Xtaltools Reference Manual

David A Case

May 17, 2023

1 Overview of XtalTools

This section gives a bird's-eye view of capabilities of the *XtalTools* package. The hope is to get more detailed examples and documentation as time permits. The organization is based on the compuational tools required, which almost certainly not the best approach.

2 ccp4_scripts

These scripts are based on the *CCP4* and *refmac5* programs.

conf_solvent.sh Takes an electron density for the solvent (say from MD simulations or from integral equation models), plus an atomic model for the "solute" (generally a macromolecule like a protein), and refines the model with *refmac5*. As a comparison, ignores the input solvent density and uses the standard *refmac5* bulk solvent contribution, as would be done in a conventional simulation. This offers one way to see if the input density is any "better" than the standard solvent contribution.

md2diffuse.sh Takes snapshots from an MD simulation, and works through the steps to compute both the Bragg intensities and diffuse intensities. The *sfall* program is the key engine to compute scattering amplitudes. Special attention is paid to reducing disk space requirements, so that scattering amplitudes from many snapshots can be stored and analyzed. The code optionally will constuct map coefficients and an electron density map arising from the simulation.

md2map.sh

mdv2map.sh The two scripts compute the average electron density, as a CCP4 map and as map coefficients, from an input MD simulation. Unlike the **md2diffuse.sh** script, which computes the average scattering amplitudes, then uses an FFT to obtain the final map, these two scripts directly add the maps together, then uses a Fourier transform to obtain the map coefficients. Since no diffuse scattering information is needed, there is much less need to optimize disk space; still the overall amount of computation is about the same, *i.e.* that required to run *sfall* on each frame of the input trajectory. The **mdv2map.sh** script assumes that the unit cell dimensions don't change throughout the trajectory, whereas **md2map.sh** does not make this assumption.

3 phenix_scripts

XrayPrep

CryoPrep These two scripts take an atomic model (in the form of a PDB-format file), and either an electron density (as a CCP4 map) or structure factors (as an sf-cif file), and prepare all the files needed to carry out refinements that employ a molecular force field (rather than Engh-Huber like distance and angle restraints) to constrain the molecular geometry.

run_phenix.solvent This is a sample script to allow comparisons of the bulk-solvent model in *phenix* and a user-supplied solvent density. Unlike the **conf_solvent.sh** script based on *refmac5*, this only works for a single-point, and doesn't carry out refinements that include the user-supplied solvent density.

run_phenix_refine This is just an alternative way to run *phenix.refine* from the command line, with flexible ways to choose the input paramters. It's probably not for everyone.

run_fmodel This just runs *phenix.fmodel* and re-formats the output in a simple but occasionally useful way.

4 rism_scripts

run_rism A simple script to run 3D-rism calcualtions (using the *msander* program) on a single snapshot. Typical values for grid spacing, closure and solvent model are hard-wired here, but you should edit this if you wish

run_metatwist_rho Takes the output from *run_rism* and computes an electron density map for the solvent. This can be used to replace the "flat" bulk solvent density model in programs like *msander* or *refmac5*.

5 ensemble_scripts

These are scripts that are used to run ensemble refinements, where many complete copies of the macromolecule are present, generally arranged in a supercell. These are mainly useful as part of a bigger workflow, whose documentation is not yet ready.

find_alts output a listing of alternate conformations found in an input pdb-file

select_alts this is the main script, that takes an input pdb file with alternate conformations (for example, as found in the PDB). For each chain, it randomly selects an alternate conformer for each residue, based on its input occupation. The output file sets all occupations to 1.0. This is commonly used to create a starting model for ensemble refinement.

modify_pdb is a utility perl script for making all manner of modifications to PDB files, at least that can be done on a line-by-line basis. The scripts reads an input PDB file, makes whatever changes are coded in an intermediate section, and writes out the result.

modify_frac is a variant of modify_pdb that allows modification based on fractional coordinates, useful for repacking coordinates into a single unit cell, and so on.

collapse_pdb takes an ensemble model and converts it into a more standard "alternate conformer" model

6 Setting up crystal simulations

David S. Cerutti

Simulations of biomolecular crystals are in principle no different than any of the simulations that AMBER does in periodic boundary conditions. However, the setup of these systems is not trivial and probably cannot be accomplished with the LEaP software. Of principal importance are the construction of the solvent conditions (packing precise amounts of multiple solvent species into the simulation cell), and tailoring the unit cell dimensions to accommodate the inherently periodic nature of the system. The LEaP software, designed to construct simulations of molecules in solution, will overlay a pre-equilibrated solvent mask over the (biomolecular) solute, tile that mask throughout the simulation cell, and then prune solvent residues which clash with the solute. The result of this procedure is a system which will likely contract under constant pressure dynamics as the pruning process has left vacuum bubbles at the solute:solvent interface. Simulations of biomolecular crystals require that the simulation cell begin at a size corresponding to the crystallographic unit cell, and deviate very little from that size over the course of equilibration and onset of constant pressure dynamics. This demands a different strategy for placing solvent in the simulation cell. Four programs in the *AmberTools* release are designed to accomplish this. An example of their use in given in a web-based tutorial at *https://ambermd.org/tutorials/advanced/tutorial13/XtalTutor1.html* . A recent (2018) review of crystal simulations is also worth consulting.[**?**]

For brevity, only basic descriptions of the programs are given in this manual. All of the programs may be run with command line input; the input options to each program may be listed by running each program with no arguments.

6.1 UnitCell

A macromolecular crystal contains many repeating unit cells which stack like blocks in three dimensional space just as simulation cells do in periodic boundary conditions. Each unit cell, in turn, may contain multiple symmetry-related clusters of atoms. A PDB file contains one set of coordinates for the irreducible unit of the crystal, the "asymmetric unit," and also information about the crystal space group and unit cell dimensions. The *UnitCell* program reads PDB files, seeking the SMTRY records within the REMARKs to enumerate the rotation and translation operations which may be applied to the coordinates given in the PDB file to reconstruct one complete unit cell.

Usage of the *UnitCell* program is as follows. The simple command rests on a critical assumption, that the PDB file contains an accurate CRYST1 record and that the REMARK 290 SMTRY records provide its space group symmetry operations.

```
UnitCell -p MyProtein.pdb -o UnitCell.pdb
```

6.2 PropPDB

Simulations in periodic boundary conditions require a minimum unit cell size: the simulation cell must be able to enclose a sphere of at least the nonbonded direct space cutoff radius plus a small buffer region for nonbonded pairlist updates. Many biomolecular crystal unit cells come in "shoebox" dimensions that may have one very short side; many unit cells are also not rectangular but triclinic, meaning that the size of the largest sphere they can enclose is further reduced. The workhorse simulation engine, pmemd.cuda, even requires that the simulation cell be at least three times as thick as the cutoff plus some buffer margin in order to run safely: for typical sum conditions this thickness is about 30Å. For these reasons, and perhaps to ensure that the rigid symmetry imposed by periodic boundary conditions does not create artifacts (crystallographic unit cells are equivalent when averaged over all time and space, but are not necessarily identical at any given moment), it may be necessary to include multiple unit cells within the simulation cell. This is the purpose of the *PropPDB* program: to propagate a unit cell in one or more directions so that the complete simulation cell meets minimum size requirements.

Drawing on the hypothetical example above, if the unit cell is too small we can extend it in the x and z dimensions:

```
PropPDB -p UnitCell -o ExpandedCell.pdb -ix 2 -iy 1 -iz 2
```

6.3 AddToBox

The *AddToBox* program handles placement of solvent within a crystal unit cell or supercell (as may be created by PropPDB). As described in the introduction, the basic strategy is to place solvent such that added solvent

molecules do not clash with biomolecule solutes, but *may* clash with one another initially. This compromise is necessary because enough solvent must be added to the system to ensure that the correct unit cell dimensions are maintained in the long run, but it is not acceptable to place solvent within the interior of a biomolecule where it might not belong and never escape.

The *AddToBox* program takes a PDB file providing the coordinates of a complete biomolecular unit cell or supercell (argument -c), the dimensions by which that supercell repeats in space (the unit cell dimensions are taken from the CRYST1 record of this file), a PDB file describing the solvent residue to add (argument -a), and the number of copies of that solvent molecule to add (argument -na). *AddToBox* inherently assumes that the biomolecular unit cell it is initially presented may contain some amount of solvent already, and according to the AMBER convention of listing macromolecular solute atoms first and solvent last assumes that the first -P atoms in the file are the protein (or biomolecule). *AddToBox* will then color a very fine grid "black" if the grid point is within a certain distance of a biomolecular atom (argument -RP, default 5.0Å) or other solvent atom (argument -RW, default 1.0Å); the grid is "white" otherwise (the grid is stored in binary for memory efficiency). *AddToBox* will the make a copy of the solvent residue and randomly rotate and translate it somewhere within the unit cell. If all atoms of the solvent residue land on "white" grid voxels, the solvent molecule will become part of the system and the grid around the newly added solvent will be blacked out accordingly. If the solvent molecule cannot be placed, this process will be repeated until a million consecutive failures are encountered, at which point the program will terminate. If *AddToBox* has not placed the requested number of solvent molecules by the time it terminates, the -V option can be used to order the program to recursively call itself with progressively smaller solvent buffer distances until all the requested solvent can be placed. The output of the *AddToBox* program is another PDB named by the -o option.

Successful operation of *AddToBox* may take practice. If multiple solvent species are required, as is the case with heterogeneous crystallization solutions, *AddToBox* may be called repeatedly with each input molecular cell being the previous call's output. When considering crystal solvation, the order of addition is important! It is recommended that rare species, such as trace buffer reagents, be added first, with large -RW argument to ensure that they are dispersed throughout the available crystal void zones. Large solvent species such as MPD (an isohexane diol commonly used in crystallization conditions) should be added second, and with a sufficiently large -RW argument that methyl groups and ring systems cannot become interlocked (which will likely lead to SHAKE / vlimit errors). Small and abundant species such as water should be added last, as they can go anywhere that space remains.

Below is an example of the usage for a hypothetical protein with 5431 atoms and a net charge of +6 that is to be neutralized with ammonium sulfate:

```
AddToBox -c ExpandedCell.pdb -a Sulfate.pdb -na 18 -RP 3.0 -P 5431 -o System.pdb
AddToBox -c System.pdb -a Ammonium.pdb -na 30 -RP 3.0 -P 5431 -o System.pdb
AddToBox -c System.pdb -a Water.pdb -na 1089 -RP 3.0 -P 5431 -o System.pdb
```

The use of the -V flag ensures that the desired amounts of each species are included. The protein clipping radius of 3Å is lower than the default, but safe (remember, this radius stipulates that no solvent atom, regardless of the size of the solvent molecule, come within 3Å of the protein). Note how the original protein PDB file serves as the base for system, but thereafter we work with the System.pdb to accumulate more solvent particles. Here, the ammonium sulfate serves both to neutralize the system and replicate a salty bath, perhaps from a crystallization mother liquor, hence the break from the usual 2:1 stoichiometry of ammonium sulfate ions.

It is likely that the unobservable "void" regions between biomolecules in most crystals *do not* contain solvent species in proportion to their abundance in the crystallization solution–the vast majority of these regions are within a few Ångstroms of some biomolecular surface, and different biomolecular functional groups will preferentially interact with some types of solvent over others. Also, in many crystals some solvent molecules *are* observed; in many of these, the amount of solvent observed is such that it would be impossible to pack other species into the unit cell in proportion to their abundances in the crystallization fluid. In these cases, we recommend estimating the amount of volume that must be filled with solvent *apart from solvent which has already been observed in the crystal*, and filling this void with solvent in proportion to the composition of the crystallization fluid. For example, if a crystal were grown in a 1:1 mole-to-mole water/ethanol mixture, and the crystal coordinates as deposited in the PDB contained 500 water molecules and 3 ethanol molecules, we would use *AddToBox* to add water and ethanol in

a 1:1 ratio until the system contained enough solvent to maintain the correct volume during equilibrium dynamics at constant pressure.

Finally, it is difficult to estimate exactly how much solvent will be needed to maintain the correct equilibrium volume; the advisable approach is simply to make an initial guess and script the setup so that, over multiple runs and reconstructions, the correct system composition can be found. We recommend matching the equilibrium unit cell volume to within 0.3% to keep this simulation parameter within the error of most crystallographic measurements. While errors of 0.5-1% will show up quickly after constant pressure dynamics begin, a 10 to 20ns simulation may be needed to ensure that the correct equilibrium volume has been achieved.

6.4 ChBox

After the complex process of adding solvent, the LEaP program may be used to produce a topology and initial set of coordinates based on the PDB file produced by *AddToBox*. By using the *SetBox* command, LEaP will create a periodic system without adding any more solvent on its own. The only problem with using LEaP at this point is that the program will fail to realize that the system *does* tile in three dimensions if only the box dimensions are set properly. If visualized, the output of UnitCell / PropPDB will likely look jagged, but the output of *AddToBox*, containing lots of added water, will make it obvious how parts of biomolecules jutting out one face of the box fit neatly into open spaces on an opposite face. The topology produced by LEaP needs no editing; only the last line of the coordinates does. This can be done manually, but the *ChBox* program automates the process, taking the same coordinates supplied to AddToBox and grafting them into the input coordinates file.

The program is even unnecessary in the case of orthorhombic (rectangular) unit cells, as this the tleap command will substitute:

```
set [unit] box { <x> <y> <z> }
```

For cells that do not have only 90-degree box angles, *ChBox* will do the trick.