phenix*. Liebschner D, Afonine PV, Baker ML, Bunkóczi G, Chen VB, Croll TI, Hintze B, Hung LW, Jain S, McCoy AJ, Moriarty NW, Oeffner RD, Poon BK, Prisant MG, Read RJ, Richardson JS, Richardson DC, Sammito MD, Sobolev OV, Stockwell DH, Terwilliger TC, Urzhumtsev AG, Videau LL, Williams CJ, Adams PD: Acta Cryst. (2019). D75, 861-877.

Downloads, documentation and changes are available at phenix-online.org

Expert advice

Fitting Tip #20 – In-plane or out-of-plane H atom placement on the edges of aromatic rings

Jane Richardson and Dave Richardson, Duke University

Most H atom placements in proteins and nucleic acids, and even on ligands, are fairly straightforward matters, either staggered tetrahedral on single-bonded parent atoms or planar on double-bonded parents. But cases where the parent atom is bonded to the edge of an aromatic ring do not neatly fit those paradigms and are somewhat more complex and diverse.

Methyl groups on aromatic ring edges: out-of-plane

Heme groups in proteins and thymine bases in DNA are the most commonly encountered cases of a methyl on the edge of an aromatic ring. Figure 1 shows clear, positive difference peaks for the H atoms of a heme methyl at 0.88Å resolution.

When a 3-fold tetrahedral group lies across a bond to a 2-fold planar group, the most favored dihedrals put no H either in-plane (0°) or staggered (60°). Instead, there are two minimum-energy conformations, each with one H perpendicular to the plane and the other two H's 30° from the plane, as seen in the figure. As true for other methyl moieties, Reduce does not freely rotate these methyl H's. However, instead of a single staggered placement, on ring edges there are two possibilities to decide between, chosen in Reduce by minimizing clashes. In the case illustrated, Reduce clearly picked the correct alternative. If you use another method of placing hydrogen atoms, you should check that on aromatic ring edges it produces an arrangement close to one of the favored 90°-30°-30° alternatives. Longer aliphatic branches off the edge of aromatic rings just follow normal rules.

OH groups on aromatic ring edges: in-plane

Most common OH groups, such as on Ser or Thr, or SH for Cys, favor staggered dihedral angles (+60°, 180°, or -60°). However, the OH on Tyr or other

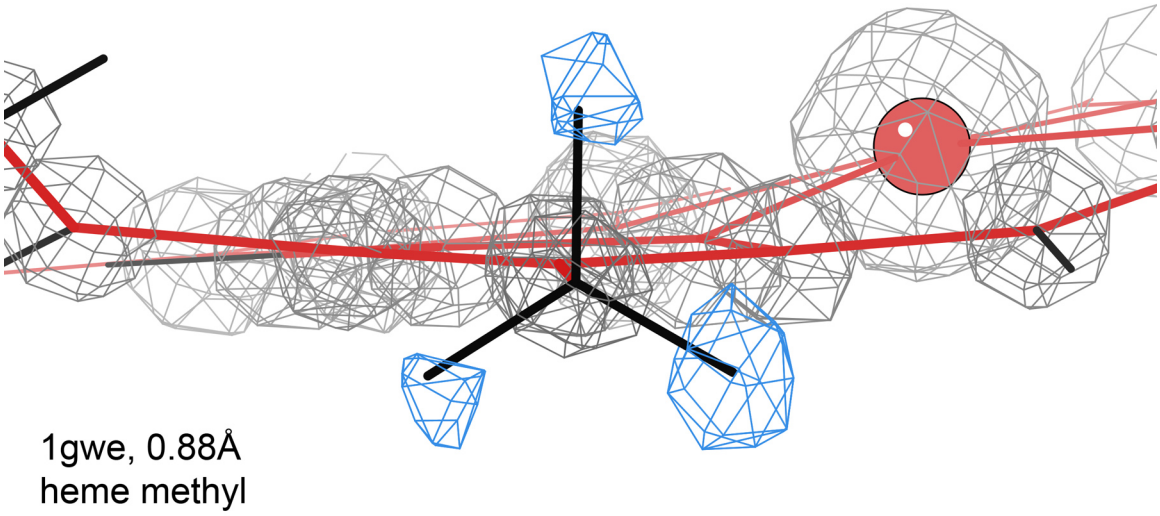


Figure 1: Correct H atom placement (bonds in black) for the Cmc heme methyl in the 1gwe catalase structure (Murshudov 2002), with one H perpendicular to the plane of the local heme ring (red). Blue contours are the methyl H difference density at +2.7σ, calculated from the model with hydrogens deleted. Gray contours are 2mFo-DFc electron density at 5σ.

aromatic rings prefer an in-plane position. Figure 2 shows the clear, positive difference peak for the H of a Tyr OH at 0.69Å resolution, in the plane of the Tyr aromatic ring. This case also has a well-placed H-bond to reinforce that position, but Tyr OH favors either the 0° or 180° in-plane position even with no H-bond. An NH_2 (such as on a guanine base), in contrast, is a simpler case, because the two H atoms occupy both of the possible in-plane positions and have no alternatives.

The bottom line

Hydrogen addition, placement and optimization is done by various automated routines. However,

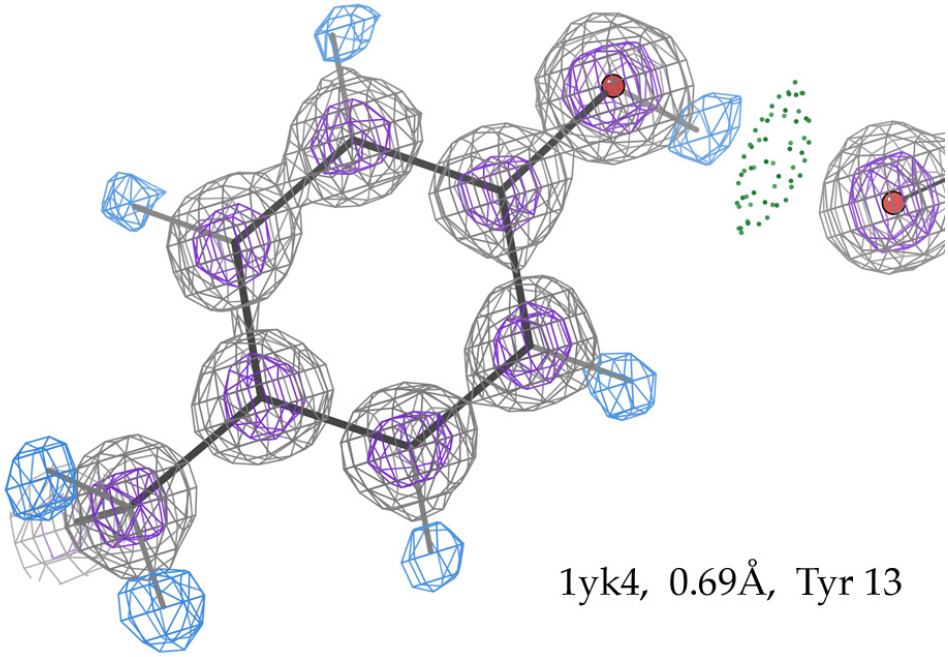


Figure 2: Correct placement of an in-plane OH hydrogen on a Tyr ring, confirmed by an H-bond (green dots) and a positive difference density peak (blue contours) at 0.69Å resolution in the 1yk4 rubredoxin structure (Bonisch 2005).

understanding the differing geometries of favored conformations for H atoms whose parent atom is attached to the edge of an aromatic ring can provide a sanity check on whether the automated system is doing those cases correctly, which has not always been true. Incorrect placement can produce surprisingly large clashes to nonpolar H or miss H-bonds for polar H. And if you're lucky enough to have ultra-high resolution, you can check the hydrogen difference density.

References:

Bonisch H, Schmidt CL, Bianco P, Ladenstein R (2005) *Acta Crystallogr* **D61**: 990-1004 [1gwe]

Murshudov GN, Grebenko AI, Brannigan JA, Antson AA, Barynin VV, Dodson GG, Dauter Z, Wilson KS, Melik-Adamyan WR (2002) The structures of Lysodeiktius catalase, its ferryl intermediate (Compound II) and NADPH complex, *Acta Crystallogr* **D58**: 1972-1982 [1yk4]

PDB50: Celebrating 50 Years of the Protein Data Bank in 2021

Worldwide Protein Data Bank Foundation

Correspondence email: zardecki@foundation.wwpdb.org

In 1971, the structural biology community came together and established the single worldwide archive for macromolecular structure data–the Protein Data Bank (PDB)–as the first open access digital data resource in all of biology. From its inception, the PDB has embraced the FAIR Principles (Findability, Accessibility, Inter-operability, Reusability). PDB data are used directly by many millions of users exploring fundamental biology, biomedicine, biotechnology, bioengineering, and energy sciences, and repackaged and distributed by more than 400 other digital data resources.

Since 2003, the PDB archive has been jointly managed by the Worldwide PDB (wwPDB, wwpdb.org) partnership, which ensures that the PDB is freely and publicly available to the entire global community of data depositors and data consumers at no charge and with no limitations on usage. Current wwPDB member organizations include the US RCSB Protein Data Bank (RCSB.org), the Protein Data Bank in Europe (pdbe.org), Protein Data Bank Japan (pdbj.org), and the Biological Magnetic Resonance Data Bank (bmrb.io; headquartered in the US, also operating in Japan).

Today, the PDB archive contains >170,000 structures of proteins, nucleic acids, and complex assemblies that help researchers, educators, and students understand facets of the entire spectrum natural, physical, and engineering sciences in three-dimensions at the atomic level, spanning from agriculture to zoology.

To celebrate the first half centenary of the PDB, wwPDB partners will host golden anniversary symposia and other celebratory events scheduled throughout 2021.

The inaugural wwPDB sponsored Global Online Celebration will occur on May 4-5, 2021, hosted by the American Society for Biochemistry and Molecular Biology. Registration will open in January.

Confirmed Speakers include:

Edward Arnold - Rutgers, The State University of New Jersey
Helen M. Berman - Rutgers, The State University of New Jersey and University of Southern California
Thomas L. Blundell - University of Cambridge
Alexandre M. J. J. Bonvin - Utrecht University
Stephen K. Burley - Rutgers, The State University of New Jersey and University of California San Diego
Wah Chiu - Stanford University
Johann Deisenhofer - University of Texas Southwestern Medical Center
Juli Feigon - University of California Los Angeles
Angela M. Gronenborn - University of Pittsburgh
Jennifer L. Martin - University of Wollongong
Stephen L. Mayo - California Institute of Technology
Zihe Rao - ShanghaiTech University and Tsinghua University
Hao Wu - Boston Children's Hospital and Harvard Medical School

Additional international/regional events and celebrations will include the 2021 American Crystallographic Association Transactions Symposium, an International Union of Crystallography 25th Congress Satellite Meeting, an EMBL Meeting: Bringing Macromolecular Structures to Life, and a symposium at the 2021 Asian Crystallographic Association Meeting. Please visit the regularly updated wwPDB Foundation website (foundation.wwpdb.org/pdb50.html) for announcements of additional events and the wwPDB website (wwpdb.org/pdb50) for other ways that you and your colleagues to celebrate PDB50.

Lessons from using the Cambridge Structure Database: II – Element specification X/Z

Nigel W. Moriarty[a*]

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

[*]Correspondence e-mail: nwmoriarty@lbl.gov

Preface

Continuing the series about lessons from using the Cambridge Structural Database (CSD), this work delves deeper into the nuances of the Conquest interface. More information about goals in the previous installment (Moriarty, 2020).

Introduction

Conquest (Bruno *et al.*, 2002), a structure based search tool, is an interface to the Cambridge Structure Database (CSD, Groom *et al.*, 2016). The CSD is arguably a better source for accurate geometry information that is greatly enhanced by the user-friendly tools for dredging up the structural matches.

One feature of the Conquest search is the specification of internal coordinates that can be used in analyses. For example, one can add a specific bond in the search fragment the length of which is determined for each of the matching entries. This results in a list of matching entries along with the value of the bond length in each. This is exactly the desired information for determination of the ideal values for restraints.

Element specification of X versus Z

Arginine is a charged essential amino acid containing the guanidinium moiety. Figure 1 shows the guanidinium group terminating the side chain with a positively charged central carbon atom and an electronic resonance bond structure. Note that the generally planar

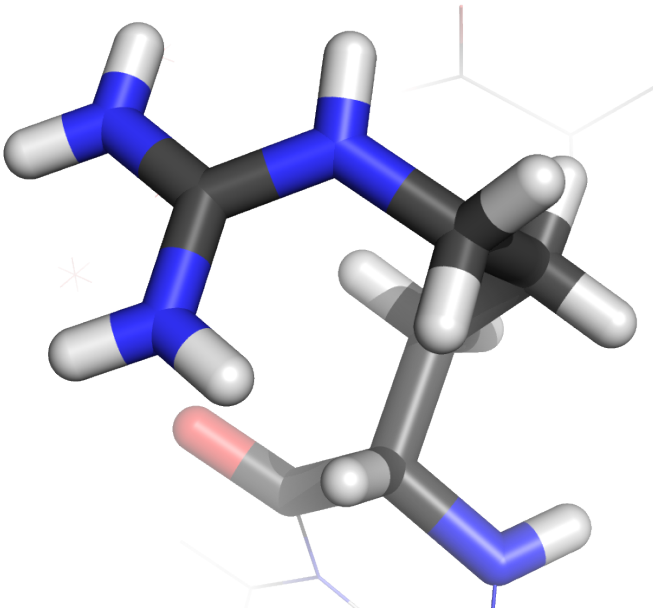


Figure 1: Example of an arginine amino acid.

structure causes one of the nitrogen atoms (Nη1) to be *cis* to the 1-4 interactive carbon atom (Cδ). This steric interaction causes the Nη1–Cζ–Nε angle to be larger than Nη2–Cζ–Nε as determined by Malinska *et al.* (2016). Part of the work involved using the CSD.

Recently, the CSD structure search of the guanidinium group was repeated as part of study (Moriarty *et al.*, 2020) into the planarity of the guanidinium group. Using the Conquest chemical fragment interface, a simple search can be constructed as shown in figure 2a. Searching with the filters in figure 2b results in 208 matching entries.[1] The number of entries is increased but the general point is

[1]Note that the version of the CSD database used in this work – November 2019 + 2 updates – was different from both the two previous works.

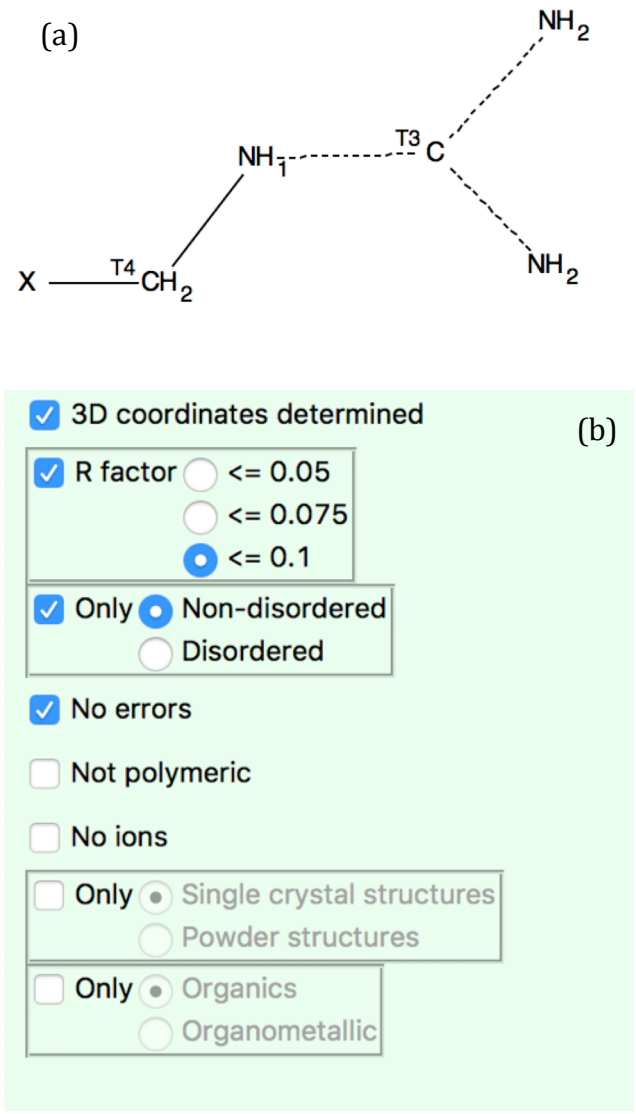


Figure 2: (a) Simple Conquest search fragment for the arginine. Image taken from Conquest Draw window. (b) Conquest filter settings.

the same. In addition, a set of internal coordinates was specified in order to get values suitable for restraints. These included the dihedral from the X to the branching carbon atom. A total of 208 matching entries containing 898 examples of the search fragment were found. This implied that, on average, there were more than 3 examples of moiety in each matching entry. However, a viewing of each of the entries seemed to contradict that assertion. Furthermore, the list of results specifies that 3 examples exist in ARGIND11 but the entry (shown in figure 3) appears to have only one guanidinium.

The problem lies with the use of X at the connecting end of the side chain. Conquest uses X to search for *any* atom at that position. This includes hydrogen atoms. Therefore, the search finds three examples of the fragment and internal coordinates because in arginine the Cδ atom is bound to a carbon atom (Cγ) and two hydrogen atoms (Hγ2, Hγ3). This effectively triples the number of instances of the search fragment and would contaminate the internal coordinates that include the X atom.

The image in figure 3 is produced by Mercury (Macrae *et al.*, 2006, 2008), a visualisation and analysis in the CSD suite. Another clue to the unusual behaviour of the X designation is the three torsion angles shown including two ending at Hydrogen atoms.

Thankfully, Conquest has an alternative to X that solves the problem. The Z specification matches any atom except hydrogen atoms. Repeating the search with Z substituted for X results in 208 entries and 312 examples. It also removes the additional torsion angles from the Mercury interface.

Conclusions

As stated in the conclusions of the previous article in this series "Always verify that the results from a structure search are reasonable." The first simple search attempt in this example resulted in too many examples of the guanidinium group. Using the X and Z element specification correctly is a powerful tool for filtering results.

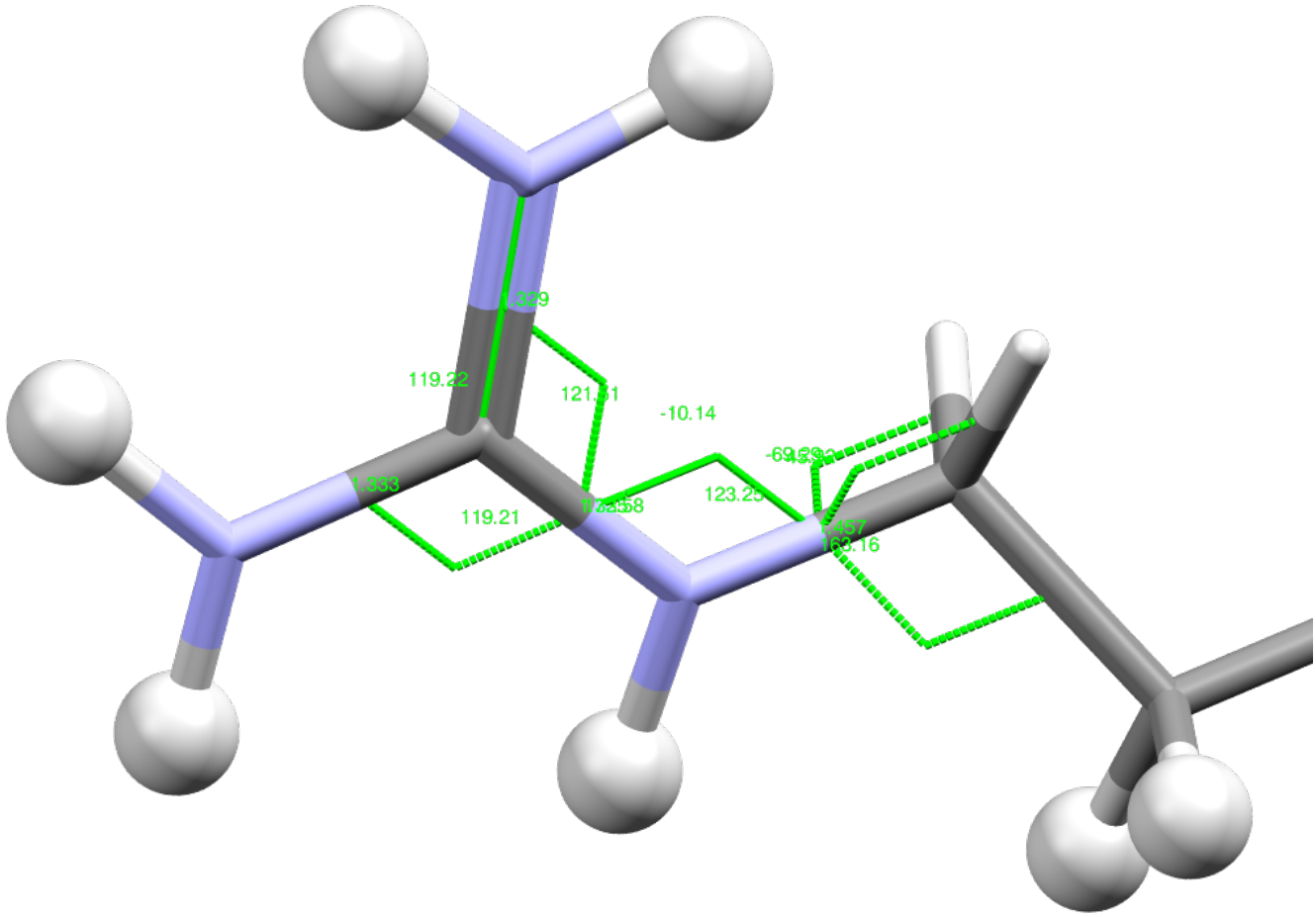


Figure 3: Conquest search fragment for the arginine showing the guanidinium group with internal coordinates in green.

References

Bruno, I. J., Cole, J. C., Edgington, P. R., Kessler, M., Macrae, C. F., McCabe, P., Pearson, J. & Taylor, R. (2002). *Acta Crystallogr. B.* **58**, 389–397.

Groom, C. R., Bruno, I. J., Lightfoot, M. P. & Ward, S. C. (2016). *Acta Crystallogr. Sect. B Struct. Sci. Cryst. Eng. Mater.* **72**, 171–179.

Macrae, C. F., Bruno, I. J., Chisholm, J. A., Edgington, P. R., McCabe, P., Pidcock, E., Rodriguez-Monge, L., Taylor, R., Streek, J. van de & Wood, P. A. (2008). *J. Appl. Crystallogr.* **41**, 466–470.

Macrae, C. F., Edgington, P. R., McCabe, P., Pidcock, E., Shields, G. P., Taylor, R., Towler, M. & Streek, J. van de (2006). *J. Appl. Crystallogr.* **39**, 453–457.

Malinska, M., Dauter, M. & Dauter, Z. (2016). *Protein Sci. Publ. Protein Soc.* **25**, 1753–1756.

Moriarty, N. W. (2020). *Comput. Crystallogr. Newsl.* **11**, 7–10.

Moriarty, N. W., Liebschner, D., Tronrud, D. E. & Adams, P. D. (2020). *Acta Crystallogr. Sect. Struct. Biol.* **76**, 1159–1166.

The Importance of Being Positive: Charged ligands that bind nucleic acids

Jane S. Richardson[a], Nigel W. Moriarty[b], Christopher J. Williams[a], & David C. Richardson[a]

[a]Department of Biochemistry, Duke University, Durham, NC 27710, USA

[b]Molecular Biosciences and Integrated Bioimaging, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

Introduction

Both PDB and mmCIF format include possible ways to specify explicit charge. However, that is only seldom used and charge is primarily coded by adding or subtracting H atoms, which is done from dictionary lookups. Those dictionaries have been tuned quite well for proteins in the usual pH range, with NH_3 in Lys or at the N-terminus, 5 hydrogen atoms on Arg guanidinium, and negative COO in Asp, Glu, and C-terminus. Charges are respected on single-atom ions such as Cl^-, Mg^+, or Zn^{2+}. However, by current convention, neutrality is constrained for essentially everything else. That has involved adding a hydrogen atom to one oxygen atom of each nucleic-acid phosphate, distorting conformation for ring-buried charges such as 1-methyl adenine, and making non-protein NH_3 groups such as spermidine into NH_2 groups (either with planar or with tetrahedral geometry). Two H atoms on a tetrahedral N would presumably occupy some mixture of the three possible positions, so even at a pH favoring neutrality the best, but probably impractical, procedure would actually be to model all three H atoms at 2/3 occupancy and refine those occupancies with respect to local contacts.

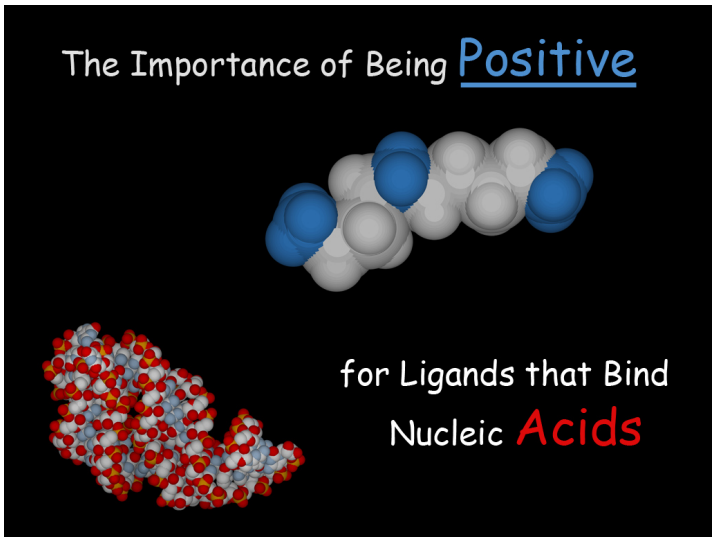


Figure 1: Positive spermidine (NH atoms blue) stabilizes negative RNA (red PO_4 O atoms).

It is not possible to solve this charge problem correctly as a general case, without detailed information about pH and local environment plus difficult calculations, since a great many cases have more than one possibility. Most of us are familiar with that issue for histidine protonation. However, we would advocate for expanding charge exceptions beyond the currently provincial protein-centric dictionaries to include at least the most important exceptions appropriate for nucleic acids and their commonest ligands.

Positive charges for spermidine & spermine

Spermidine, with three amino groups, and spermine, with four, are the most common biological polyamines, serving quite varied roles. In structural biology they are primarily seen as stabilizers of nucleic acid backbone in 3D structures, similar to the effect of Mg^+. The positive charges on those amino groups (NH_3 ends and internal tetrahedral NH_2 groups) are of course central to that function, making strong and directionally specific H-bonds, mostly to phosphate oxygen atoms. Figure 2 above shows the striking contrast in H-

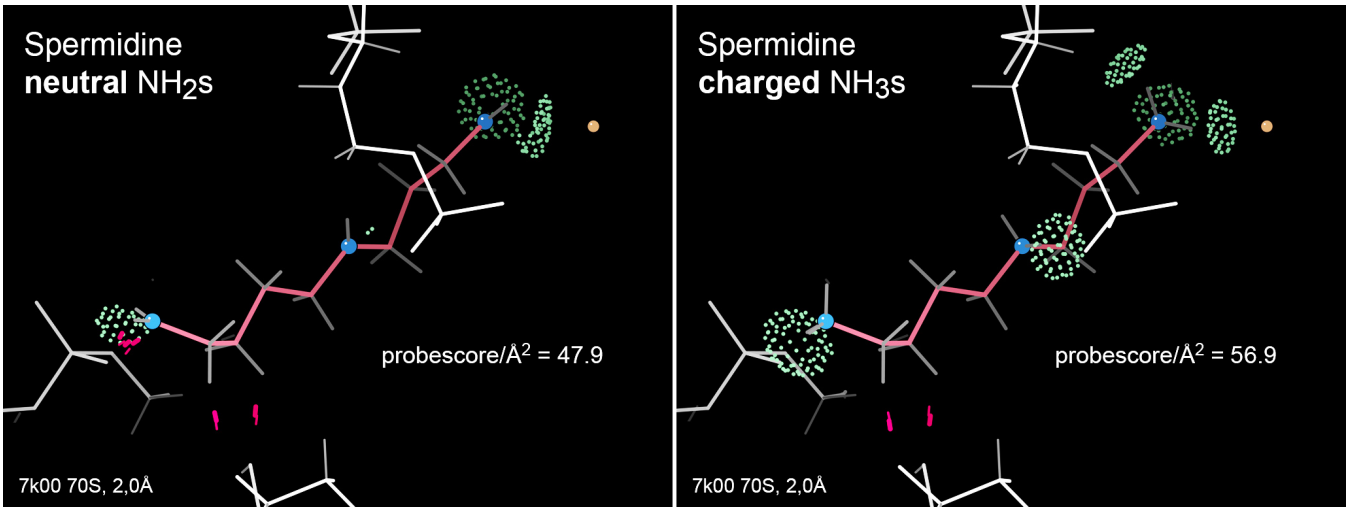


Figure 2: Spermidine (SPD) a6218 in Jamie Cate's new 7k00 cryoEM ribosome structure at 2Å resolution (Watson 2020), with all-atom contacts for clashes (hotpink spikes) and H-bonds (pillows of pale green dots). At left, neutral default-dictionary H atom placement gives so-so contacts and probescore, while charge-aware H atom placement gives 5 excellent H-bonds and a 12% better absolute probescore. Visualization in KiNG (Chen 2009)

bonding and clashes for a spermidine modeled as neutral (following the PDB dictionary) versus the same ligand modeled with charges. The MolProbity probescore (given in these figures) is a weighted combination of:

- -10 x unfavorable clash overlap volume,
- +4 x favorable H-bond overlap volume,
- +1 x a more complex function of favorable van der Waals contact closer than 0.5Å.

Those weights were chosen to approximate the vdW function for interaction of two isolated atoms (Word 1999). For probescores, as well as for nucleic-acid ligands, it's important to be positive.

Spermidine is actually a very difficult ligand for which to model the non-H atoms in the correct conformation, since at resolutions typically attainable for large RNA or DNA molecules it just looks like a long smooth tube without clear wiggles. The two small clashes of CH_2 groups at bottom in Figure 2 probably mean that this spermidine conformation is not quite right. Providing the right H atoms can allow fitting to do a bit better for spermidine and other cases, since you can then check alternative models for better H-bonding and fewer clashes.

Positive charges for RNA-directed inhibitors & antibiotics

Aminoglycoside antibiotics act by binding to the RNA in ribosomes and, as "amino" in their name suggests, they carry multiple positive charges. The 7k00 ribosome structure has bound paromomycin, which has four sugar rings labeled I - IV and five charged nitrogen atoms, marked by blue balls in the figures. This and similar antibiotics are strong mutagens for bacteria: normally, the two bases at lower right in Figure 3 are extruded from the base-paired helix only if the correct tRNA is bound, but the antibiotic by itself extrudes the two bases and turns on that signal all the time, as seen here, allowing an incorrect amino acid to be added to the growing chain quite frequently.

Looking locally at the charge sites on individual rings shows the charge consequences much more dramatically. Ring II of paromomycin has two nitrogen branches.

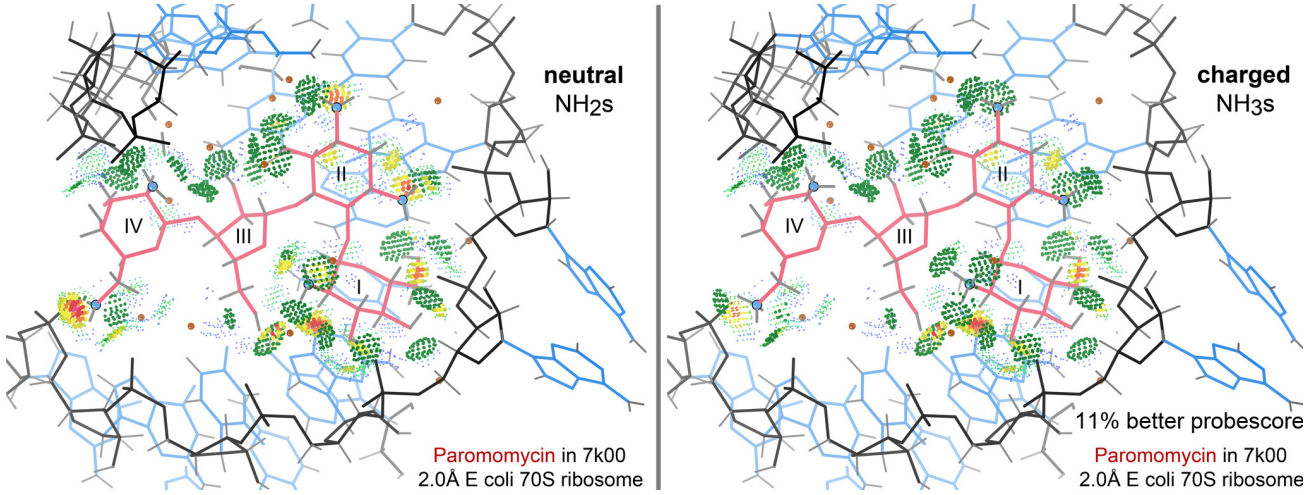


Figure 3: Paromomycin antibiotic bound to the 7k00 bacterial ribosome. At left, adding NH_2 H atoms to the 5 nitrogens gives overall ligand neutrality and a quite decent absolute probescore of 69.9. At right, NH_3 H atoms give + charges, fewer clashes (red and yellow), more H-bonds (green), and an 11% better probescore of 77.5.

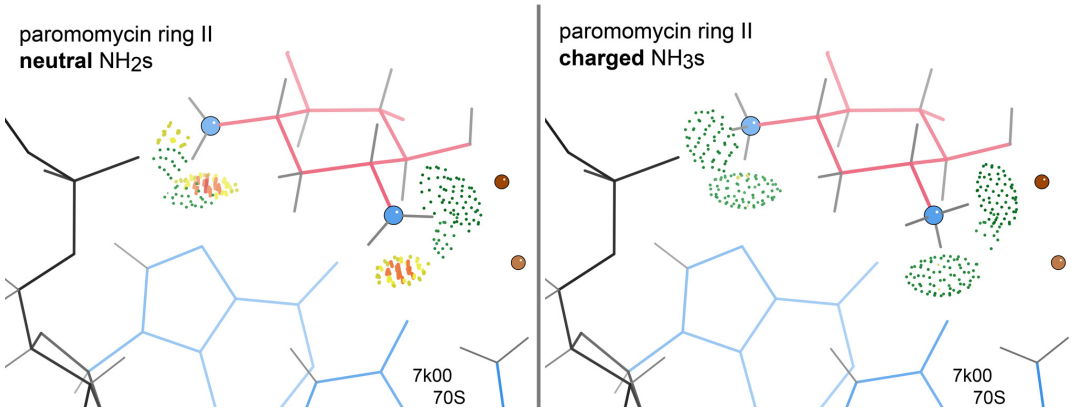


Figure 4: Contact quality for paromomycin ring II with neutral NH_2 (at left) versus charged NH_3 (at right).

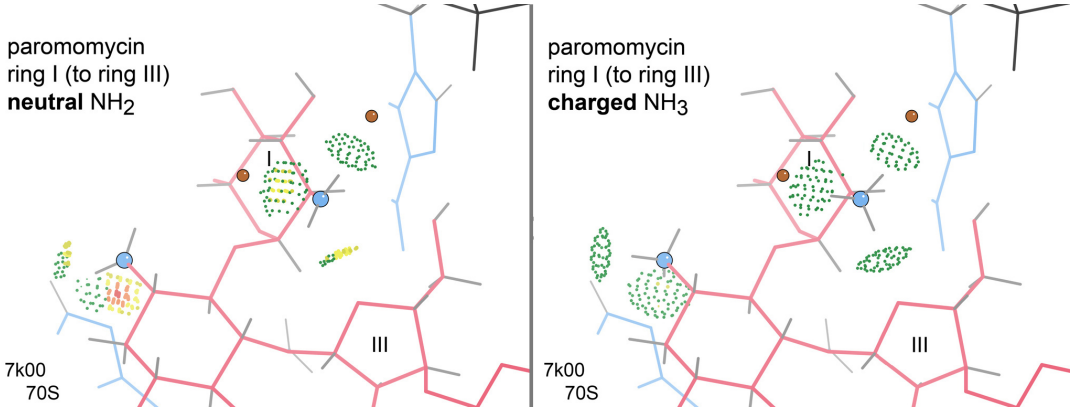


Figure 5: Contact quality for paromomycin ring I. This N21 NH_3 group replaces small overlaps with H-bonds, and also introduces a stabilizing H-bond between ring I and ring III.

If they are given uncharged NH_2 groups, the left side of Figure 4 below shows clash overlaps and poor H-bond geometry. If they are given tetrahedral, positively charged NH_3 groups, the right side of Figure 4 shows excellent H-bonds and no unfavorable overlaps at all.

Paromomycin rings I and III are even more interesting, as seen in Figure 5. Here, the NH_3 version of ring I trades slight bad overlaps for good H-bonds. Even more importantly there is then a good H-bond to the ring O of ring III, which stabilizes the compact overall binding conformation of the paromomycin.

Better charges now in Phenix and MolProbity

The libraries in Phenix and MolProbity, and in most other model-building, validation or refinement packages, have always shown RNA and DNA phosphates as negatively charged (that is, with no extra H). In the latest updates of MolProbity and Phenix, the libraries now also correctly treat the always-positive ligands shown here and some similar ones. As additional unambiguous cases are identified, those will also be added. If you deal with another such ligand that is unambiguously charged across usual biological environments, we'd appreciate hearing about it (and its 3-letter PDB residue name).

These comparisons also show that MolProbity's probescore is a sensitive and useful measure of model quality for bound ligands. It can be run with *phenix.probe* on the command-line (see notes below) to test alternative possible models for charge or conformation. The contact dots or scores make explicit what the surrounding macromolecule is actually seeing on the ligand, in your particular structure. The Reduce program quite successfully uses such probescores to determine protonation and orientation of histidine rings.

Technical notes:

The probescore (Word 1999) and details of its component scores can be obtained for a ligand in a PDB-format file with H atoms added & optimized, e.g. the SPD 350 A-chain ligand in the 1pot spermidine-binding protein, by the following command:

phenix.probe -c -both "chainA 350" "not (chainA 350)" 1potH.pdb >1potH_SPD-350_probescore.txt

The text output appears in Schema 1. The probescore (underlined above) is the sum of 1) the "grand tot" probescore with the ligand as source atoms and the surroundings as target plus 2) the "grand tot" probescore with the surroundings as source atoms and the ligand as target. It becomes larger for larger ligands, so comparisons are only valid as relative probescores between different instances of the same ligand. For a ligand that is actually seen to bind, a negative probescore means the model must be fit wrong. "Type" is either the atom type, or is the parent atom for an H. Probescores are also used (with the -both flag and two atom selections) for the interactions of a single sidechain with its surroundings (in Reduce's optimizations and flips), for macromolecular interfaces, for all interactions inside a macromolecule (with a -self flag and just one atom selection) or, with custom atom selections, for instance to score and show all interactions between pairs of carboxyl oxygens, as for Fitting Tip #13 about "O-pairs" (Richardson 2017).

To visualize those ligand-to-protein all-atom contacts in KiNG, first make a 1potH kinemage of the model (easiest is to drop

```
program: probe.2.16.160928, run Mon Nov 16 06:25:16 2020
command: probe -c -both "chaina 350" "not (chaina 350)" 1potH.pdb
selection: both
name: dots
density: 16.0 dots per A^2
probeRad: 0.250 A
VDWrad: (r * 1.000) + 0.000 A
score weights: gapWt=0.25, bumpWt=10, HBWt=4

subgroup: 1->2; atoms: 32; max dots: 3124; max area: 195.2 A^2
  type                 #       %       score score/A^2 x 1000
  C wide_contact       333   10.7%        3.6      18.44
  C close_contact      271    8.7%       11.7      60.02
  C small_overlap      115    3.7%       -5.1     -26.20
  C bad_overlap          4    0.1%       -0.5      -2.76
  N wide_contact        90    2.9%        0.8       4.33
  N close_contact      122    3.9%        5.6      28.57
  N small_overlap       29    0.9%       -1.0      -5.01
  N H-bond             136    4.4%        3.2      16.49

     tot contact:      816   26.1%       21.7     111.35
     tot overlap:      148    4.7%       -6.6     -33.96
     tot  H-bond:      136    4.4%        3.2      16.49

       grand tot:     1100   35.2%       18.3      93.89

contact surface area: 68.8 A^2

subgroup: 2->1; atoms: 5945; max dots: 685045; max area: 42815.3 A^2
  type                 #       %       score score/A^2 x 1000
  C wide_contact       318    0.0%        3.3       0.08
  C close_contact      284    0.0%       12.4       0.29
  C small_overlap      105    0.0%       -4.7      -0.11
  C bad_overlap          3    0.0%       -0.4      -0.01
  C H-bond              11    0.0%        0.1       0.00
  N wide_contact         5    0.0%        0.0       0.00
  N close_contact       12    0.0%        0.6       0.01
  N small_overlap        5    0.0%       -0.1      -0.00
  O wide_contact       115    0.0%        1.2       0.03
  O close_contact      113    0.0%        4.9       0.11
  O small_overlap       30    0.0%       -0.9      -0.02
  O H-bond             116    0.0%        2.7       0.06

     tot contact:      847    0.1%       22.5       0.52
     tot overlap:      143    0.0%       -6.2      -0.14
     tot  H-bond:      127    0.0%        2.8       0.06

       grand tot:     1117    0.2%       19.1       0.45
       total probescore                  37.4

contact surface area: 69.8 A^2
```

Schema 1: Sample output from the phenix.probe command

1potH.pdb onto the KiNG window, ask for a "lots" kin and save it). Then run the same command as above but substituting -kin for the -c flag (stands for "count") and appending the output onto your lots kinemage:

phenix.probe -kin -both "chainA 350" "not (chainA 350)" 1potH.pdb >>1potH_SPD_lots.kin

The spermidine in 1pot is an interesting case because the protein binds the SPD with good specificity by sandwiching its chain between

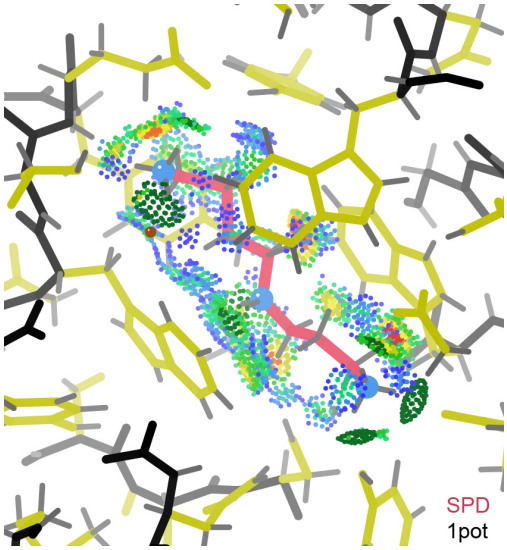


Figure 6: Probescore contacts for the SPD (pink) in the 1pot SPD-binding protein, bound mainly by pairs of aromatics.

two near-orthogonal pairs of aromatics, whose extensive van der Waals contacts are a much more dominant component of the probescore than the H-bonds (Figure 6). This example serves as a reminder that vdW contacts should indeed be taken into account as well as clashes and H-bonds.

At present, *phenix.probe* (or a locally installed copy of Probe) cannot read mmCIF files directly, but can read hybrid36 PDB files. So for very large files like 7k00, one should first run *phenix.cif_as_pdb*. Our rewrite of Probe now in progress will fix that problem.

References:

Chen VB, Davis IW, Richardson DC (2009) KiNG (Kinemage Next Generation), a versatile interactive molecular and scientific visualization program, *Protein Sci* **18**: 2403-2409

Richardson J, Prisant M, Williams C, Deis L, Videau L, Richardson D (2017) Fitting Tip #13 - O-pairs: The love-hate relationship of carboxyl oxygens, *Comp Cryst Newsletter* **8**: 2-5 (http://phenix-online.org/newsletter/CCN_2017_01.pdf)

Sugiyama S, Matsuo Y, Maenaka K, Vassylyev DG, Matsushima K, Kashiwagi K, Igarashi K, Morikawa K (1996) The 1.8Å X-ray structure of the *E coli* POTD protein complexed with spermidine, and the mechanism of polyamine binding, *Protein Sci* **5**: 1984-1990 [1pot]

Watson ZL, Ward FR, Meheust R, Ad O, Schepartz A, Banfield JF, Cate JH (2020) Structure of the bacterial ribosome at 2 Ångstrom resolution, *Elife* doi:10.7554/eLife.60482 [7k00]

Word JM, Lovell SC, LaBean TH, Zalis ME, Presley BK, Richardson JS, Richardson DC (1999) "Visualizing and Quantitating Molecular Goodness-of-Fit: Small-probe Contact Dots with Explicit Hydrogen Atoms", *J Mol Biol* **285**: 1711-1733

HKLviewer, a new 3D reflection data viewer for CCTBX

Robert D. Oeffner[a], Airlie J. McCoy[a], Duncan H. Stockwell[a], Massimo D. Sammito[a], Kaushik S. Hatti[a,b], Tristan I. Croll[a] and Randy J. Read[a]

[a]Department of Haematology, Cambridge Institute for Medical Research, University of Cambridge, Hills Road, Cambridge CB2 0XY, United Kingdom
[b]Drug Discovery Unit, Wellcome Centre for Anti-Infectives Research, School of Life Sciences, University of Dundee, Dow Street, Dundee DD1 5EH, United Kingdom

Correspondence e-mail: rdo20@cam.ac.uk

The **HKLviewer** is a reflection data viewer for X-ray crystallography. It is now part of the **CCTBX** [1] library and available in software derived therefrom.

Introduction

As computational methods for crystallography are becoming ever more sophisticated a greater part of structure solving now concerns data analysis as to recognise deviations of reflection intensities from the Wilson distribution that underpins the theory for maximum likelihood methods used for structure solving. There are automated methods for handling most of these. However, occasionally pathologies in some data sets escape detection, leading to difficulties in solving the structure. The **HKLviewer** can help the visual detection of such cases. Other reflection data viewer programs exist, such as the **Reflection data viewer** [2] or **ViewHKL** [3] or the **StarANISO server** [4]. But they do not address our needs, one of which is that it must be possible to integrate the reflection data viewer into a GUI, **phaser.voyager** [5] that leverages new features in the **Phasertng** [6]. Both are currently under development and have not yet been released.

Implementation

HKLviewer Qt5 GUI

The built-in GUI of the **HKLviewer** uses Qt5 [7] and is written in Python using *PySide2* modules. Qt5 is a cross platform GUI architecture for MacOS, Linux and Windows. It is well supported and robust and encourages a clean programming style.

ZMQsocket

A *ZMQsocket* [8] is used for communication between the *HKLviewer Qt5 GUI* and *cctbx.python*. The *ZMQsocket* library implements a bidirectional messaging protocol and has been ported to several programming languages. Since the *HKLviewer Qt5 GUI* and the *cctbx.python* are running as separate processes this provides an extra level of protection against exceptions in the unlikely event should one emerge from the *cctbx.python* process. It also makes the GUI appear responsive even when *cctbx.python* remains busy loading a file or calculating cartesian coordinates. Thus although not strictly necessary there are intrinsic benefits of using a network socket for calling the *cctbx.python* process in flexibility and improvements in the design. All messages from the *HKLviewer Qt5 GUI* to *cctbx.python* are formatted as PHIL [1] strings. The *cctbx.python* process then relays information back as strings that are organised typically as python dictionary data structures containing built-in python data types.

cctbx.python scripting

The *cctbx.python* process executes functions in the file `cmdlineframes.py`. It reuses a substantial part of the code of the **Reflection**

data viewer. This code leverages the API in **CCTBX** for handling crystallographic data such as matching different miller arrays to one another or converting coordinates from reciprocal space into cartesian coordinates. When expanding reflections to P1 and generating Friedel mates, the rotation matrices are converted from real space fractional coordinates to cartesian coordinates before submitting them to the browser running our JavaScript code. Inside the browser the matrices are used by WebGL API functions, like `r.applyMatrix3(…)`, for defining new reflections when doing the crystallographic expansion.

WebGL

Due to uncertainty of support for OpenGL on macOS [9] the **HKLviewer** uses WebGL for 3D visualisation. WebGL is an industry standard for 3D hardware supported graphics rendering in modern web browsers. There are many JavaScript libraries that abstract the intricate coding details of WebGL into robust higher-level JavaScript libraries. The NGL viewer [10] is a predecessor to MOL* [11]. It is based on other JavaScript libraries that implements WebGL. Unlike MOL* which is still undergoing development NGL is well-documented and mostly undergoing maintenance. Its API therefore remains stable. Importantly NGL is provided as small (less than 2Mb) static JavaScript library file that can be distributed in programs written by other developers, subject to the MIT license. This is what is used in the **HKLviewer**. The NGL API provides functions for drawing graphics primitives such as spheres, arrows and other shapes in a web browser. This is the only part of the API that the **HKLviewer** uses. No molecules are drawn with our use of NGL.

Websocket

The *websocket***Invalid source specified.** protocol is a bidirectional messaging protocol used for communication between the web browser and *cctbx.python*. It supports binary data transfer that is useful for transmitting images from the browser. In the file `cmdlineframes.py`, the *websockets* module is used for setting up a server. This server sends messages as plain strings to a *websocket* client listening on a specified port on localhost. The client is instantiated in our JavaScript file, `HKLJavaScripts.js`, that runs in a web browser. The *websocket* protocol is built into modern web browsers and requires no special libraries, though occasionally security settings may have to be relaxed to permit the client to connect. The inbound messages are picked up by the *websocket* client. An event handler in `HKLJavaScripts.js` causes the web browser to react to the incoming messages. The messages consist of coordinates of the reflection spheres, their RGB colour values, their radii, tooltips, requested orientations and other messages. Conversely the JavaScript code will send back messages either on request or automatically to the *websocket* server running in *cctbx.python*. These are messages such as the orientation and location of the coordinate system, any reflection that is clicked and other messages.

Architecture

The architecture of the HKLviewer is outlined in Figure 1. The *HKLviewer Qt5 GUI* starts *cctbx.python* as a subprocess and runs the file, `cmdlineframes.py`. One of the first messages sent from *cctbx.python* to the *HKLviewer Qt5 GUI* is the path of the HTML file the web browser should load. The HTML file lives as a temporary file on the user's computer for the lifetime of the session. Its only contents are references to `NGL.js` and the `HKLJavaScripts.js` files. The *HKLviewer Qt5 GUI* also contains a web browser available in Qt5. As soon as the HTML file is loaded into the browser functions in `HKLJavaScripts.js` will attempt to connect to the *websocket* server running in the *cctbx.python* process.

It is possible to replace components in this architecture. The process running

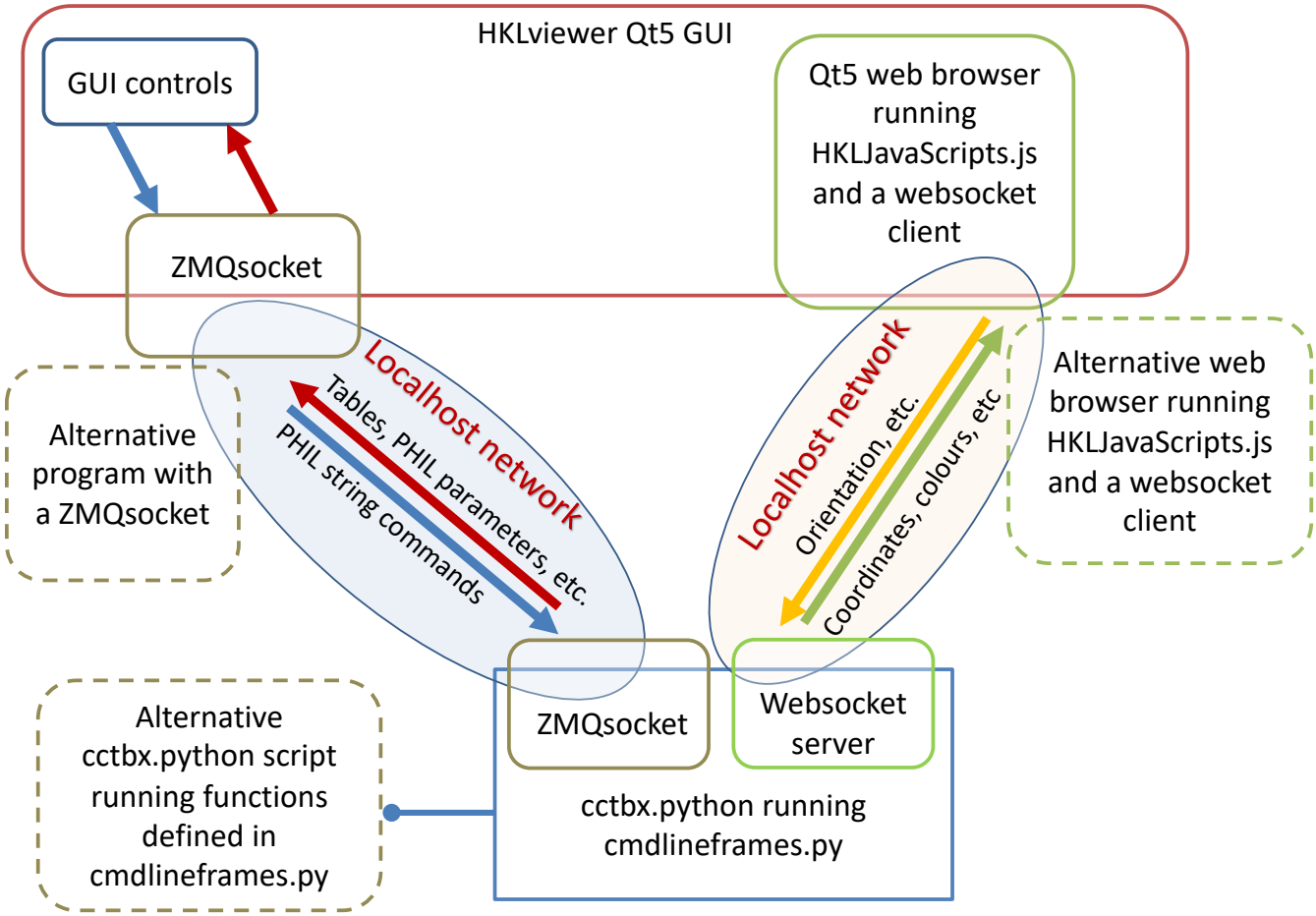


Figure 1: Architecture of the HKLviewer

`cmdlineframes.py` can be accomplished by an alternative process running *cctbx.python*. The *HKLviewer Qt5 GUI* can be replaced with an alternative program provided by a developer running *ZMQsocket* for communication with *cctbx.python*. The web browser program can also be set to either an alternative browser or the OS system default web browser. This yields several possibilities for incorporating **HKLviewer** into other programs. For **phaser.voyager** the approach will likely be to use a customised alternative *cctbx.python* script for executing functions in `cmdlineframes.py` as well as providing an alternative web browser for display of the reflections.

Usage

Standalone GUI

The primary way to invoke the **HKLviewer** is using it from the built-in GUI. It can be launched with a click on its program icon in a file browser. Alternatively, on a command line with the **CCTBX** environment enabled type `cctbx.HKLviewer`, optionally adding a reflection file name as an input parameter. **HKLviewer** reads most standard reflection file formats, like MTZ, HKL, SCA and CIF. It can save files as either CIF or MTZ. Some limitations exist with unmerged data. On invocation the *HKLviewer Qt5 GUI* as pictured in Figure 2 will appear. A particular data set in a reflection file can be displayed by double

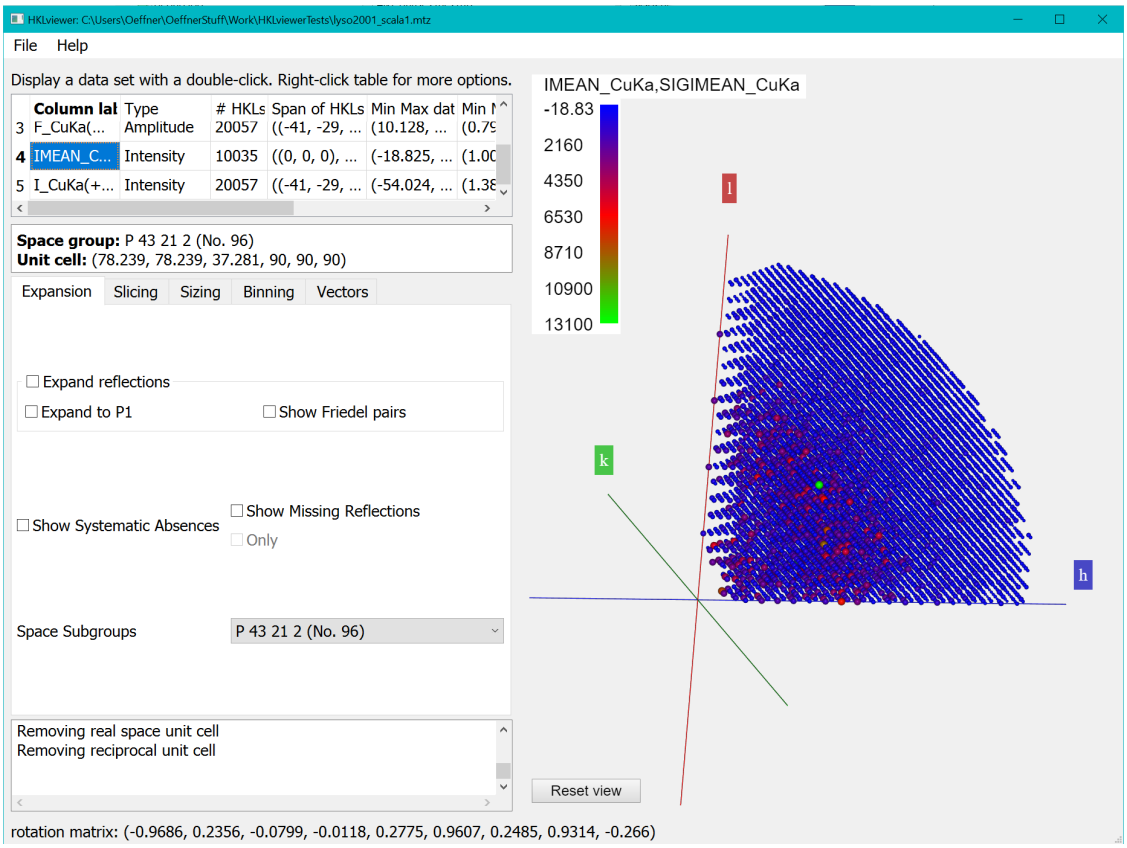


Figure 2: The HKLviewer loaded with a tutorial data set.

clicking the row with the corresponding label in the upper left table. That table contains a summary of properties of each data set in the reflection file. When a data set is displayed the tabs labelled "Expansion", "Slicing", "Sizing", "Binning" and "Vectors" become enabled. In the view of reflections, a tooltip will appear when the mouse is hovering over or clicking a reflection.

The *HKLviewer Qt5 GUI* consists of a left-hand panel and a right-hand reflection viewer area.

Expansion tab

On the "Expansion" tab the displayed reflections can be expanded to P1, to include their Friedel mates (for non-anomalous data) or both. In the **Reflection data viewer** this is done in python. This is computationally expensive and often leads to the **Reflection data viewer** becoming unresponsive simply due to the shear number of reflections to display. Our implementation benefits from computing this expansion in JavaScript within the memory space of WebGL. This avoids expensive transfer of data from CPU bound memory to GPU bound memory. As a result, a complete expansion of a data set in the **HKLviewer** typically takes just a few seconds for most data sets. For instance, expanding the data set with PDB code 2ASC that has cubic symmetry takes about 1.5 seconds on an Intel i7 multicore PC and uses about 500Mb memory. Doing this expansion in the **Reflection data viewer** made the program grind to a halt using memory in excess of 5-10 Gb.

As in the **Reflection data viewer**, we also provide check boxes allowing viewing systematic absences or missing reflections as well as the ability to reduce the space group symmetry to a subgroup.

Slicing tab

On the "Slicing" tab it is possible to view a slice of reflection either by imposing a clip plane within the view area of the browser or as is done in **Reflection data viewer** by drawing only a single selected plane of reflections. The first method allows for viewing arbitrary slices of the sphere of reflections at varying thickness. Due to implementation limitations of the NGL library the clip plane remains fixed parallel to the screen.

Sizing tab

On the "Sizing" tab the user can resize the spheres used for drawing each reflection. Reflections can be displayed either independent of their data values or as a monotone function of the data values. In our implementation the radius of a reflection sphere is a power function of its data value where the user can select the power factor to be used;

$\text{radius} = \text{value}^{p}$, where $0 \leq p$.

With $p = 1$ the sizes of the reflections are directly proportional to their data values. This may be unhelpful if illustrating intensities or amplitudes where there will be orders of magnitude difference between the sparse number of very strong reflections and the vast majority of more moderate or weaker reflections; only the strongest reflections would be visible. Conversely, by setting $p = 0$ all reflections will be shown with the same size.

It is also possible to use an automatically computed value for p that is defined so that the reflection with the smallest data value is only ten times smaller than the reflection with the largest data value. To do this leave the "User defined power scaling" check box unticked.

Binning tab

On the "Binning" tab a dataset can be divided into one or more bins according to resolution or values in another dataset from the same file. By entering a number greater than 1 in the "Number of bins" spin box control new bins will be created. The default is the put an equal number of reflections in each bin and compute the bin thresholds accordingly. This can be overridden manually by entering an explicit bin threshold in the "lower bin value" column for a selected bin. The produced new set of bins may not match the requested bin thresholds or number of bins if one or more bins are empty. The number of bins is limited to 40 for performance reasons. It is also possible to bin data for singletons in anomalous data, giving one bin for data with Friedel mates and one each for lone reflections with either the positive or negative hand mate missing.

It is also possible to adjust the transparency of the reflections in a particular bin by entering a number between 0 and 1 in the opacity column of that bin. Reflections with opacity values less than 0.3 will not react to mouse hovering or clicking activity. Due to implementation issues of 3D graphics there will often be border and shading artifacts when viewing reflections half transparent.

Vectors tab

The "Vectors" tab provides controls for showing some vectors with origin (0,0,0). These include vectors illustrating rotation axis for the symmetry operators within the space group for the currently displayed data set. Vectors specified by rotation operators are labelled 2-fold, 3-fold and so on. The labels and the associated rotation operator are listed in the table on the "Vectors" tab. The vectors for rotation operators part of the spacegroup are always listed whenever a dataset is loaded. The displayed length of rotation operator axes is set to be somewhat larger than the sphere of displayed reflections and bears no relation to the rotation operator.

In addition, if the reflection file is of the MTZ type and the file header contains information

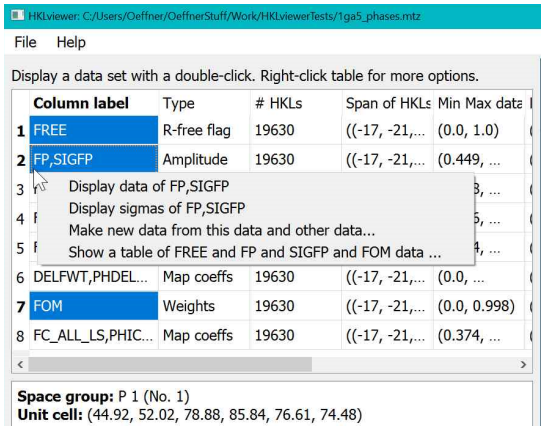


Figure 3: The context menu shown when right-clicking several highlighted rows representing datasets.

about a translational non-crystallographic symmetry (tNCS) vector (obtained from **Phasertng**) this vector will be listed as well in the table. Its real space fractional coordinates will be present in the "as abc" column.

It is also possible to enter a user-defined vector. This can either be a rotation operator, a vector in reciprocal space or a real space vector in fractional coordinates. In addition, it is possible to rotate reflections around a vector or to orient the reflections so that the displayed vector is normal or parallel to the screen. This is useful for visual inspection of data that possess tNCS or twin laws. The real space and reciprocal space unit cells can also be displayed.

Standard output

Below the five tabs mentioned above is a text field for standard output emitted by the program. Information about the files loaded as well as warnings or error message may appear there.

Context menu

Figure 3 shows the context menu that appears when right-clicking a row in the upper left table. If more than one row is highlighted the last item of the context menu will provide an option to show a table of those data.

Selecting the "Show a table" item will display a window with tabulated data of these

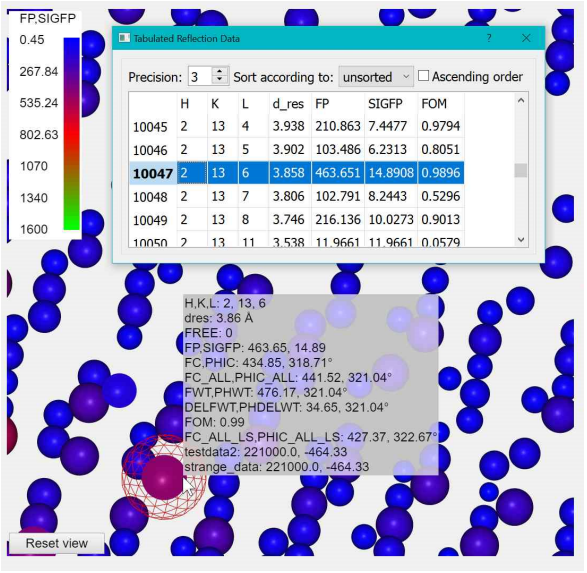


Figure 4: Tabulated Reflection Data window with a highlighted reflection in the viewer

reflections. The reflections remain unaltered throughout the session and will not under go crystallographic expansion if requested.

If a reflection in the table in Figure 4 is double clicked the viewer will zoom in on it and put a red wireframe around it. Likewise, right-clicking a reflection in the viewer will highlight the row(s) in the table with the same *hkl* index as the reflection.

Creating a new set of reflection data

A new set of reflection data can be created based on one or two existing sets of reflection data. Right click on a desired set of reflections in the upper left table. From the context menu in Figure 3 click on the item "Make new data from this data…". The dialog in Figure 5 opens up with a text field and a drop down list of available data set. This allows the user to quickly create a new dataset with features that may be of interest.

Colour mapping

In the **Reflection data viewer** there are only a few colour schemes available. We chose to make available all colour gradient mapping schemes present in the *MatPlotLib* module, which is included with the **CCTBX**. To select a

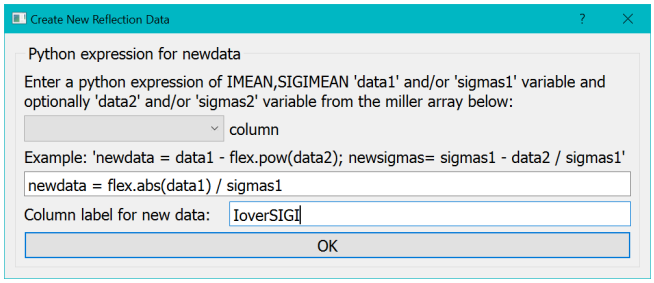


Figure 5: The "Create New Reflection data" dialog used for creating I/SigI data.

different colour mapping the user can click the upper left colour chart in the viewer area and the dialog in Figure 6 will appear.

A new colour mapping is chosen with a single mouse click on the desired colour scheme. As discussed in relation to the power scaling of the radii of reflection spheres it may be difficult to discern any features from the colour schemes if the values vary over orders of magnitude. Therefore, we have also implemented a mapping of the colour gradient with a power factor with 1 being the default. Values ranging from 0.15 and 2.59 were found to be sensible values for the majority of use cases.

Circular rainbow for phases of map coefficients

When displaying a dataset of map coefficients, representing amplitudes and phases, the **HKLviewer** colours the reflections so that the colour is a linear function of the phase value within the interval [0°, 360°]. In this case it makes sense to choose colour gradient maps from *MatPlotLib* which are circular, such as `gist_rainbow` or `hsv` where the starting colour is the same as the final colour. This is then mapped onto the interval of [0°, 360°]. A map coefficient is then displayed with the colour corresponding to its phase value. For map coefficients, the "Power factor for map scaling" has no effect.

If the reflection file contains a figure of merit (FOM) dataset in addition to map coefficients it is also possible to display map coefficients with colour saturation as a function of FOM

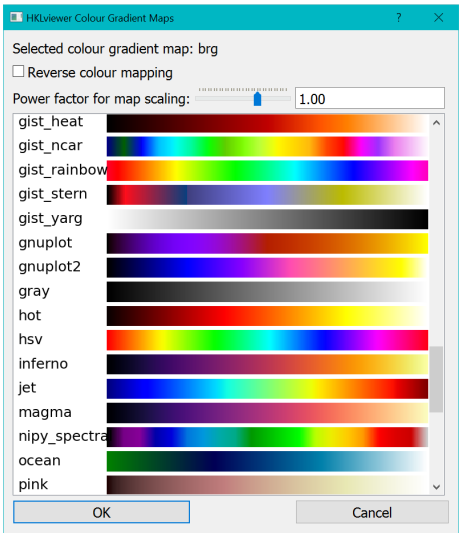


Figure 6: The Colour Gradient Map dialog for selecting different colour mapping.

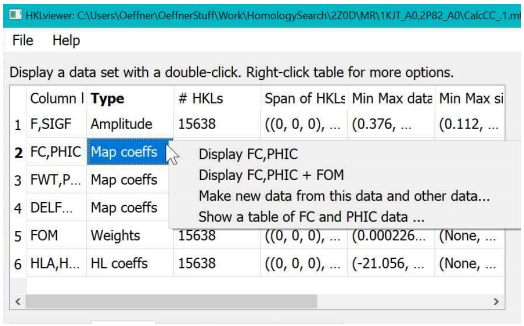


Figure 7: Context menu when FOM data is present together with map coefficients.

values. A FOM value of 1 or 0 causes full colour saturation or no colour saturation, respectively. This feature is available from the context menu as in Figure 7. Here one would select the item labelled "Display FC,PHIC + FOM".

A future version of **HKLviewer** may implement this feature for data files containing Hendrickson-Lattman coefficients. An example of visualising map coefficients is given in Figure 8.

Persistence

The layout of the GUI as well as some other properties such as font size persist across sessions. The GUI layout can be adjusted by tugging the borders between the four

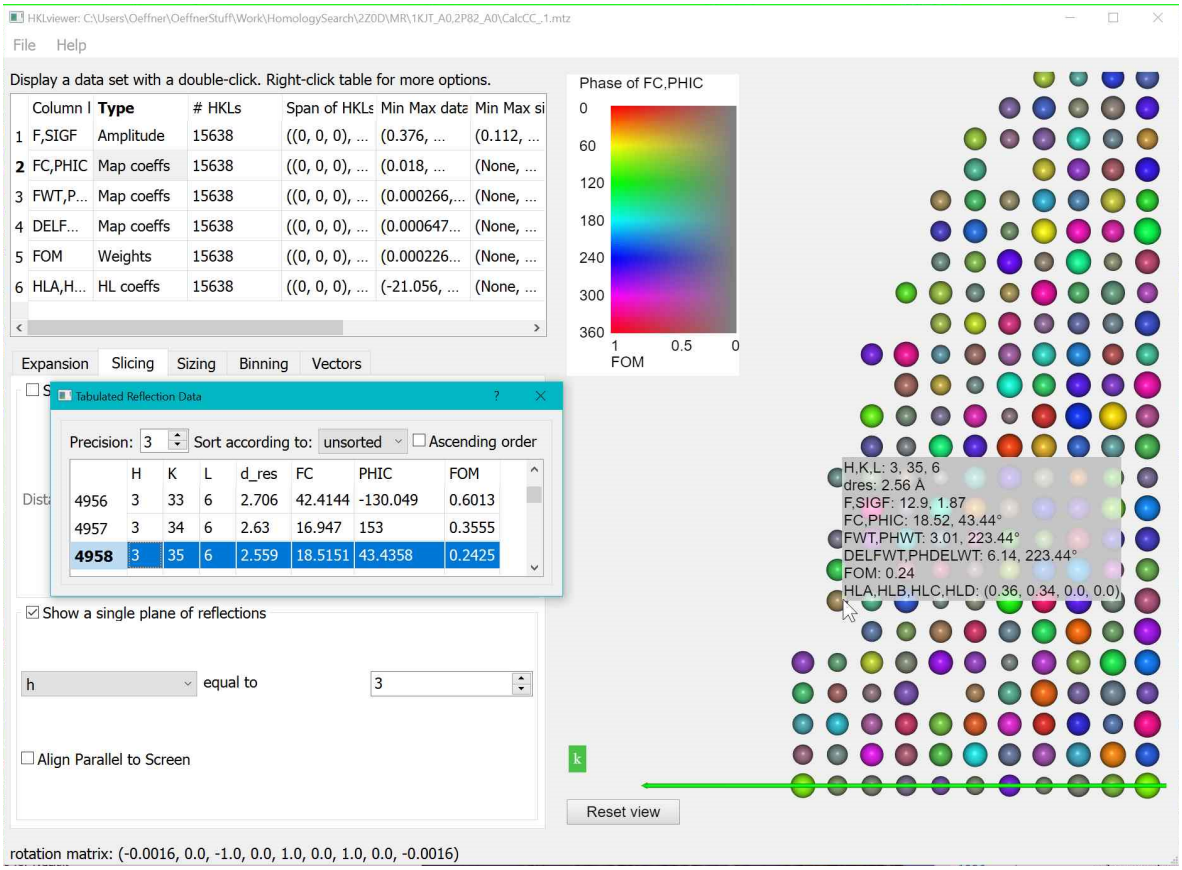


Figure 8: Inspecting a map coefficient together with a FOM value in a dataset containing both.

components to the left of the view area as well as the vertical border between the view area and the four components on the left.

Examples

Inspecting tNCS modulation in a dataset

In Figure 9 is an example of tNCS in the data set (PDB id: 6C5F) found by *xtriage*. The tNCS vector was found to be (0.333, -0.334, 0.040). With a Patterson peak height of 19.483% of the origin peak the tNCS modulation is moderate. But it is nevertheless visible with the settings used here for the **HKLviewer**.

In Figure 9, the "FP,SIGFP" reflections have been expand to P1 and Friedel mates. On the "Binning" tab, 8 bins were specified and data binned according to "FP,SIGFP". The few large data values in amplitude or intensity data set are likely to obscure subtle features such as tNCS modulation. In **Error! Reference source not found.**, we therefore deselected the last bin of data values larger than 900. This number was entered explicitly. That and the number of bins was found by trial and error.

On the "Sizing" tab, we have chosen a power factor of two to roughly reproduce the strength of recorded intensities (not present in the file). For this data set a scale factor of two makes most of reflections visible without expanding into space of neighbouring reflections. Rotating the reflections around the xtriageTNCS vector reveals the weak modulation in the data values as ripples perpendicular to the xtriageTNCS vector.

Inspecting a twin law for a dataset

The **Phenix** tutorial with the file, *porin.mtz*, illustrates twinning. In **Error! Reference source not found.**, the data set "F-obs,SIGF-obs" is displayed in the viewer. The reflections have been expanded to P1 and Friedel mates. The user defined power scaling was set to one. The twin operator, (-h-k,k,-l), has been

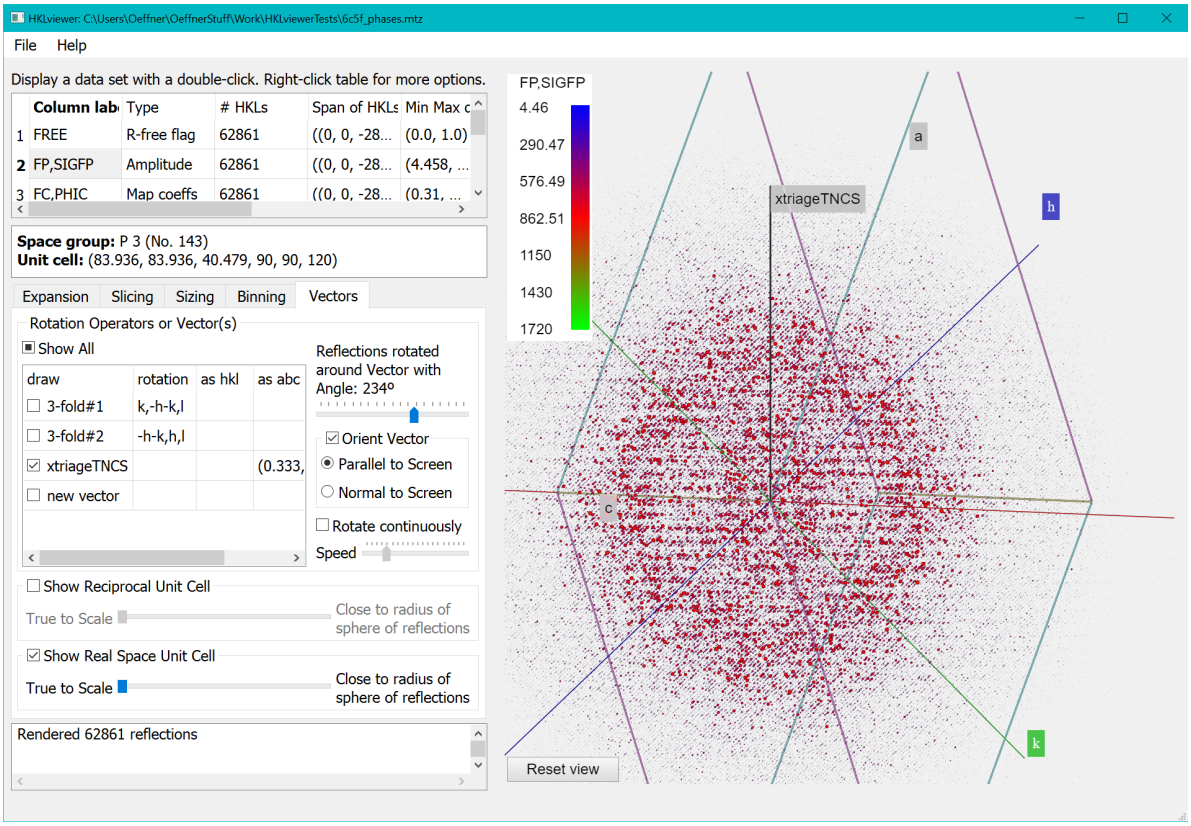


Figure 9: Displaying tNCS modulation in amplitude data.

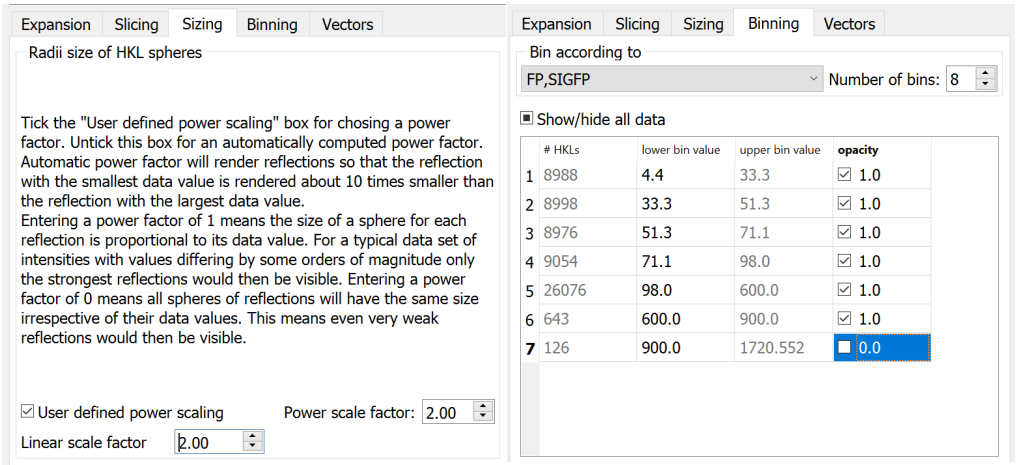


Figure 10: Adjusting size of displayed reflections and hiding the strongest reflections from view is helpful for revealing subtle features in the diffraction pattern like tNCS.

entered on the "Vectors" tab and oriented normal to the screen. A slice with a clip plane was made at a distance of five from the origin and a width of 1.2. Right-clicking a reflection, say the one at (13,5,8) invoke tooltips at the reflection itself as well as at its twin. The data value of the reflection is simultaneously highlighted in the "Tabulated Reflection Data" window.

Tutorials and further documentation

For tutorial examples on using **HKLviewer** and documentation on using it from python script or through a *ZMQsocket* we refer to the online documentation on http://cci.lbl.gov/docs/cctbx/doc_hklviewer/.

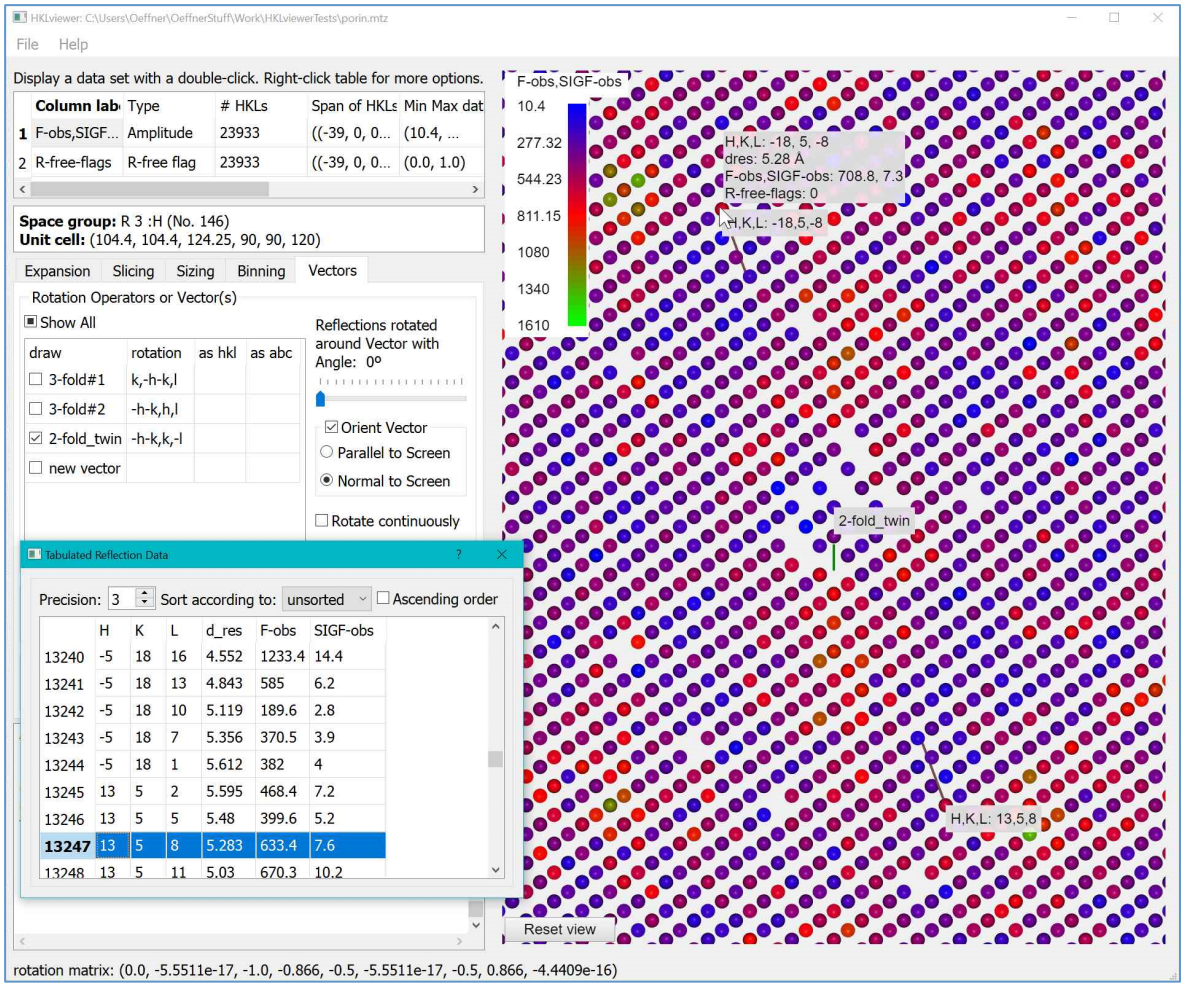


Figure 11: Reflection and its twin highlighted by tooltips. The twin axis is specified on the "Vectors" tab as the 2-fold_twin.

Works Cited

[1] R. W. Große-Kunstleve, "Algorithms for deriving crystallographic space-group information," *Acta Cryst.,* pp. A55, 383-395, 1999.

[2] N. Echols and P. D. Adams, "A lightweight, versatile framework for visualizing reciprocal-space data," *Computational Crystallography Newsletter,* vol. 2, pp. 88-92, 2011.

[3] P. Evans and E. Krissinel, "ViewHKL," 2011. [Online]. Available: http://legacy.ccp4.ac.uk/newsletters/newsletter48/articles/ViewHKL/viewhkl.html.

[4] I. J. Tickle, C. Flensburg, P. Keller, W. Paciorek, A. Sharff, C. Vonrhein and G. Bricogne, "STARANISO," Cambridge, United Kingdom: Global Phasing Ltd., 2018. [Online]. Available: http://staraniso.globalphasing.org/cgi-bin/staraniso.cgi.

[5] M. D. Sammito, *Phaser.voyager,* Manuscript in preparation.

[6] A. J. McCoy, D. H. Stockwell, M. D. Sammito, R. D. Oeffner, K. S. Hatti, T. I. Croll and R. J. Read, "Phasertng: directed acyclic graphs for crystallographic phasing," *Acta Crystallographica Section D,* vol. 77, pp. 1-10, 2021.

[7] The Qt company, "Qt documentation," [Online]. Available: https://doc.qt.io/.

[8] "ZeroMQ," 2020. [Online]. Available: https://zeromq.org/.

[9] Wikipedia, "macOS Mojave," [Online]. Available: https://en.wikipedia.org/wiki/MacOS_Mojave.

[10] A. S. Rose, R. A. Bradley, Y. Valasatava, M. J. Duarte, A. Prlić and W. P. Rose, "NGL viewer: web-based molecular graphics for large complexes," *Bioinformatics,* vol. 34, no. 21, p. 3755–3758, 2018.

[11] D. Sehnal, R. Svobodova, K. Berka, A. S. Rose, S. K. Burley, S. Velankar and J. Koca, "High-performance macromolecular data delivery and visualization for the web," *Acta Crystallographica Section D,* vol. 76, pp. 1167-1173, 2020.

[12] Wikipedia, "WebSocket," [Online]. Available: https://en.wikipedia.org/wiki/WebSocket.

Running CCTBX and PyMOL in the same Jupyter Notebook

Blaine H.M. Mooers[a,b,c]

[a]Department of Biochemistry and Molecular Biology, University of Oklahoma Health Sciences Center, Oklahoma City, OK 73104
[b]Stephenson Cancer Center, University of Oklahoma Health Sciences Center, Oklahoma City, OK 73104
[c]Laboratory of Biomolecular Structure and Function, University of Oklahoma Health Sciences Center, Oklahoma City, OK 73104

Correspondence email: blaine-mooers@ouhsc.edu

Introduction

The Jupyter Notebook is a popular coding platform (Kluyver *et al*., 2016). There are 9.7 million notebooks on GitHub as of December 13, 2020[1]. The notebooks are written in JavaScript and run in a web browser. The notebook is a front-end to a kernel that does the computing. The default kernel is a Python interpreter. Kernels for scores of other programming languages are available. These include compiled programming languages. For example, the xeus-cling kernel uses a just-in-time (JIT) compiler for C++ code (https://github.com/jupyter-xeus/xeus-cling). As a result, C++ code can be developed interactively in a Jupyter notebook.

The user of a notebook can interleave explanatory text, code, and output in tables or plots. Both the code and its output are often visible in the browser without needing to scroll down. The proximity of both code and output provides a sensation of instant feedback. The user can change the source of input data or a model's parameters and rerun the code. This interactivity facilitates the rapid development of the code. The interleaved blocks of text also make the notebooks useful as tutorials. Several inquiries about running CCTBX in Jupyter Notebooks have been posted on CCTBX's bulletin board (Grosse-Kunstleve *et al*., 2002).

PyMOL is also accessible from Jupyter through a recently released Python API (Application Programming Interface) (Delano, 2002). This API makes it easier to interleave code for CCTBX and PyMOL in one notebook. The API also eliminates the need to open the PyMOL GUI, although the user may still want to optimize the molecular object's orientation manually in open GUI session of PyMOL. The PyMOL's Python API eases combining code and output from CCTBX and PyMOL in one document. PyMOL complements CCTBX by providing stunning images of molecular objects. For example, PyMOL could be used to illustrate the results of coordinate manipulations computed with CCTBX. The ability to run both PyMOL and CCTBX in Jupyter Notebooks is highly desirable if these notebooks are used for literate programming with CCTBX.

This contribution demonstrates the use of CCTBX and PyMOL in the same Jupyter Notebook. We provide several alternate protocols for installing Jupyter, CCTBX, and PyMOL. The protocols include the installation of JupyterLab. The latter is a browser-based IDE (Integrated Development Environment) for editing and running Jupyter Notebooks. JupyterLab has an extension that supports access to code snippets via pulldown menus. We developed snippet libraries for CCTBX and PyMOL for use in JupyterLab. These libraries are easy to install and extend. We also show how to use shortcuts for PyMOL from the script pymolshortcuts.py in Jupyter Notebooks (Mooers 2020). Users can use the shortcuts and snippets together to enhance their productivity with PyMOL. Anyone interested in using Jupyter for computational structural biology should be interested in this article.

Methods

We tested the installation protocols on Ubuntu 20.04 LTS, Mac OS 10.15.7, and Windows 10 with the incentive and open-

[1]https://github.com/parente/nbestimate

source versions of PyMOL. We used JupyterLab version 2.2.0 because later versions of JupyterLab were not compatible with the **jupyterlab-snippets-multimenus** extension.

The snippets are code-fragments in Python script files. We used the PyMOL Python API directly or passed PyMOL command language (pml) code as arguments to the pymol module's **cmd.do()** method. The code in the snippets contains no extra formatting. The **jupyterlabcctbxsnips** library has over 70 snippets, and the **jupyterlabpymolpysnips** library has over 260 snippets. The 'plus' variants of both libraries include commented lines. These lines contain a second copy of the code. This second copy has sites to be considered for editing marked with a leading dollar sign and enclosed with a pair of curly braces to draw attention to them. The markup format follows the standard format used for tab stops in the snippet libraries for the Sublime Text 3 text editor.

Users can download the libraries from GitHub. Advanced users may not need the plus variants of the libraries. The names of the libraries in the table below are as they appear on the menu toolbar in JupyterLab. The URLs for the four libraries are listed in the table below.

Associated GitHub Pages list the snippets by category
https://mooerslab.github.io/jupyterlabcctbxsnips/
and
https://mooerslab.github.io/jupyterlabpymolpysnips/.

Results

In the next section, we describe the protocols for installing PyMOL, CCTBX, Jupyter, and snippet libraries. Next, we show how to access the snippets in JupyterLab. Then, we show examples of using CCTBX and PyMOL in the same Jupyter Notebook. We demonstrate the use of several PyMOL shortcuts in a Jupyter Notebook. The snippets and shortcuts improve productivity, self-directed learning, and documentation of computational work when using Jupyter.

Five or more alternative approaches can be used to install CCTBX, PyMOL, and Jupyter. The descriptions of these protocols take over 2000 words. To save space, we stored the protocols in the README.md file at https://github.com/MooersLab/jupyterlabcctbxsnips. Several protocols use the open-source version of PyMOL. Each protocol leads to the ability to run CCTBX and PyMOL in the same Jupyter Notebook. The GUI of JupyterLab provides pulldown menus for access to the libraries of Python code fragments for running CCTBX and PyMOL.

The examples below are included along with several others in a Jupyter Notebook that is available at the above repository. A static html version of this notebook also can be downloaded from the GitHub site by clicking on the file name, displaying the raw code, and saving the file without the appended txt extension added by the browser. The notebook also can be viewed with the easy to install *nteract* application (https://nteract.io).

Library name	GitHub repository
ccbtx	**https://github.com/MooersLab/jupyterlabcctbxsnips**,
cctbx+	**https://github.com/MooersLab/jupyterlabpymolpysnipsplus**
pymol	**https://github.com/MooersLab/jupyterlabpymolpysnips**
pymol+	**https://github.com/MooersLab/jupyterlabpymolpysnipsplus**

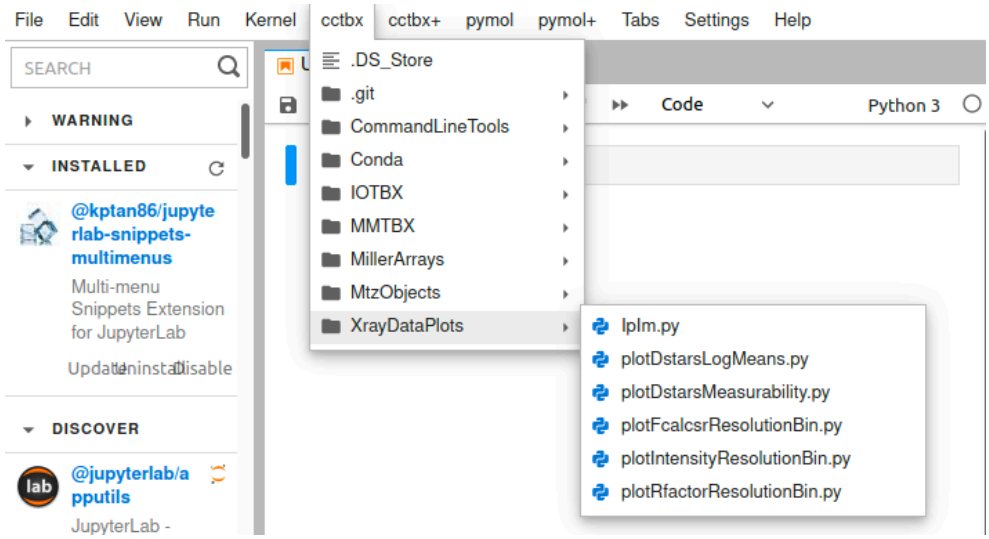


Figure 1. Code snippets accessed from cascading menus in JupyterLab.

The code cells will run after PyMOL and CCTBX and the required kernel have been installed following the instructions for Jupyter. However, *nteract* does not deploy the snippet libraries reported here.

Figure 1 shows a Jupyter Notebook opened JupyterLab after success with installing the software. The menu bar has four new menus (**cctbx**, **cctbx+**, **pymol**, and **pymol+)** listed between the **Kernel** and **Tabs** menus. The **cctbx** menu leads to a cascade of submenus. The XrayDataPlots submenu leads to a list of several snippets files.

The *IpIm.py* snippet was selected by left-clicking on it with the mouse cursor. The corresponding code inserts into the active notebook cell (**Figure 2**). The code is run by entering shift-enter (or return). The output appears below the active cell.

We used the **iotbx** module of CCTBX to read a mtz file from disk into a Miller array data structure (this mtz file can be downloaded from the data folder on the GitHub site). We made a scatterplot of I(+) vs I(-) with Matplotlib (Hunter, 2007). The anomalous signal in the data is proportional to the deviations from the x=y line. Reuse this snippet with a new dataset by editing the marked sites.

Figure 3 demonstrates running PyMOL from inside a Jupyter notebook. The **cmd.do()** method is used to send the PyMOL macro language (pml) commands to PyMOL. This method takes the PyMOL command language (pml) as its argument. Semicolons concatenate several PyMOL commands to save space. A set of single or double quotes enclose the concatenated commands.

It is not yet possible to have PyMOL's interactive viewport embedded in a Jupyter cell. Instead, the nglview package can provide an interactive view of the molecule. The problem with this approach is that PyMOL and nglview do not yet have an interface to pass molecular objects between each other inside the notebook. Instead, the coordinates have to be loaded into nglview from the disk. Nglview does have the advantage of allowing the loading and interactive display of molecular dynamics trajectories. Such interactive views are useful for workshops and lectures.

However, such interactivity is often not desired with preparing images for publication. It is easy to reissue commands that change the molecular object's orientation (e.g., translate, rotate, turn, orient), save an image, and reload

[12]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.ticker as ticker
from matplotlib.ticker import MultipleLocator #, FormatStrFormatter
from matplotlib.ticker import FuncFormatter
from iotbx.reflection_file_reader import any_reflection_file

# >>> change the mtz file name
hkl_file = any_reflection_file('3hz7.mtz')
miller_arrays = hkl_file.as_miller_arrays(merge_equivalents=False)
Iobs = miller_arrays[1]
i_plus, i_minus = Iobs.hemispheres_acentrics()
ipd = i_plus.data()
ip=list(ipd)
imd = i_minus.data()
im = list(imd)
len(im)

comma_fmt = FuncFormatter(lambda x, p: format(int(x), ','))

mpl.rcParams['savefig.dpi'] = 600
mpl.rcParams['figure.dpi'] = 600

# Set to width of a one column on a two-column page.
# May want to adjust settings for a slide.
fig, ax = plt.subplots(figsize=[3.25, 3.25])
ax.scatter(ip,im,c='k',alpha=0.3,s=5.5)
ax.set_xlabel(r'I(+)',fontsize=12)
ax.set_ylabel(r'I(-)',fontsize=12)
ax.xaxis.set_major_locator(MultipleLocator(50000.))
ax.yaxis.set_major_locator(MultipleLocator(50000.))
ax.get_xaxis().set_major_formatter(comma_fmt)
ax.get_yaxis().set_major_formatter(comma_fmt)

plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
ax.grid(False)

# >>> change name of the figure file
plt.savefig('3hz7IpIm.pdf',bbox_inches='tight')
```

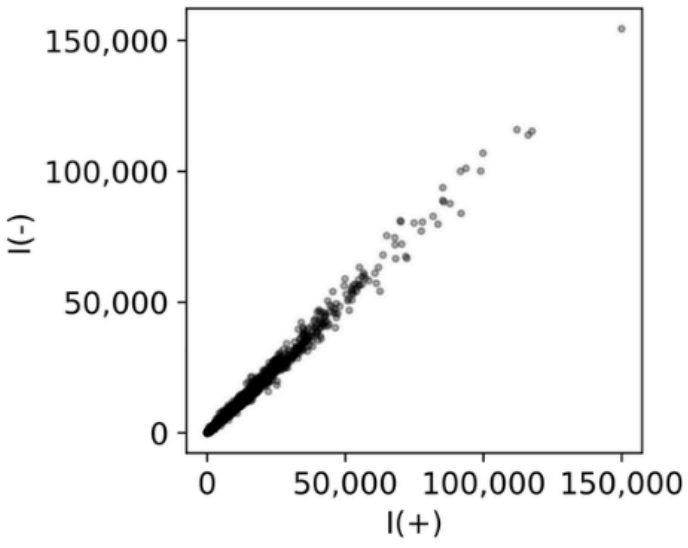


Figure 2. Example of snippet from the cctbx library.

[1]:
```
from pymol import cmd
from IPython.display import Image

PATH="/Users/blaine/"
cmd.do('run /Users/blaine/Scripts/\
PyMOLScripts/pymolshortcuts.py');
```

•••

The three horizonal dots above are the folded background information that is printed below the cell when the pymolshortcuts.py script is run.

[2]:
```
cmd.do("rein;bg_color white;U8;")
cmd.do("hide nonbonded;AOD;")
cmd.png("3nd4.png",
        width=300,
        height=450,
        dpi=600)
Image(filename = PATH + "3nd4.png",
      width=300,
      height=450,
      unconfined=True)
```

[2]:

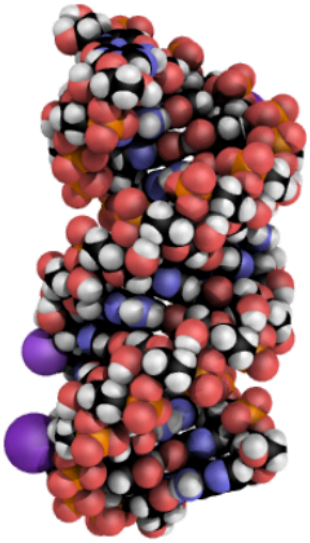


Figure 3. Two snippets from the pymol library and use of the *pymolshortcut.py* script.

the image. Often only three to five iterations of commands to reorient the molecule lead to the desire orientation. Zooming in and out is accomplished by translations along the z-axis. Alternately, the molecular object is loaded into PyMOL, manually oriented with the mouse,

and the **get_view** command retrieves the 18 parameter values that set the view. The **roundview()** function or **rv** shortcut (see below) returns these settings on a single line in a more compact format that is easier to use (Mooers, 2020).

The **cmd** class has other methods beyond the **do** method. For example, the **cmd.png()** method takes arguments like a Python function. Here, **cmd.png() saves** to disk a png file of an RNA duplex. The Image class from IPython loads the png file into the notebook and displays it below Cell [2].

The RNA duplex is displayed with the black carbon atom variant of the ambient occlusion effect, which is not available in PyMOL. The **AOD**() function in the *pymolshortcuts.py* file generates this effect (https://github.com/MooersLab/pymolshortcuts). This script is loaded in Cell [1] (**Figure 3**) with the **run** command. This line can be added to the *.pymolrc* file to load the shortcuts upon starting the PyMOL GUI. However, the pymol API in Jupyter starts up without reading the pymolrc file, so placing the run command in the pymolrc will not load the shortcuts into Jupyter on startup. Instead, the run command has to be given as an argument to the cmd.do() as in Cell [1]. The *importPyMOLandShortcuts.py* snippet inserts in Cell [1]. This snippet is found in the **Jupyter** submenu under the **pymol** menu. The path to *pymolshortcuts.py* will have to be edited in either the snippet or in the Jupyter cell. Other Python scripts are loaded in PyMOL with the same syntax.

Entering the **AOD** shortcut at the PyMOL prompt in the GUI invokes the black carbon atom variant of the ambient occlusion effect. On line 2 of Cell[2], we include **AOD** with another pml command as the argument for **cmd.do()**. The related **AO** shortcut colors the carbon atoms light grey. **AODBW** and **AOBW** color all atoms in grayscale. PyMOL lacks grayscale coloring, but it is available from the *pymolshortcuts.py* file. Grayscale figures are often required for book chapters. The **rein** shortcut runs the **reinitialize** command. The **U8** shortcut fetches PDB file 3nd4, generates the biological unit, and orients the molecular object with the helical axis aligned along the view's y-axis. The shortcuts save time and space.

All of the 250+ shortcuts in *pymolshortcuts.py* are listed below Cell [1] when the script is loaded. We hid this list by folding it. The folding operation is accomplished by left clicking on the transient blue bar that appears to the left of the active cell (not shown in Figure 3). Clicking on the three dots displays the list again. This list is not displayed when each shortcut is used. Instead, detailed documentation for a specific shortcut is displayed by using Python's help function (e.g., **help("AO")**). The documentation includes examples of usage for non-expert Python users. The documentation also includes the code in the snippet in pml and in Python. The pml code is also displayed with all of the statements concatenated on one line to ease pasting the commands at the PyMOL prompt in the PyMOL GUI. The list of shortcuts can be printed again elsewhere in the notebook by entering **cmd.do("sc")**. Users can find more information about the shortcuts at https://github.com/MooersLab/pymolshortcuts/ and in Mooers (2020).

We generated the above examples inside a conda environment that used that same Python interpreter to install CCTBX and PyMOL, so only one kernel is needed to run both packages. There is no need to switch kernels between cells, and both packages can be accessed from the same cell. The default Python 3 kernel is sufficient. At this time, molecular objects cannot be passed directly between PyMOL and CCTBX. Instead, coordinates are written to disk from one package and loaded from disk by the other package.

Discussion

We draw attention to the ability to run CCTBX and PyMOL side-by-side in Jupyter Notebooks. The user can interleave code and output from CCTBX and PyMOL. Code snippets and shortcuts described above enhance the efficiency of assembling the interleaved code. These snippets are accessible from cascading pulldown menus at the top of the JupyterLab GUI. The jupyterlab-snippet-multimenus extension provides this access. This extension is easy to install and use. Users can create new cascading sub-menus by making new subfolders. The snippet files require no special formatting. The PyMOL code has to be written in Python rather than the PyMOL command language (pml). We provide starter snippet libraries for PyMOL and CCTBX that are easy to install for use with the jupyterlab-snippet-multimenus extension. Below, we discuss limitations on the use of snippets in Jupyter. We also discuss several uses for Jupyter Notebooks in structural studies.

Several extensions for JupyterLab support the use of snippets. All of these extensions do not support tab triggers and tab stops at this time. These two missing features further enhance the efficiency of using snippets and are normally available in text editors and IDEs. We address the lack of tab stops by providing "+" variants of the libraries. These variants have a commented out copy of the code that has the tab stops marked with dollar signs and curly braces. The marked sites draw attention to locations in the snippet that should be considered for editing. These guides compensate for the absence of tab stops in the snippets.

The Jupyter Notebook can store the image-generating code and the images for a structure project in a single document. Usually, many images fill multi-panel figures in a manuscript. Often, authors change the images between successive drafts of the manuscript. The number of code and image files grows large as a manuscript matures. The image and the code that generated it can be found by scrolling through the notebook. This scrolling is often faster than searching through the files in dozens of subfolders. Furthermore, the need often arises to remake the images months after a manuscript is accepted for publication. The user may need to make variations of the existing images for journal cover artwork, news releases, posters, platform talks, seminars, lectures, lab webpages, review articles, book chapters, and framed wall hangings. The presence of the code in one notebook eases finding, modifying, and reusing the code.

The storage of code and output in one notebook simplifies the sharing of the code and the images. Only one document needs to be sent electronically if the notebook does not require local data files. If the code blocks in the notebook are left exposed, collaborators can edit and run the modified code. Jupyter notebooks ease collaborations by reducing the number of shared files.

If the collaborators are not experts in CCTBX or PyMOL, the code blocks can be hidden to reduce the clutter. The shared document can be instead by a non-interactive HTML file or a pdf generated from the Jupyter Notebook. The JupyterLab GUI supports exporting a notebook to alternate formats. Alternately, the **nbconvert** command in the terminal converts a notebook to one of many formats. (The **nbconvert** command is installed when Jupyter is installed.) Static formats of Jupyter notebooks can be opened without using Jupyter; this makes the static formats easier to share with collaborators who do not use Jupyter.

Past and current versions of the four libraries are archived with Zendo.org and are open to download. The provided **jupyterlabcctbxsnips** library is a version 0.2 prototype library. It may inspire users to create snippets. We plan to add more

snippets, and we welcome contributed snippets. Users can add snippets via the Issues tab on the project's GitHub page or send them by email to the author.

References

DeLano, W. L. (2002). PyMOL 0.99.

Grosse-Kunstleve, R. W., Sauter, N. K., Moriarty, N. W. & Adams, P. D. (2002). *J. Appl. Cryst.* **35**, 126–136.

Hunter, J. D.(2007) *Comput. Sci. Eng.* **9**, 90-95.

Kluyver, T., Ragan-Kelley, B., Fernando Perez, Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P. Avila, D., Abdalla, S., & Willing, C. (2016). Jupyter Notebooks – a publishing format for reproducible computational workflows. In F. Loizides & B. Schmidt (Eds.), *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (pp. 87–90).

Mooers, B. H. M. (2020). *Protein Sci.* **29**, 268-276. doi: 10.1002/pro.3781.