**Instructions for CrystalPredictor2.x**

For the basic concepts involved in the program CrystalPredictor, see the original paper:

“*Ab initio* crystal structure prediction – I. Rigid molecules”, P.G.Karamertzanis and C.C.Pantelides,

*J. Comput. Chem*, **26**, 2005, pp304-324.

And for further information on version 1, see:

“*Ab initio* crystal structure prediction II. Flexible molecules”, P.G.Karamertzanis and C.C.Pantelides,

*Mol. Phys*, **105**, 2007, pp273-291.

The major innovation in this new version of CrystalPredictor is in the handling of molecular flexibility. In version 1, molecules were treated as rigid fragments with an intramolecular energy that varied according to a spline function, parameterized by (i) the energy values and (ii) the energy derivatives calculated at a grid of points in the space of the flexible torsion angles. The atomic charges were either held constant (not satisfactory for large variations in molecular geometry) or varied according to what turned out to be a highly inaccurate spline. In version 2, molecular geometry, atomic charges and intramolecular energies are all given by a Local Approximate Model, a technique detailed in:

“Efficient handling of molecular flexibility in lattice energy minimization of organic crystals”, A.V.Kazantsev, P.G.Karamertzanis, C.S.Adjiman and C.C.Pantelides, *JCTC,***7**, 2011, pp1998-2016

The technique may be summarized as follows:

+ As with version 1 CP, a grid of points in the space of flexible torsion angles is defined.

+ Each point sits at the center of a hypercube of torsion values that extends half way to the next point in each dimension / torsion. Within that cube, molecular flexibility is modelled using the LAM parameterized by calculations at the central point.

+ The intramolecular energy is calculated using a quadratic model based on the energy and its first and second derivatives, calculated at the center point.

+ The atomic charges are set at the values calculated at the center point, i.e. a constant set of charges is used for each LAM region.

+ The molecular geometry is set to the value calculated at the center point, then modified (usually very slightly) by a linear model as the flexible torsion angles vary away from the center point.

Hence, compared to version 1 CrystalPredictor, molecular flexibility in version 2 is handled more accurately using LAMs: molecular geometry is updated to reflect the effect of changing torsions on the rest of the molecule, atomic charges are adapted to the changing molecular geometry without the need for inaccurate splines, and the intramolecular energy is handled using a model that is more accurate than the splines previously employed.

This document explains how to run CrystalPredictor version 2.

**1. Generating a LAM grid**

CrystalPredictor v2 describes the molecular geometry entirely in terms of z-matrices. As was also the case when generating the grid for version 1, it is therefore vital to acquire a z-matrix for the molecule that allows the molecule to respond appropriately to changes in the flexible torsions.

<< see v1 notes>>.

Once you have a z-matrix for the molecule (in any initial geometry), you can proceed to generate the LAM grid. For each point, this proceeds in two steps, using *ab initio* calculations in Gaussian: local optimization of the molecular geometry, then generation of the hessian matrix. Local optimization minimizes the energy of the molecule with respect to all degrees of freedom *except* the torsions that you want to search; these are frozen at the values chosen selected by you to form the grid. From the locally optimized degrees of freedom, we can approximate the changes in the molecular geometry with the varying flexible torsions.

Note that you can optionally designate ‘constrained’ or ‘frozen’ degrees of freedom. These are aspects of the molecular geometry that you do not want to change *at all* during the search; for example, because they are themselves highly flexible, but you are not interested in searching them at the moment, so you do not want their large variations complicating the current search. They will not be optimized during LAM grid generation and will not be changed by the flexible degrees of freedom.

A series of scripts have been prepared to help construct the LAM grid. All of these work from an input file called ‘min\_input’ (see example files). This contains:

+ A list of the flexible degrees of freedom

+ A list of any constrained degrees of freedom

+ A list of the range and intervals of variation for each flexible degree of freedom; this defines the grid.

+ A complete Gaussian input file for the initial local minimisation.

As well as this, you will also need a Gaussian runscript for your cluster. With these files, you should then run the scripts:

(i) min\_setup. This will make a series of directories corresponding to the points on your grid. Each contains a Gaussian input file and runscript, ready to be submitted.

(ii) grid\_manag. Set this going using the command ‘nohup ./grid\_manag *Njobs* &’, where *Njobs* is the total number of grid points to be run. This will run continually on your front end, submitting local minimizations to your queue. It will ensure that a set number are always on the queue, until all have been submitted. To select how many jobs you are comfortable with, make a file called ‘num\_cor’ containing *Njobs.*

(iii) hess\_setup. This will go through the local optimization output, extract the optimized parameters, and set up the input files for the hessian calculation, in a subdirectory of each grid point called ‘hessian’. You will also need a submission script called ‘runhgauss.csh’ that deals with the hessian submission (example provided).

(iv) hess\_manag. Works exactly the same way as grid\_manag.

(v) makelam. This fortran program extracts the hessians, derivatives, molecular geometries and point charges and constructs a (usually very large) file containing the complete LAM grid. This can then be appended to a molecular information file header (see below) to give a complete molecular information file.

**2. Input files**

CrystalPredictor2 requires two input files plus one information file per molecular type. These are ‘input.in’, ‘potential.in’, and additional molecular information files as designated in ‘input.in’.

Note that, unlike in version 1, there is no need for a ‘unix\_executable’ or ‘input’ subdirectory; all files are kept in the run directory.

**2.1 input.in**

See also the accompanying input.in examples. This contains:

+ The number of different molecular types.

+ For each molecular type:

* the number of that type of molecule in the asymmetric unit cell
* the number of atoms in the molecule
* the name of the associated molecular information file
* the number of flexible degrees of freedom (‘0’ if rigid) and

a list of their names, and upper and lower bounds in the search.

 Note that the bounds for each flexible degree of freedom should not exceed the bounds of the available LAM grid, including the region of LAM validity.

+ The list of space groups to be searched. In version 1, space groups were sampled randomly with a frequency determined by their frequency of occurrence in the Cambridge Structural Database. This is also the default in version 2. If the word ‘UNIFORM’ is added after the ‘SPACE\_GROUPS’ header, though, the listed groups will instead be sampled uniformly.

+ The parameters for the local minimizations and the search. Note that three values are given for the upper and lower bounds on angle and cell length. These are bounds on the 1st, 2nd and 3rd angle / cell lengths; setting all three equal will give a search equivalent to version 1. They can be set to different values for e.g. to focus on a particular range of parameters that match an X-ray powder pattern.

**2.2** **potential.in**

Contains the parameters for the repulsion-dispersion potential to be used in the search. Exactly the same as in version 1. Cross terms are calculated by the usual combination rules.

**2.3** **molecular information files**

See the accompanying example files. Each file includes a header with basic information about the molecule. If the molecule is flexible, this will be followed by the LAM grid, with each entry listing point charges, molecular geometry, energy, energy derivatives, and the hessian matrix.

The header section has 11 lines of text (not read by CrystalPredictor, but usually naming the molecule and the level of theory used to construct the LAM gird or just acquire the basic geometry). Following that is:

+ a line giving the number of flexible torsions (0 if rigid)

+ lines listing the upper and lower bounds of the LAM grid – note these are the upper and lower points, the range of validity will extend beyond these.

+ The number of frozen / constrained degrees of freedom, if any, followed by a list of them. Leave this section out ENTIRELY if none are constrained.

+ The initial z-matrix. This section contains a column listing the atom names, which double as the potential type, and hence must correspond to the types in ‘potential.in’. Note also that these must be in the format (element)(number). The second column is the symmetry type for the atom; if the molecule is symmetric then the matching atoms should have the same number. This will be used for clustering later on. e.g. in benzene all the carbons are type 1 and all hydrogens are type 2. The third column is the atomic charges. In a flexible search these will be completely ignored and can be set to 0, or anything else. In a rigid search these charges will be used. All the other columns in this section are then a conventional z-matrix.

+ The values of the z-matrix parameters. These will not be used in a flexible search (the values for individual LAMs will be used instead). These are listed in the usual z-matrix order, except that flexible and frozen degrees of freedom are moved to the end.

+ Any LAMs.

**3. Post-processing**

As in version 1, this is done by Analyse. As before, it will produce .pdb and .spf files. However, Analyse v2 will also generate .res files; additionally, these are modified to give the versions of their unit cells that are closest to orthogonal (giving a ‘neater’ unit cell), i.e. Analyse now incorporates the functionality of Imperial\_Rename (which is no longer needed).

A new version of the code ‘Minimise’ is also available. This works similarly to the old version except that, at the time of writing, the symmetry reduction functionality has not been incorporated into the new version. Note that the atom order in expcrys.pdb must match that in the molecular information file.

The script ‘see’, which analyses the file crystals.out, has also been updated. It works as before except that a new category of local minimization failure has been added. -7 indicates ‘LAM trapping’; a local minimization has been ‘trapped’ cycling back and forth between two (or more) neighbouring LAMs, which in the current code version is an unrecoverable error.

**Appendix A: complete list of available space groups**

P1 P-1 P21 P21/C P21212 P212121 PNA21 PCA21 PBCA PBCN

C2/C CC C2 PC CM P21/M C2/M P2/C C2221 PMN21

CMC21 ABA2 FDD2 IBA2 PNNA PCCN PBCM PNNM PMMN PNMA

PBA2 CMCM CMCA FDDD IBAM P41 P43 I-4 P4/N P42/N

I4/M I41/A P41212 P43212 P-421C I-42D P31 P32 R3 P-3

R-3 P3121 P3221 R3C R-3C P61 P63 P63/M P213 PA-3

P2221

**Appendix B: torsion group functionality**

In version 1, it was possible to define ‘torsion groups’, that is, separate groups of torsions that calculate separate intramolecular potentials using different grids. The big advantage of this is that if you have, say, 6 torsions each varying through 90o on a grid of 30o spacing, you need to calculate

46 = 4096 grid points. If you split them into two groups of three, you instead need to calculate

2 x (43) = 128 grid points. In addition, version 1 couldn’t handle more than 4 torsions in a group at all. So this functionality was quite important, even though it led to much lower accuracy in principle.

It should be less important in version 2, which can handle any number of flexible torsions and allows for much wider grid spacing, e.g. in the above example you could space the LAM points at 45o intervals and calculate 26 = 64 grid points. However, there are still conceivable examples where it might be necessary to break the flexible torsions down into separate groups. Hence, this functionality has also been built into version 2. The costs in accuracy are potentially severe. Charges and molecular geometry can no longer be extracted from the nearest LAM point, since any point in the torsion space will correspond to one LAM from each torsion group, and there is no rigorous way currently known to combine the charges and geometries. Charges and molecular geometry (except for the flexible torsions) are hence drawn from the header section (just like a rigid molecule) and held constant. Only the intramolecular energies are still calculated from the LAMs.

Instructions for running with torsion groups are given below. Note, though, that I have only had time to try one test case; and although that reproduced the experimental structures, I got some very strange results otherwise, including 70% LAM trap failures. I still don’t know if this means there’s a bug or if it was just that one case or if LAMs and torsion groups don’t work together. Hence this functionality is NOT RECOMMENDED – at least until someone else has looked at the code and tested it a bit more.

+ Only the molecular information file needs special treatment – the rest of the input is the same as for a normal run.

+ For each torsion group, use all the normal scripts to generate a grid of LAM points BUT hold all the torsions in every *other* torsion group as ‘constrained’.

+ In the molecular information file header, after the line ‘Across N dimensional grid:’ (set N to the *total* number of torsions), add a line: ‘In T groups of a b c… torsions’. If you are not running with torsion groups, leave this out completely. The number of torsions in each group are listed in order separated only by spaces.

+ List the torsions in each group and the ranges in the usual format, *but* precede each group with a line: ‘Group M’. Leave out these lines if you are not running with torsion groups.

+ Put in the LAM data for each torsion group, one after another. Precede each group with a row of hashes (like those used to separate individual LAM points) with ‘GROUP M’ in the middle. Start numbering of LAM points from 1 again with each torsion group.

+ Remember that the charges and molecular geometry you put in the header will be used throughout the search!

+ See also the accompanying example files.