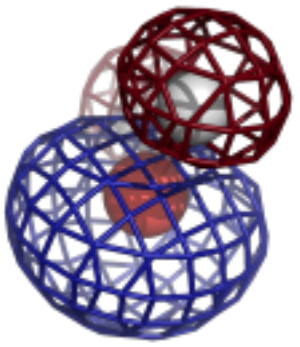
VOLUME THREE



COMPUTATIONAL CRYSTALLOGRAPHY NEWSLETTER

JULY MMXII

QUEUING, ALTERNATIVE LOCATIONS, AM1/RM1

Table of Contents

Editor

Nigel W. Moriarty, NWMoriarty@LBL.Gov

PHENIX News

New releases

Model-building

New beta-testing releases in model-building include phenix.combine_models and a trace_chain option for phenix.fit_loops. The combine_models method will take two overlapping models and a map and try to identify the best parts of each model based on the map. It then creates a single new model. The method can be used for models that are assigned to sequence as well as those that are not. The trace_chain option for loop fitting uses a fast chain-tracing algorithm to find a path between two ends of a loop and then builds a loop based on that chain.

New features

New autosol features

Classical density modification is available in the autosol program using the command mask_type="classic". Autosol (versions 1078 and higher) now has a "derscale=xxx" keyword that allows you to fix the scale factor for derivative datasets (useful for rip phasing where the scale factor can be critical for structure solution).

Anisotropy corrections and sharpening

phenix.autosol and phenix.autobuild now automatically apply an anisotropy correction and sharpening for all density-modified map calculations. At the end of these methods a final model is produced that is refined against the original (uncorrected) data. This model is written out as overall_best.pdb. The uncorrected data for refinement is written out as overall_best_refine_data.mtz. You can adjust how this works by specifying a target overall B-value if you want (e.g., b_iso=25) , and you can also turn it off if you like (remove_aniso=False).

Ensemble output options in phaser.MRage.solutions

Ensemble models, which contain structural information from a series of homologous templates, can be very powerful in molecular replacement. However, after molecular replacement has succeeded, information content of the solution is reduced, since due to limitations of the PDB format, the ensemble model is normally replaced by the best model in the collection. Novel output options have been added to phaser.MRage.solutions that allow the preservation of the previously discarded information content in part or in full. (It is important to note that the output map is not affected in any way, and is always calculated with the full information content of the selected solution.) The list of output options are as follows (available via the --output switch): - single (default). This just select the first model from the ensemble. - best-single-model. This calculates the LLG of each model and selects the one that gives the highest LLG. The best scoring model of one ensemble is selected independently for each copies of the ensemble, and can therefore vary if there are significant differences between the NCS-related molecules. - combined. This option uses phenix.combine_models to combine all components of the ensemble model into a "chimera" model, based on their fit to the electron density. As for the previous option, this is also done independently for each copies of the ensemble, and the procedure can yield slightly different results for NCS-related molecules. - ensemble. Outputs all components of the ensemble model in one file. Altloc identifiers are assigned to each component of the ensemble.

Crystallographic meetings and workshops

Gordon Research Conference on Diffraction Methods in Structural Biology, Bates College, Lewiston, ME, July 15-20, 2012

Jeff Headd will be presenting at the Gordon Conference on Diffraction Methods in the “Getting the Best Out of Your Data: Data Analysis” section.

The 27th Meeting of the European Crystallographic Association, Bergen, Norway, August 6-11, 2012.

Pavel Afonine will give the plenary and receive the Fifth Erwin Felix Lewy Bertaut Prize of the European Crystallographic Association (ECA) and European Neutron Scattering Association (ENSA).

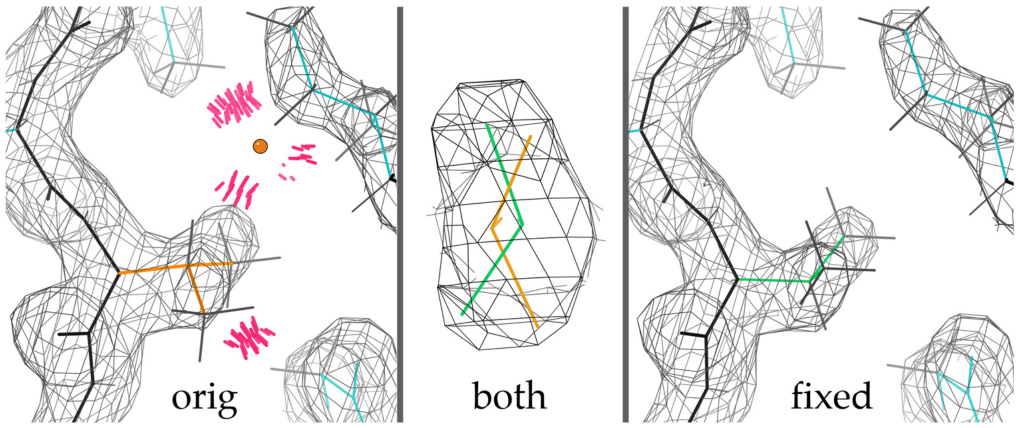
Expert advice

Fitting Tip - Rotamer correction, with backrubs

Jane Richardson, Duke University; Jeff Headd, LBL

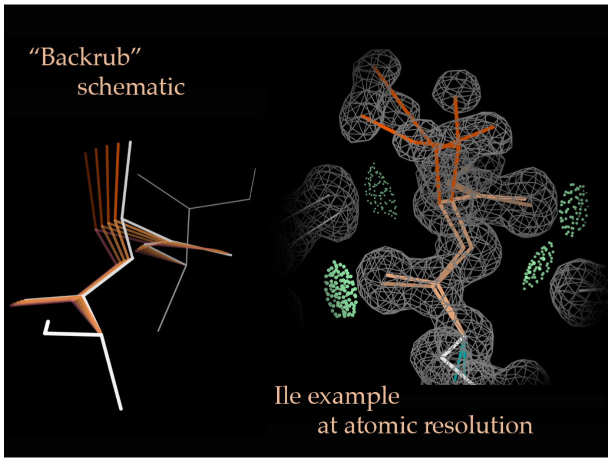
Backward-fit `Asn/Gln/His` aminoacid side-chains (including His protonation - see Tip CCN, January, 2012) can be corrected essentially without affecting the match of model to data. However, sidechains with tetrahedral geometry (`Thr`, `Val`, `Ile`, `Leu`, and even `Arg`) may also be modeled backwards into elongated or flat density, and those corrections must be done along with a refinement process, either integrated or as rebuilding (e.g. in Coot or in KiNG) alternated with refinement.

The figure on the next page shows an example (`Val 218 in 1LPL`) where the density does not clearly indicate directionality of the tetrahedral branch, and the wrong choice was modeled (in gold). A problem is diagnosed by the doubly-eclipsed rotamer outlier and the serious all-atom clashes. The imaginary water is also suspicious, and such cases distort bond



angles, often enough to be 4σ outliers although not here. If you know that such misfits are a common systematic error, you will see immediately how to make the correction, which is easy and satisfying to do. The Cβ position and sidechain geometry should first be idealized; then the χ1 value changed about 180° to align with the density from the other direction (for a backward `Leu` or `Arg`, 2 χ angles must be changed). This will usually place the sidechain parallel to but somewhat outside the density, so it must be shifted in by a manual fit or by real-space refinement. The result (green), as replaced in `1TOV` (Arendall 2005) occupies more-or-less the same space, fits the density a bit better, and corrects all of the validation outliers.

The most effective motion for achieving the shift of sidechain sidewise into density is the "backrub" (Davis 2006), a simple rotation around an axis between the i-1 and i+1 Cα's that has a good lever-arm for shifting the Cβ in a direction perpendicular to the backbone. The second figure (below) shows a schematic of the backrub motion and an example of how this same motion is used dynamically to relate two clear alternate conformations seen at high resolution. You can get a good feeling for how the backrub motion works by rebuilding backward-fit sidechains in KiNG, where it is implemented as a separately controllable feature under the "fit rotamer" tool. Try `Thr` 77 in `1BKR`, for instance, or `Ile` A 120 in `1MOO`. Then you will understand better how backrubs are helping the real-space rotamer rebuilding in Coot or in other automated procedures, as well as being able to do them yourself when that's needed. In Phenix, backrubs have now been added to enable even better rotamer correction in the torsion-NCS procedure, and in future they will also be added to the real-space rotamer fit option within phenix.refine.



References

Davis IW, Arendall WB III, Richardson JS, Richardson DC (2006). "The Backrub Motion: How Protein Backbone Shrugs When a Sidechain Dances." *Structure* **14**: 265-274.

Arendall WB III, Tempel W, Richardson JS et al. (2005). "A test of enhancing model accuracy in high-throughput crystallography." *Journal of Structural & Functional Genomics* 6, 1-11.

Contributors

P. D. Adams, P. V. Afonine, G. Bunkóczi, F. C. Chou, R. Das, N. Echols, J. J. Headd, N. W. Moriarty, J. S. Richardson, N. K. Sauter, O. V. Sobolev, T. C. Terwilliger, A. Urzhumtsev

FAQ

How can I use semi-empirical quantum chemical methods to optimise the geometry of my ligand?

There is an option in the ligand restraints building module in PHENIX known as eLBOW (Moriarty, Grosse-Kunstleve & Adams, 2009) the can do this for you. The option is `--opt` and will minimise the energy of geometry for the ligand. The minimised geometry is reflected in the ideal restraints written to disk.

It was originally decided to use the Austin Model 1 (AM1) with the implementation being released in 2007. It soon became clear that the reparameterisation for a subset of elements (H, C, N, O, P, S, F, Cl, Br, and I) known as Recife Model 1 (RM1) was easily implemented on top of the AM1 method and performed better for the smaller set of elements. For more information on the implementation of RM1 in eLBOW see the reference and for more information of the RM1 method see Rocha et al., 2006.

In addition to the semi-empirical methods implemented in eLBOW, external quantum chemistry packages can be used to optimise the ligand geometry. For example, with the GAMESS package installed, adding the `--gamess` option will use the AM1 method in GAMESS. Furthermore, the other higher-level quantum mechanical methods are available via the `--method` and `--basis` options.

References

Moriarty, N. W., R. W. Grosse-Kunstleve, et al. (2009). "electronic Ligand Builder and Optimization Workbench (eLBOW): a tool for ligand coordinate and restraint generation." Acta Crystallographica Section D **65**: 1074-1080.

Rocha, G. B., R. O. Freire, et al. (2006). "RM1: A Reparameterization of AM1 for H, C, N, O, P, S, F, Cl, Br and I." J. Comput. Chem. **27**: 1101-1111.

DETAC: tools to detect alternative conformations by unrestrained refinement.

Oleg V. Sobolev

Institute of Mathematical Problems of Biology, Russian Academy of Sciences, Pushchino, Moscow region, 142290, Russia

Correspondence email: sobolev@impb.psn.ru

Introduction

Unrestrained refinement is stable for the vast majority of atoms when working at atomic resolution. Nevertheless geometrical restraints should be used in refinement for residues that are present in several (alternative) conformations in the crystal used for the X-ray experiment, otherwise residue geometries deteriorate significantly. We believe that large distortions of the residue geometries in unrestrained refinement may hint at the presence of alternative conformations for this residue.

To get these hints in a routine way we suggest two methods to analyze shifts of atomic centers resulted from several cycles of unrestrained refinement. A simple diagram plotting the values of atomic shifts against the residue number may give an idea of the crystallographic order of different parts of the structure at qualitative level (Sobolev & Lunin, 2008). To put the analysis on a more quantitative basis several decision-making procedures were developed and tested which compose a list of residues that are likely to be present in alternative conformations or disordered. These residues should be checked thoroughly with Fourier syntheses and included into the model with alternative conformations, when necessary (Sobolev & Lunin, 2012).

To study the mobility of atoms in unrestrained refinement and derive parameters for decision-making procedures we studied 203 structures from PDB with resolution of experimental data better than 1.2Å and R-work better than 0.13. We took care in setting the limits for R-factor values of the models to select high-quality structures in which the alternative conformations were assigned reliably. The following steps were applied sequentially to each selected model:

1. The alternative conformation possessing the largest occupancy was left in the model with the occupancy set to 1.
2. Hydrogen atoms were included.
3. The standard restrained refinement was performed for the model with `phenix.refine` using anisotropic ADP and riding hydrogens.
4. Three macro-cycles of unrestrained refinement were performed with `phenix.refine`.
5. Calculation of atomic shifts between model after step 3 and 4.

The performed analysis of atomic shifts values revealed significant difference between the atoms in alternative and single conformations. Moreover, side chain atoms usually get larger shifts than main chain atoms, and atoms at the surface of the molecule get larger shifts than atoms inside the globule. At the average, shifts are increasing with decrease of data resolution and R-factor growth. These differences provide the possibility of distinguishing atom types by the study of their atomic shifts in unrestrained refinement.

TUR-routine

To analyze a particular model, we suggest the application of the Trial Unrestrained Refinement (TUR). The TUR should consist of three macro-cycles of unrestrained refinement by `phenix.refine` with anisotropic ADP and riding hydrogens. The crystal should diffract to approximately 1.2Å or better and the model should be refined rather well (to the R-factor around 0.15 or better). The following two programs (`shift_plot` and `ac_prediction`) may be

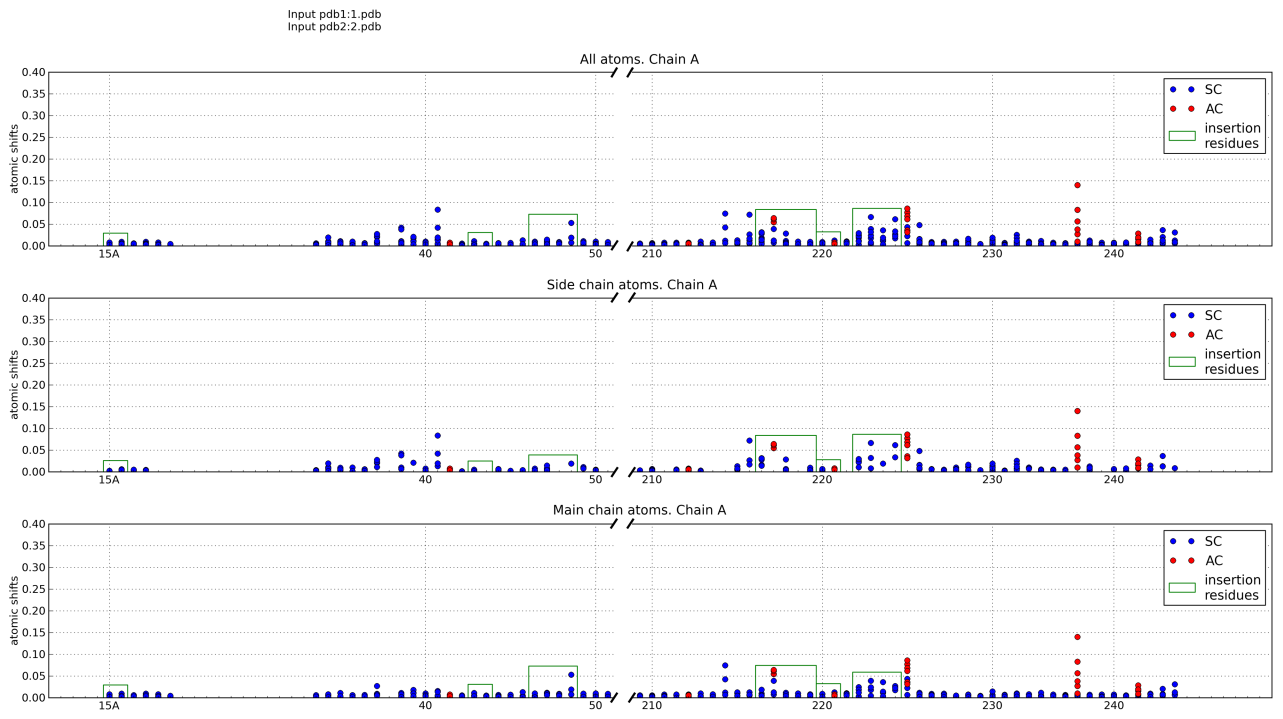


Figure 1: Fragment of diagram of atomic shifts produced by shift_plot for 1SSX structure (Fuhrmann *et.al.*, 2004).

used to analyze the results of the TUR.

Shift_plot

`Shift_plot` displays the results of TUR as several diagrams presenting atomic shifts vs. residue number (Fig.1). In these diagrams each dot corresponds to a non-hydrogen atom and the dots corresponding to the same residue are grouped in a column. High columns reveal residues that had significant atomic shifts in unrestrained refinement and thus are unstable and suspected to be present in alternative conformations. `Shift_plot` produces three diagrams: (i) for all atoms, (ii) for side chain atoms and (iii) for main chain atoms. The diagrams hint at which residues may have alternative conformations, but the proper choice of cut-off level to select high columns is not always obvious. These diagrams may be used at any resolution and do not depend on the refinement program.

Atoms that were in alternative conformations in input PDB files are plotted as red circles. Some PDB files contain residues with equal residue number but different insertion code (column 27 of PDB files). Such residues will be plotted as separate columns in the diagrams but these columns will be marked by green rectangle that shows that residues in it have the same residue number.

ac_prediction

`ac_prediction` implements automatic decision-making procedure that classifies every residue as 'single conformation' or 'alternative conformations' (SC and AC below). The current version of `ac_prediction` works with atomic-resolution data and unrestrained refinement conducted by `phenix.refine`. The procedure is based on the observation that SC-residues and AC-residues have different mobility in unrestrained refinement. The decision is made either by comparing the observed atomic shifts with a predetermined threshold, or comparing the probabilities of such shifts for SC and AC-residues. The thresholds and probability distributions of atomic shifts were calculated during analysis of unrestrained refinement of 203 atomic-resolution structures from PDB.

To run the program, two PDB files with models before TUR and after TUR should be provided along with resolution and R-factor values of the model before TUR. `ac_prediction` produces a list of residues that should be checked first with electron-density maps as main candidates for introducing alternative conformations.

It should be noted that, `ac_prediction` does not guarantee the correctness of the decision. It provides a user with sorted list of residues for which the presence of alternative conformations is the most probable. The testing of `ac_prediction` proved that prediction based on atomic shifts is more accurate than prediction based on ADP or density values in atomic centers in '2mFo-DFc' syntheses (Sobolev & Lunin, 2012).

Availability

The programs are available for free download from the following address:
http://www.impb.ru/lmc/programs/ac_prediction/

Questions, comments and bug reports are welcome.

Acknowledgments

The author is grateful to Prof. Vladimir Lunin for useful discussions. This work was partially supported by grant 10-04-00254a of Russian Foundation for Basic Research.

References

Fuhrmann C.N., Kelch B.A., Ota N., Agard D.A. (2004). *The 0.83 Å resolution crystal structure of alpha-lytic protease reveals the detailed structure of the active site and identifies a source of conformational strain.* J. Mol. Biol. **338**. 999-1013.

Sobolev O. V. & Lunin V. Y. (2008). *Unrestrained reciprocal space refinement of macromolecular structures as a tool to indicate alternative conformations*. Mathematical Biology and Bioinformatics, **3**(2), 50-59 (in Russian). URL: http://www.matbio.org/downloads/Sobolev2008(3_50).pdf

Sobolev O.V. & Lunin V.Y. (2012). *Detection of alternative conformations by unrestrained refinement*. Acta Cryst., D**68** (in press).

ERRASER, a powerful new system for correcting RNA models

Fang-Chieh Chou[a], Jane Richardson[b] and Rhiju Das[a]

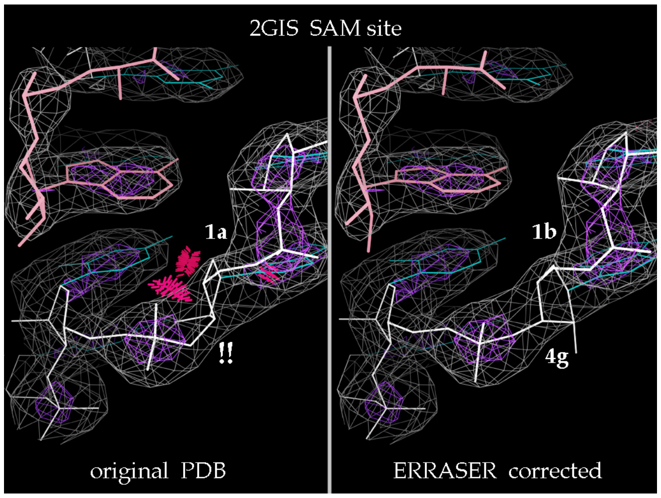
[a]*Stanford University, Stanford, CA*

[b]*Duke University, Durham, NC*

Correspondence email: rhiju@stanford.edu

PHENIX collaborators Rhiju Das and Fang Chou at Stanford have combined their full-enumeration local search called Step-Wise Assembly, or SWA, (Sripakdeevong 2010) with the Rosetta RNA energy function (Das 2010), Rosetta electron-density score (DiMaio 2009), MolProbity diagnosis, and Phenix refinement into an automated procedure called Erraser. The Duke team has evaluated this as being by far the most effective method yet for correcting fitting errors in RNA structures, especially aimed at the difficult backbone conformation and ribose pucker. ERASSER starts with refinement in PHENIX (`phenix.refine`), including backbone-specific RNA target parameters (Adams 2010), followed by Rosetta "relax" minimization (Leaver-Fay 2011). It then runs the `phenix.rna_validate` MolProbity-style validation (Chen 2010) to identify nucleotides with problems in ribose pucker or suite conformers (Richardson 2008), all-atom clashes, or covalent geometry. Those residues are then put through the exhaustive SWA sampling protocol to look for better conformations that match the density at least as well. SWA and Rosetta relax are cycled three times, and those results considered clear improvements are given a last round of refinement.

As reported in the on-line preprint (arXiv:1110.0276v2) of their paper (Chou 2012), for the 24-structure test set *R-free* improves and all-atom clashes, dubious ribose puckers, and poor backbone conformers are all reduced on average by factors of 2 to 8, while bond lengths and angles can then remain outlier-free in refinement. This works even at lower resolution -- for instance, the clashscores at 3.2-2.7Å resolution drop to values typical in PDB RNAs at 1.8Å. Some of the independently validated improvements are at critical binding or active sites. An example of coupled local correction of ribose pucker, conformers, and clashes is shown in the figure, at the `SAM-I` binding site of the `2GIS` riboswitch (Montange 2006).



ERRASER's success depends on its exhaustive enumeration of the possibilities, so that it is somewhat compute-intensive -- but not unreasonably so because the sampling is purely local. The user needs to install Rosetta as well as PHENIX. The scripts for running both the PHENIX refinements and the ERRASER cycles in Rosetta are included in the supplementary material for the in-press paper. ERRASER has also been integrated into PHENIX, and the users can run it under command line through the "`phenix.erraser`" application.

References

Adams PD, *et al.* (2010) PHENIX: a comprehensive Python-based system for macromolecular structure solution, *Acta Crystallogr* **D66**: 213-221.

Chen VB, *et al.* (2010) MolProbity: all-atom structure validation for macromolecular crystallography, *Acta Crystallogr* **D66**: 12-21.

Chou F-C, Sripakdeevong P, Dibrov SM, Hermann T, Das R (2012) Correcting pervaseive errors in RNA crystallography through enumerative structure prediction, *Nature Methods* (under revision).

Das, R., Karanicolas, J., Baker., D (2010) Atomic accuracy in predicting and designing noncanonical RNA structure, *Nature Methods* **7**:291-294.

DiMaio F, Tyka MD, Baker ML, Chiu W, Baker D (2009) Refinement of Protein Structures into Low-Resolution Density Maps Using Rosetta, *J Mol Biol* **392**: 181-190.

Leaver-Fay A, *et al*. (2011) ROSETTA3: an object-oriented software suite for the simulation and design of macromolecules, *Methods Enzymol* **487**: 545-574.

Montange RK, Batey RT (2006) Structure of the S-adenosylmethionine riboswitch regulatory mRNA element, Nature 441: 1172-1175.

Richardson JS, *et al*. (2008) RNA backbone: Consensus all-angle conformers and modular string nomenclature (an RNA Ontology Consortium contribution). *RNA* **14**: 465-481.

Sripakdeevong P, Kladwang W, Das R (2010) An enumerative stepwise ansatz enables atomic-accuracy RNA loop modeling, *Proc Nat Acad Sci USA* **108**: 20573-20578.

cctbx tools for transparent job execution on clusters

Gábor Bunkóczi[a] and Nathaniel Echols[b]

[a]*Department of Haematology, University of Cambridge, Cambridge, UK*
[b]*Lawrence Berkeley National Laboratory, Berkeley, CA*

Correspondence email: gb360@cam.ac.uk

Introduction

With the availability of multi-core CPUs, parallelization is becoming increasingly common in software development for speeding up execution. Specialized supporting tools and libraries have been made available for various programming languages to help developers in writing parallel code. The Python programming language (http://www.python.org) provides the `threading` and `multiprocessing` modules in the Python Standard Library (http://docs.python.org/library/index.html), which allow parallel execution of calculations on a separate thread or process, respectively. These modules have proved very useful in shared-memory-based parallelization, but cannot harness the computational power provided by clusters without specialized code. For certain tasks in scientific computing, access to CPUs in clusters would boost productivity considerably, but due to the inherent heterogeneity in mechanisms of access and job control, it is not possible (or not practical) to maintain specialized code that would allow universal deployment. The current work describes a component that allows transparent access to an underlying submission queue, and enables execution in a unified way. It is available with the `libtbx` module of the Computational Crystallography Toolbox (`cctbx`, http://cctbx.sourceforge.net; Grosse-Kunstleve *et al.*, 2002), the open-source component of the *PHENIX* project (http://www.phenix-online.org; Adams *et al.*, 2010).

Goals

The `threading` and `multiprocessing` modules in the Python Standard Library provide the `Thread` and `Process` classes, respectively, to execute code concurrently with the main thread. Both of these classes share a common interface, and therefore code designed to use one of them would run with the other correctly. Providing a `Job` class that shares the same interface with the `Thread` and `Process` classes, and acts as a drop-in replacement, but submits the calculation to a queuing system, would enable a suitably written program to run seamlessly on clusters.

In addition, the need for another component arises from the fact that both `Thread` and `Process` rely on data channels to send the result of the computation back to the main thread (Figure 1). Such channels are provided by e.g. the `Queue.Queue` and `multiprocessing.Queue` classes. `Queue.Queue` is memory-based and is primarily useful in conjunction with `Thread`, because the data cannot cross a process boundary, while `multiprocessing.Queue` uses a (file or network) socket that allows inter-process data exchange, and could also be used with `Process`. Unfortunately, even the latter would not be suitable for the proposed `Job` class, because execution in general would take place on a different host, and makes the development of a novel `Queue` necessary.

It is clear at this stage that a fair amount of temporary data has to be written to the current directory. This needs to be done in a way that concurrent instances of the same program running in the same directory does not result in a race condition. As a non-functional goal, the number and size of temporary files should be kept at the necessary minimum.

Constraints

A completely general implementation of such functionality may be possible, but beyond available resources. For practical reasons, the only function calls supported are those that can be serialized with the `pickle` module. This includes all pure Python objects, Boost.Python (Abrahams & Grosse-Kunstleve, 2003) exported objects with serialization support, and named functions

```
from multiprocessing import Process, Queue

def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = Queue()
    p = Process(target=f, args=(q,))
    p.start()
    print q.get()    # prints "[42, None, 'hello']"
    p.join()
```

Figure 1 Example demonstrating typical use of the `multiprocessing` module (taken from http://docs.python.org/library/multiprocessing.html). Note that the result is returned via a `multiprocessing.Queue` instance that is passed to the call as an argument.

(http://docs.python.org/library/pickle.html#what-can-be-pickled-and-unpickled). Notable exceptions are `lambda`-functions and functions defined in the interactive interpreter.

Design

Analyzing the format of job submission commands and available options on queuing system we have access to, i.e. Sun Grid Engine (SGE, http://en.wikipedia.org/wiki/Oracle_Grid_Engine), Load Sharing Facility (LSF, http://en.wikipedia.org/wiki/Load_Sharing_Facility) and Portable Batch System (PBS, http://en.wikipedia.org/wiki/Portable_Batch_System), allowed us to formulate common and varying properties that can be exploited in the design:

- Command-line options are rarely conserved, but control similar behavior. These could be encapsulated so that the actual switches are hidden, but the behavior is exposed.
- All systems capture `stdout` and `stderr` from the process in files. Default naming convention varies, but it is possible to assign a name via command-line options.
- All systems provide a command-line option to change the default directory to the current one.
- The studied systems accept commands from `stdin`, i.e. rendering an initial (temporary) command file unnecessary. Unfortunately, this is not universally valid, e.g. when using Condor (http://en.wikipedia.org/wiki/Condor_High-Throughput_Computing_System), a command file and a Condor control file are required, and the documentation suggests that the command file has to be an actual file on the file system. This restriction, however, is specific to Condor, and it would not be productive to not take advantage of the read from `stdin` feature when it is available. Therefore, this part should be factored out in a component that has to be replaced for systems that do not support it, e.g. Condor.
- Passing input data to the job can be done by either writing out a temporary file, or by incorporating this data in the command file. The most efficient method depends on the size of input.
- SGE and LSF support synchronized mode (in addition to the more common asynchronous one) that has certain benefits when checking whether a job has completed or not (polling). This could be made available for queuing systems that support it by suitable encapsulation.

Since queuing systems all differ, actual details have to be passed to the `Job` constructor at instantiation. However, this changes the interface from that of `Thread` and `Process`, and makes it unsuitable for drop-in replacement. This can be solved by storing details of the queuing system in a handler class that has a method with an interface identical to `Thread` or `Process`. Varying execution details are then passed onto the `Job` constructor by the handler. The final design of the `processing` class is shown in Figure 2.

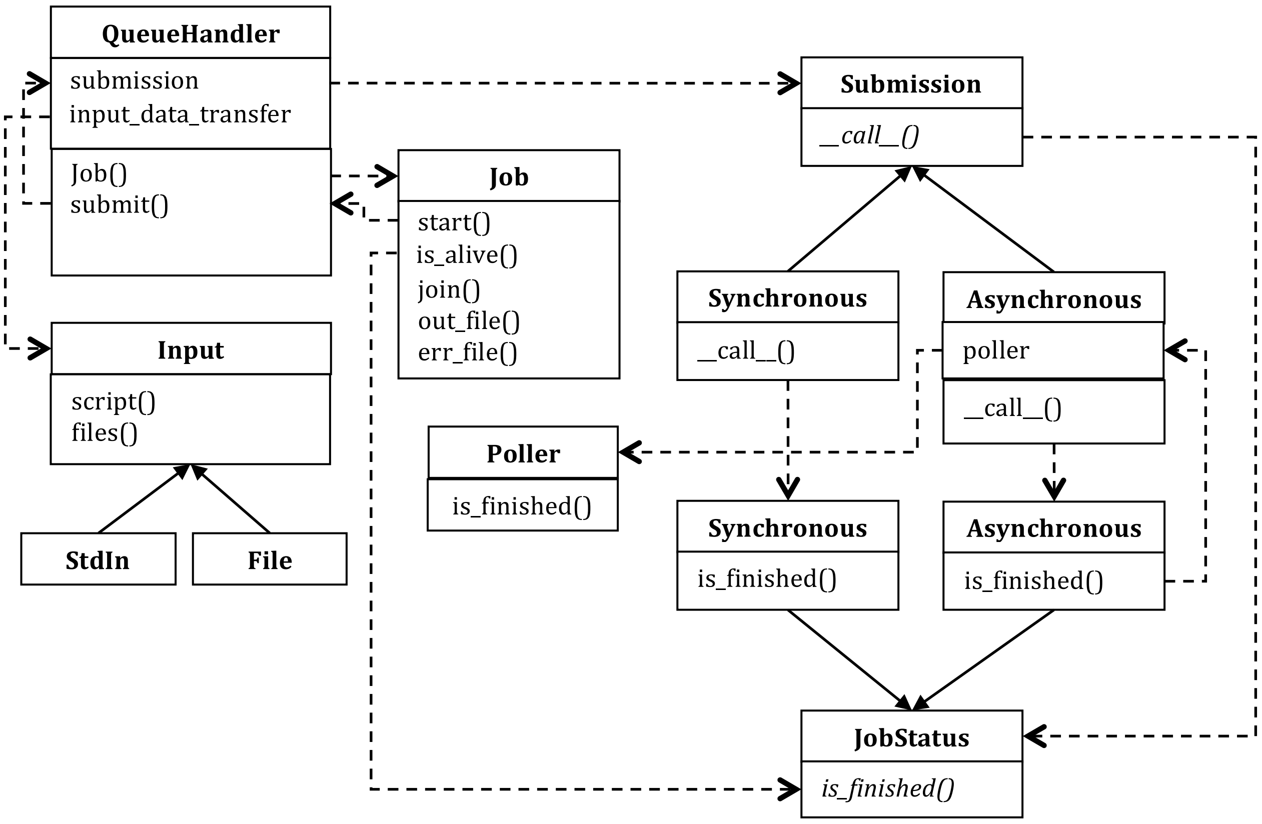


Figure 2 UML-style class diagram of components provided by the `processing` module.

Factoring the `Poller` out to a separate entity deserves an explanation. Polling for asynchronous jobs can be very slow, because it requires a system call to the appropriate queuing system command. This design allows a global `Poller` shared by all jobs, which requires less frequent updates than individual jobs would poll.

The data channel `processing.Queue` is implemented using temporary files that are placed in the current directory (that is assumed to be on a file system shared between the hosts). It mimics the interface of `Queue.Queue` and `multiprocessing.Queue`. Each instance is assigned a unique label, consisting of a root name, which is provided as a convenience for the user to know which program has created the temporary file, the process ID, and the object identifier (as returned by the Python `id` function); the latter two ensures the label is unique. This is used as the root of the file names. Temporary files are then named using the unique label and a sequence number, so that the channel could return multiple data, although this is not normally necessary. While this simple implementation suffices for most practical uses, this class is not functionally equivalent to `Queue.Queue` or `multiprocessing.Queue`, since those are multi-producer, multi-consumer channels, while `processing.Queue` is strictly single-producer and single-consumer. In addition, due to latency associated with network file systems, a generous timeout has to be allowed. Therefore, for functional and performance reasons, `processing.Queue` is not a generic replacement for `Queue.Queue` or `multiprocessing.Queue`, but rather should be employed when necessary, i.e. with the `processing.Job` class. In certain cases, a higher performance replacement is possible. MPI (message-passing interface) provides network-based inter-host communication that could be exploited. In addition, network-based channels between hosts can also be used in favorable cases, if there are no firewalls blocking access to incoming connections.

```
from libtbx.queuing_system_utils import processing

def f(q):
    q.put([42, None, 'hello'])

if __name__ == '__main__':
    q = processing.Queue( identifier = "tmp" )
    sge = processing.SGE()
    p = sge.Job(target=f, args=(q,))
    p.start()
    print q.get()    # prints "[42, None, 'hello']"
    p.join()
```

Figure 3 Example code from Figure 1 rewritten to run on SGE.

Coding

The details of the implementation in Figure 2 may appear daunting. However, only few applications will need to exploit the full flexibility provided by the design. For regular use, convenience constructors are provided for the queuing systems that are supported. These constructors also allow incorporation of user configuration, e.g. using a dedicated command for submitting with certain options, or extra parameters to direct the job to a certain queue. A simple example is shown in Figure 3.

Scheduling

Queuing systems manage the submitted jobs and balance the load over the allocated hosts, and make sure those are not overloaded. This is a very convenient feature that would often be useful for parallel programs. The `multiprocessing.Pool` class offers very similar functionality, but it is limited to handle `multiprocessing.Process` instances, and also provides no way for checking how many of the assigned CPUs are idle. Furthermore, in some workflows incoming results can make certain calculations unnecessary, and in this case it is important not to commit a calculation when there are no resources for execution. Therefore, we have implemented such functionality as a new module, `libtbx.scheduling`, so that it can be used independently from the `processing` module.

The class diagram is shown in Figure 4. `ExecutionUnit` encapsulates the actual execution mode. It holds a factory function that creates an execution class instance (`Thread`, `Process` or `QueueHandler.Job`). `Manager` objects hold and distribute the jobs onto `ExecutionUnits`. `Manager` provides a simple interface to submit jobs, checks jobs in various phases of execution (waiting, running, finished), and iterate through the results as they are produced or any particular order. A separate `Adaptor` class is provided to give access to the same `Manager` from various parts of the program, but keeping the jobs "private", so that a job submitted through an `Adaptor` is not accessible from other `Adaptor` instances, even if they use the same `Manager`. This allows more efficient design than by simply allocating resources to various parts of the program, because a single global resource can distribute load until all CPUs are busy, while with isolated `Managers` free CPUs from one instance cannot be reallocated to another.

The class diagram in Figure 4 suggests that the `Manager` allows `ExecutionUnits` with various types of execution classes to be used. However, when a job is submitted, the system will not know how the calculation will get executed, i.e. which execution class will be instantiated, because it is not predictable which `ExecutionUnit` will first complete the calculation it is currently running. Since a data channel has to be passed along with the calculation to return the result, and the type of the data channel depends on the execution class, this presents a problem. One solution would be to use the most general data

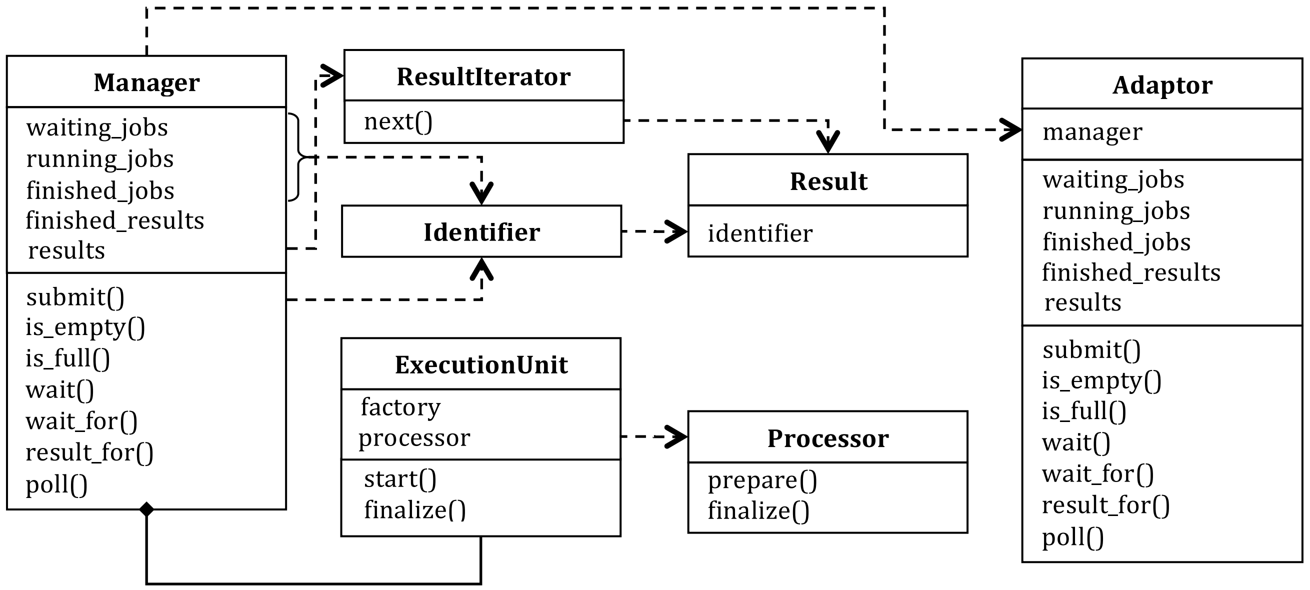


Figure 4 Class diagram of the `scheduling` module.

channel, i.e. the file-based `processing.Queue` that is implemented for `processing.Job`. However, this would represent a performance sink when `processing.Job` is not used (which may not be possible to know until run time). A better solution is to assign the appropriate data channel to the `ExecutionUnit`. This is possible using a suitable `Processor` class that bundles the call with the data channel, and extracts the results after the calculation has finished. Such functionality is available in the `RetrieveProcessor` class, which is provided with the `scheduling` module. Using a `RetrieveProcessor` may be slightly more efficient even when only uniform execution classes are used, because it allows the data channels to be reused. An example is shown in Figure 5.

By using a `Manager`, it is possible to detach execution from program logic entirely. This is desirable, because the client code need not be aware, how the result is obtained. After instantiation, the `Manager` object can be used uniformly, without reference to the number and type of execution classes. In addition, it is also possible to use main-thread-only (non-parallel) execution, with little performance overhead. In principle, using a single `Thread` or `Process` class would be an acceptable solution, but a slightly more efficient way is to define an "execution factory function" that performs the function call, and avoids job startup overhead.

Current uses

The molecular replacement pipeline `phaser.MRage` is based on a `scheduling.Manager` for handling parallel calculations. Depending on the molecular replacement search, `phaser.MRage` can use a large number of CPUs, and therefore accessing clusters can boost performance substantially. Encapsulating the execution mode was a major factor that allowed enabling additional execution modes incrementally. Currently, besides the Python standard library modules `multiprocessing` and `threading`, SGE, LSF and PBS are supported, but additional queuing systems can be added on request.

Acknowledgements

The first version of this module was based on `libtbx.queuing_system_utils.lsf`, written by Joel Bard. We thank Robert Nolte and Herbert Klei for suggestions.

```
from libtbx.queuing_system_utils import scheduling
import multiprocessing
import threading
import Queue

def f(value):
    return value

manager = scheduling.Manager(
    units = (
        [ scheduling.ExecutionUnit(
            factory = multiprocessing.Process,
            processor = scheduling.RetrieveProcessor(
                queue = multiprocessing.Queue(),
                ),
            )
        for i in range(2) ]
        + [ scheduling.ExecutionUnit(
            factory = threading.Thread,
            processor = scheduling.RetrieveProcessor(
                queue = Queue.Queue(),
                ),
            )
        for i in range(3) ]
        )
    )
adapter1 = scheduling.Adapter( manager = manager )
adapter2 = scheduling.Adapter( manager = manager )

for i in range(10):
  adapter1.submit( target = f, args = (i,) )

for i in range(10, 20):
  adapter2.submit( target = f, args = (i,) )

# after a short delay: print numbers from 10 to 19 in some order
# (adapter1 jobs are processed first)
print [ i.result for i in adapter2.results ]

# immediately: print numbers from 0 to 9 in some order
# (these jobs have already been completed)
print [ i.result for i in adapter1.results ]
```

Figure 5 Example code demonstrating the `scheduling` module.

References

Abrahams, D. & Grosse-Kunstleve, R. W. (2003). *C/C++ Users Journal* **21**, 29-36.

Adams, P. D., Afonine, P. V., Bunkoczi, G., Chen, V. B., Davis, I. W., Echols, N., Headd, J. J., Hung, L. W., Kapral, G. J., Grosse-Kunstleve, R. W., McCoy, A. J., Moriarty, N. W., Oeffner, R., Read, R. J., Richardson, D. C., Richardson, J. S., Terwilliger, T. C. & Zwart, P. H. (2010). *Acta Crystallogr D* **66**, 213-221.

Grosse-Kunstleve, R. W., Sauter, N. K., Moriarty, N. W. & Adams, P. D. (2002). *Journal of Applied Crystallography* **35**, 126-136.

On the analysis of residual density distributions on an absolute scale

Pavel V. Afonine[1], Alexandre Urzhumtsev[2,3] and Paul D. Adams[1,4]

[1]*Lawrence Berkeley National Laboratory, One Cyclotron Road, BLDG 64R0121, Berkeley, CA 94720 USA*
[2]*IGBMC, CNRS-INSERM-UdS, 1 rue Laurent Fries, B.P.10142, 67404 Illkirch, France*
[3]*Université de Lorraine, Physics Department, Vandoeuvre-lès-Nancy, 54506 France*
[4]*Department of Bioengineering, University of California Berkeley, Berkeley, CA, 94720, USA*

Correspondence e-mail: pafonine@lbl.gov, sacha@igbmc.fr

Preamble

This newsletter article was inspired by figure 1 in the B.W. Matthews' paper (Matthews, 2009). The figure shows averaged residual density distributions as a function of distance from an atom. Unexpectedly there is substantial negative density in the vicinity of the atomic centers and substantial positive density in the bulk-solvent region. This became a subject of our analysis described below.

Calibrating expectations

Analysis of electron density on an absolute scale (i.e. in units of e/Å^3) requires the value of F000 reflection and the normalization (division) of Fourier transform of the corresponding map coefficients by the unit cell volume. While F000 is never measured directly in the diffraction experiment, it can be estimated. One way is to measure the crystal average density $\rho_{average}$ and then estimate

$$F000 \approx \rho_{average} V_{cell} \qquad (1)$$

where V_{cell} is the unit cell volume. If the macromolecular structure is already known, then another alternative is to calculate it as

$$F000 = F000_{calc} + F000_{bulk} \qquad (2)$$

where $F000_{calc}$ is the sum of the scattering factors at zero diffraction angle calculated over all atoms in the unit cell (which includes the molecule itself and all the symmetry mates) and $F000_{bulk}$ is the bulk-solvent contribution $k_{sol}V_{sol} = k_{sol}V_{cell}x$, where k_{sol} and *x* are the average density of the bulk-solvent and its volume fraction, correspondingly. This obviously requires knowing k_{sol}, which can be estimated as described in Fokine & Urzhumtsev (2002) and Afonine *et al.* (2005), though may not be adequate (Afonine *et al.*, 2012, in preparation). Yet another approach relies on the assumption that residual electron density around reliably placed atoms should be zero, which in turn defines

$$F000 \approx -\rho_{average,\ selected\ atoms} V_{cell} + F000_{calc} \qquad (3)$$

where $\rho_{average,\ selected\ atoms}$ is the average density around reliably placed atoms obtained using a Fourier synthesis calculated without F000.

To test the applicability of formula (3) we performed the following numerical experiment. For four selected structures from the PDB (Berman *et al.*, 2000), `2x8h`, `3ahs`, `3m8u` and `3krr`, we simulated experimental data as $F_{obs} = |\mathbf{F}_{model}| = k_{total}|\mathbf{F}_{calc} + \mathbf{F}_{bulk}|$, where $\mathbf{F}_{bulk}$ was computed using a flat bulk-solvent model with an average density of 0.35 e/Å^3. These calculations were performed using the `phenix.fmodel` program (Afonine, unpublished). For each test case two sets of data were simulated: one is a 100% complete set up to the highest measured resolution (F^{f}_{obs}) and the second set (F^{i}_{obs}) corresponds to the set of Miller indices of actually measured reflections, which may be incomplete. Next we calculated four Fourier syntheses on an absolute scale with ($\Delta\rho_1$, $\Delta\rho_2$, $\Delta\rho_4$), and without ($\Delta\rho_3$), accounting for F000:

$$\Delta\rho_1 = \{F^{f}_{obs}/k_{total},\ \varphi_{model} - F_{calc},\ \varphi_{calc}\ ;\ \Delta F000\}$$

$$\Delta\rho_2 = \{F^{f}_{obs}/k_{total} - F_{calc},\ \varphi_{calc}\ ;\ \Delta F000\}$$

$$\Delta\rho_3 = \{F^{f}_{obs}/k_{total} - F_{calc},\ \varphi_{calc}\}$$

$$\Delta\rho_4 = \{F^{i}_{obs}/k_{total} - F_{calc},\ \varphi_{calc}\ ;\ \Delta F000\}$$

and plotted averaged density values calculated as described by Matthews (2009) (figure 1). Here

$$\Delta F000 = F000_{obs} - F000_{model}$$
$$= F000_{bulk}$$
$$\approx -\rho_{average,\ selected\ atoms} V_{cell}.$$

The first synthesis ($\Delta\rho_1$) is obviously expected to show the distribution of bulk-solvent: nearly zero density levels around atomic centers, constant density in the bulk solvent region with a level of 0.35 e/Å^3, and a smooth range of values at the solvent-macromolecule boundary. Figure 1 confirms this expectation for all four structures.

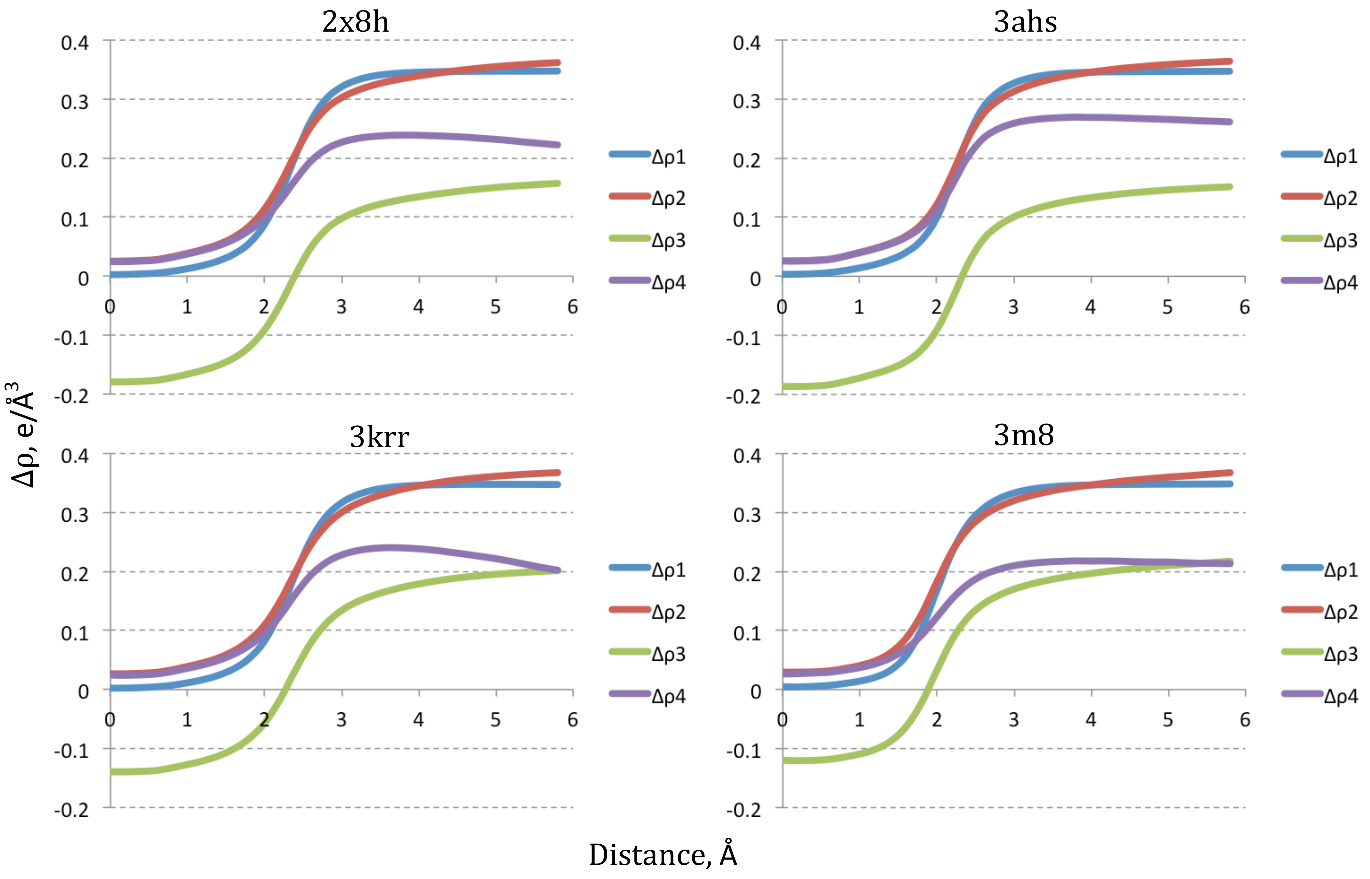


Figure 1. Average residual electron density as a function of the distance from the nearest atom (for details, see Matthews, 2009). Simulated data; see text for notations.

The second synthesis ($\Delta\rho_2$) differs from $\Delta\rho_1$ in that it is calculated with less accurate phases, corresponding to the atomic model only. This is expected to disturb the synthesis in some way but perhaps not significantly. Indeed figure 1 shows some positive density values in the vicinity of atoms and a not perfectly constant (0.35 e/Å^3) level of density in the solvent region away from the macromolecule.

The third synthesis ($\Delta\rho_3$) is equivalent to $\Delta\rho_2$ but calculated without ΔF000 term. The shape of this synthesis is very similar to that of $\Delta\rho_2$ but the values are shifted by a constant, resulting in substantially negative density around atoms and reduced density values in the bulk-solvent region (figure 1).

Finally the fourth synthesis ($\Delta\rho_4$) is equivalent to $\Delta\rho_2$ but calculated using the incomplete set F^{i}_{obs}. The density distribution is expected to be further distorted compared to $\Delta\rho_1$ and $\Delta\rho_2$ (figure 1), and the degree of distortion is a function of the number and type of missing reflections. The analysis of how missing reflections affect this distribution is beyond the scope of this article.

Results

As we now know how the density distribution in question appears in ideal or near-to-ideal settings we can meaningfully analyze the analogous syntheses calculated using real measured data, F_{obs} (as opposed to simulated data used above). For this we calculated two syntheses for each of four models:

$$\Delta\rho_5 = \{F_{obs}/k_{total} - F_{calc}, \varphi_{calc} ; \Delta F000\}$$

$$\Delta\rho_6 = \{F_{obs}/k_{total} - F_{model}, \varphi_{model} ; \Delta F000\}$$

and plotted averaged density values calculated as described by Matthews (2009) (figure 2). Here $\Delta F000 = -\rho_{average,\ selected\ atoms}V_{cell}$ and is not necessarily equal to $F000_{bulk}$ but also accounts for other missing scattering in the model.

The bulk-solvent-omit synthesis ($\Delta\rho_5$) overall

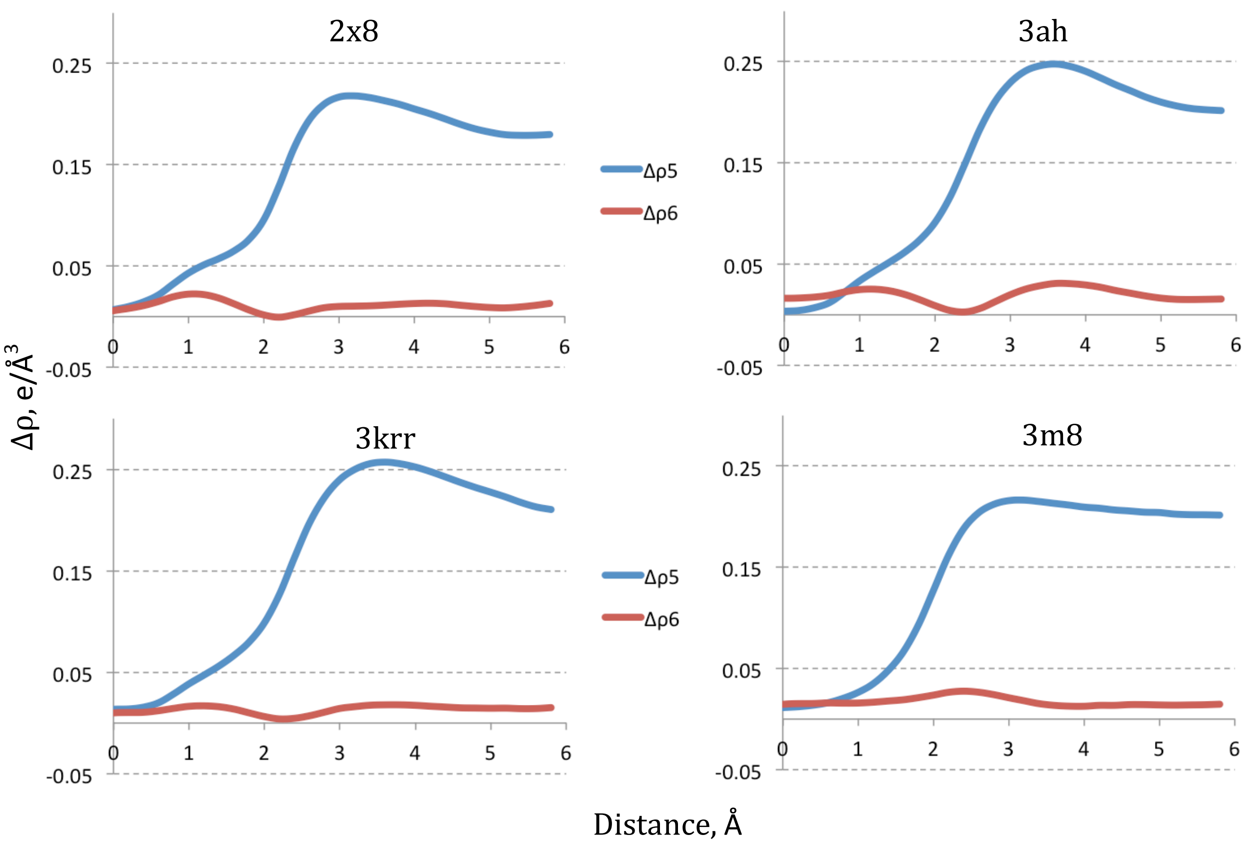


Figure 2. Average residual electron density as a function of the distance from the nearest atom (for details, see Matthews, 2009). See text for notations.

resembles the expected curve (e.g. compare with $\Delta\rho_4$). The differences may be attributed to the errors in data and model parameters, model incompleteness (missing atoms), and deviation of the flat bulk-solvent model used to simulate F^i_{obs} from the real bulk-solvent distribution. One can postulate that a peak at approximately 3.5Å in the $\Delta\rho_5$ synthesis could arise from partially structured weakly bound and therefore unmodeled solvent. However, one can notice the same behavior (though less pronounced) in the $\Delta\rho_4$ syntheses for all four models, where no such partially structured solvent was included in the calculation of the data. Indeed, the peak positions in $\Delta\rho_5$ (3.6, 3.2, 3.2 and 3.6 Å) are quite similar to those in $\Delta\rho_4$ (3.8, 3.8, 3.8 and 3.6 Å) for 2x8h, 3ahs, 3m8u and 3krr correspondingly. This may be attributed to Fourier truncation artifacts, though a conclusive answer would require further analysis.

The nearly flat residual synthesis, $\Delta\rho_6$, with a density level significantly lower than the bulk-solvent value suggests that on average there is no systematically missing features left to model in the bulk-solvent region. However, since these density distributions are the average over all atoms and grid points of the map in the unit cell, they may not capture local features that are yet to be accounted for.

The residual density distribution shown in figure 1 in Matthews (2009) (red bars) closely resembles the $\Delta\rho_3$ distribution shown above, which suggests that the ΔF000 reflection was not fully accounted for, as would be required for map calculations on an absolute scale.

References

Afonine, P., Grosse-Kunstleve, R.W., & Adams, P.D. (2005). *Acta Cryst.*, **D61**, 850-855.

Berman, H.M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T.N., Weissig, H., Shindyalov, I.N. & Bourne, P.E. (2000). *Nucleic Acids Research*. **28**, 235-242.

Fokine, A & Urzhumtsev, A. (2002). *Acta Cryst.*, **D58**, 1387-1392.

Matthews, B.W. (2009). *Protein Sci.*, **18**, 1135-1138.