The Program ZMC: A Monte Carlo Approach to Modelling the Diffuse Scattering from Short-Range Order (SRO)

D.J. Goossens

May 5, 2011

It is best to do things systematically, since we are only human, and disorder is our worst enemy.

Hesiod ($\sim$ 800 BCE)

In mathematics you don't understand things. You just get used to them.

Johann von Neumann (1903-1957)

If I had been rich, I probably would not have devoted myself to mathematics

J. L. Lagrange (1736-1813)

Preface

This is a book about Monte Carlo (MC) modelling of short-range order (SRO) in crystals, and in particular about using the program ZMC. It grew out of an attempt to document a program, called ZMC [1, 2], which is designed to allow (reasonably) ready implementation of MC modelling of SRO in molecular crystals. The program is in constant development, but in 2008 and 2009 underwent major modification into a version that will henceforth see only minor modification. This document is centred on ZMC, but is not restricted to it.

The program ZMC does some things and not others. It does not do occupancy simulations, which is to say that problems that require the ordering of different species, such as ordering in a binary alloy for example, cannot be dealt with. What ZMC will do is let the the atoms displacively adjust once the occupancy structure is established. That may seem like a very limited mandate, but the program is designed to deal with molecules. Molecules can have internal degrees of freedom such as rotations of segments around bonds. ZMC can deal with multiple types of molecules in multiple possible orientations. Hence is an occupancy simulation is required, that is done externally to ZMC (though that will be discussed herein) and then the results are fed into ZMC for displacive relaxation. The discussion of *para*-terphenyl in chapter 4 will show that sometimes occupancy problems *can* be dealt with within ZMC... but it is a rather special case.

ZMC produces an output file which can be used in various ways. Usually they are fed into into DIFFUSE [3] to calculate diffuse diffraction patterns to compare with those observed; one day a module will be added to allow it to output to a format that can be used by DISCUS [4], which is a widely available and ready-to-use program, unlike DIFFUSE. Users are encouraged to think of other quantities that could be calculated from the various forms of output; pair distribution functions, even possibly EXAFS and XANES spectra; that is beyond the current author's scope. However, it is firmly believed that ultimately, bringing multiple techniques to bear on a single problem, and fitting them all jointly to a single model, will be a powerful way to deeper understanding.

ZMC and the associated papers and documents would not exist without several people. The two prime figures being Prof. T. R. Welberry and Dr A. P. Heerdegen. Richard Welberry [5] has spent many years developing these techniques, and what is presented here is merely a development of an aspect of his work, with perhaps more of an eye on usability for the non-expert than Richard has generally concerned himself with. Aidan Heerdegen is not only a fine scientist who has contributed much to the development of the algorithms in ZMC, his advice has also been crucial in turning vague ideas into concrete Fortran code, providing many of the key components of ZMC. The author would also like to

thank Dr E. J. Chan, whose use and modification of the program in pursuit of doing some *science* (which after all is what it is ultimately all about) tested the code and in doing so contributed greatly to ironing out kinks, squashing bugs and adding in capability and changing the fundamental design of aspects of the program.

Given the verbose, disjointed, untidy and probably inefficient nature of the code, it must be stated that while many others have contributed good ideas and suggestions to the project, any insufficiencies in the code, any programming habits that offend skilled and rigorous programmers (a group to which the author does not belong) and any bugs are the sole responsibility of the author.

The project is open to contributions; please contact the author.

ZMC is written in Fortran90, and has been compiled on Linux using the Intel Fortran Compiler [6] [7] and gFortran [8]. It has also been compiled on Mac OS X using g95. This document was written in LaTeX using the TDE editor [9] which I cannot recommend highly enough. ZMC uses a module library written and compiled by Dr A. P. Heerdegen.

Contents

Chapter 1

Defining the Problem

Modeling SRO is difficult because SRO consists of deviations away from the average structure, and as a result does not have to obey space group symmetry. The average across the whole crystal, the long-range order (LRO) has to obey this symmetry, but a given instance of local structure does not. Now, the exact instantaneous position of every atom in the sample containing $\sim 10^{23}$ atoms cannot be modelled. What can be modelled is the 'average' SRO instead of (or as well as) the average LRO. The LRO manifests in the Bragg scattering, whose analysis is well established, whether it be single crystal data or powder diffraction data. What does 'average SRO' actually mean? In this case, SRO is an ordering which persists over approximately tens of unit cells, and may be an ordering of species, displacements, or some combination of these.

When solving for the LRO, which is referred to as structure solution, several useful hints are available. First, the nature of the solution is known; a unit cell is required. An exception is, for example, a modulated structure, although in some sense the answer is still a unit cell, all be it one that is described in more than three dimensions.

A second useful thing is that only the Bragg peaks need be analysed. It is possible to establish systematic absences, unit cell parameters and space group symmetry through fairly well-established routes. Again, this is an over-simplification (there will always be difficult cases), but it generally holds.

Third, there is often a 'model' structure to begin with. A given material is often a derivative of something which is known.

The study of SRO can often make use of some of these factors, for example a 'family resemblance' with other problems. However, because SRO can take on so many forms, often when the analysis begins even the broad outlines of the solution are not known. Does the model need to have SRO correlated occupancies? Multiple molecular orientations? Are they correlated? Is the disorder purely displacive? Does size-effect need to be considered (of which more later)? Often, it is unknown whether the assumed model is even *capable* of describing the 'true' situation, no matter what values its parameters take on. So in a sense the type of model needs to be found before a specific model can be constructed.

In the work presented here, the modelling of SRO is tackled through constructing a model crystal. Because of the SRO, the unit cells are not identical; however, the interactions from cell to cell should be the same. Hence if the inter-

actions are used to correlate the molecules, the problem is reduced to one with a reasonable number of parameters—the parameters describing the interactions, rather than those directly giving the atomic positions. (As will be discussed in chapter 6, a further level of parameterisation has been explored, in which the interaction constants are themselves generated from a simple model with a handful of parameters.) And by using a sizable model crystal the model should effectively model the *population* of local configurations successfully.

This is still far from trivial. It is routine to spend considerable time interacting with a model before finally coming to the conclusion that it cannot fit the data.

Chapter 2

The Approach

2.1 Generalities

ZMC uses an approach based around the idea of Monte Carlo (MC) simulation [10]. MC can be used in many contexts—to fit equations to curves, minimise χ^2 for various kinds of fits and optimisations or bring a model system into thermal equilibrium. It is the last of these that applies here. The aim *is* to perform a fit—ultimately, the goal is to produce a model whose Fourier transform gives diffuse scattering that looks like that which is observed coming from the real system — but that is a loop 'outside' the running of a particular MC simulation.

MC simulation is used to bring a given model into thermal equilibrium; whether the parameters of that model are 'right' or not, whether the model 'fits' the data, these are other questions, beyond the scope of ZMC, though not beyond the scope of this document (see chapter 5). A rough flowchart of this can be seen in figure 2.1. The chart shows that what ZMC does (if run for long enough) is bring the system into thermal equilibrium, the idea being that the interactions in the system (all scaled relative to $k_{\mathrm{B}}T$) will induce some correlation structure in the displacements which will then cause features in the diffuse scattering pattern. However, there is no *a priori* reason why the output from ZMC cannot be used to calculate other quantities. The diagram also shows that ZMC in no way evaluates the model; it relies on the user to look at the outputs (diffraction patterns, histograms, correlations) and come up with a next iteration. The user might choose to embed ZMC within a process which automatically evaluates the outputs and adjusts the parameters of the model.

Here the key problem becomes apparent; is the model capable of fitting the data? It is possible to spend a lot of time fiddling with parameters, size-effects, occupancy structures, contact vector connectivities, molecular flexibility (molecular degrees of freedom in general) and other things before a model which is even able to fit the data is found. The question: "Do I need to adjust my model or do I need a new model?" Is often almost impossible to answer with certainty. ZMC only helps with this in so far as it makes construction of the model faster, which means it is also faster to modify the model. But the insight into what needs to be changed still must most often come from the user.

ZMC fits within a broader template—the use of diffuse scattering as a probe of SRO. This document is not intended to be a thorough-going examination of

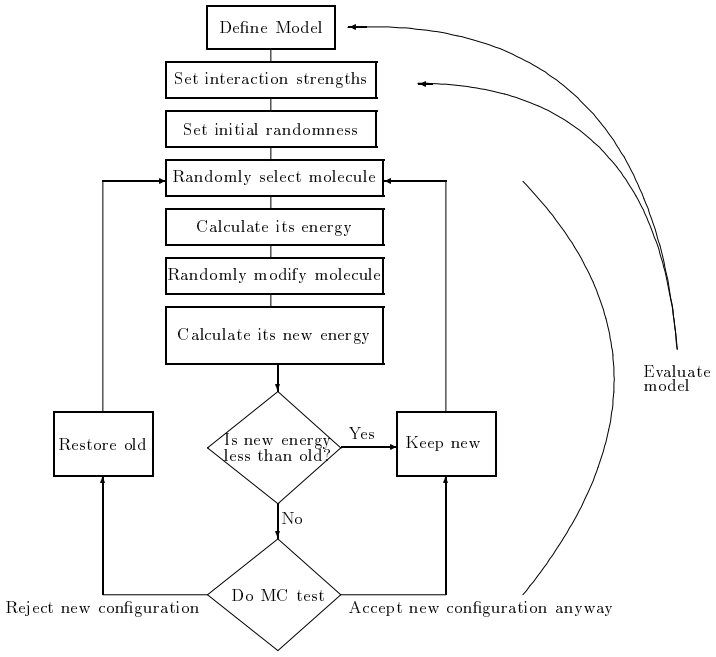


Figure 2.1: Flowchart of the MC process as in `ZMC`. The inner loops are within the program but the outer loop ('evaluate model') is not.

diffuse scattering, that has been done before, and been done very well [5, 11, 12]. In the broader context, the use of `ZMC` might be summed up by the diagram in figure 2.2, in which figure 2.1 is in a sense embedded.

Figure 2.3 is a false colour map of a section of reciprocal space, in which the x and y coordinates give the position in reciprocal space and the colour gives the intensity observed at that point. The first decision is whether an occupancy model is required or whether the SRO can be reproduced by 'only' correlating atomic or molecular displacements (and conformations). How can this be determined? First, the average structure from Bragg diffraction should be able to give some idea of whether there are split sites, for example. Otherwise, an incipient phase transition (for example a cell doubling) may give insight. The diffuse scattering itself will show some evidence, and sometimes it will be clear what features are due to the occupancies. For example well-defined diffuse spots at 'supper lattice' reflection positions, typically something like $(hkl) \pm \left(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right)$.

If an occupancy model is required, it is necessary to work out what the occupancy objects objects are. Multiple types of molecules? Multiple orientations and/or conformations of a single molecule? Then the nature of the correlations must be established. Along what directions are the entities correlated and how strongly? This most often reduces to looking at nearest neighbour correlations. Are they correlated positively or negatively with their nearest neighbours? In a more complex situation, there may be a need for a multivalued occupancy variable. This means that, for example, if there is a '1' on some site A then there will most likely be a '2' or '4' on site B, but less likely to have '3', '5' or

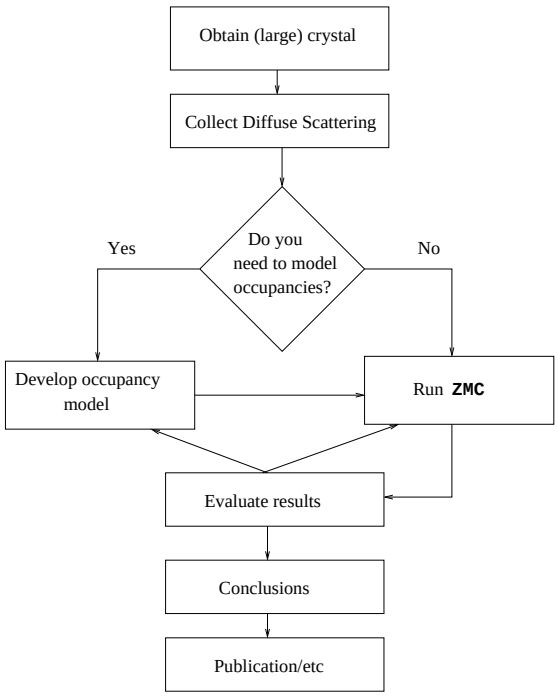


Figure 2.2: Flowchart of the SRO analysis process when making use of `ZMC`. The box 'Run `ZMC`' contains figure 2.1.

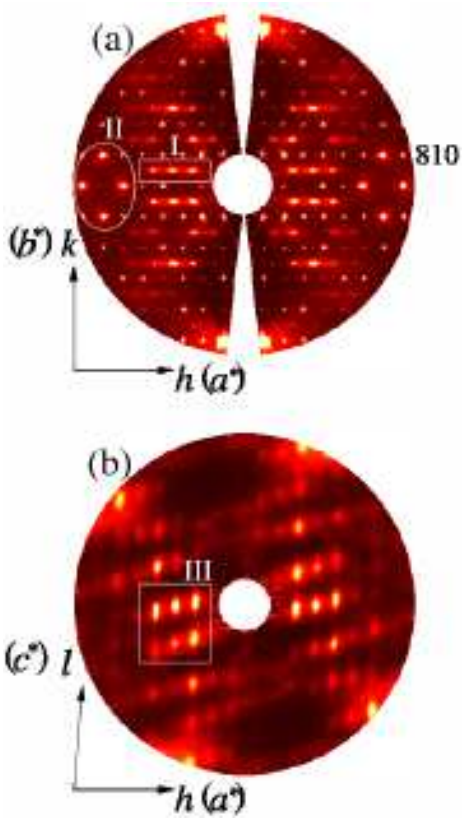


Figure 2.3: Example of observed diffuse scattering data, in this case two reciprocal space sections from *para*-terphenyl.

'6', where '1' etc just encodes say the position of some functional group on a phenyl ring, for example [13]. Similarly, there may be multibody interactions [5], in which the probability of occupying a given site depends jointly on the probabilities of several neighbouring sites.

Once (what appears to be) a good occupancy model has been obtained, it can be fed into `ZMC`. This requires thinking about how the molecules connect, what forces act between them and what size-effects are required. There is also the question of molecular flexibility; what degrees of freedom must the molecule(s) be allowed? There may be a need to take care in re-evaluating the occupancy structure at this stage, hence the going back and forth arrows in figures 2.1 and 2.2.

A model is evaluated by calculating its diffuse scattering pattern(s) using DIFFUSE then comparing with observations. The comparison may be done qualitatively (that is, by eyeball) or using some kind of refinement program, including least squares [14] or genetic algorithms [15].

2.2 Specifics

In this section, some of the details of the implementation will be discussed, with a view to defining terms and outlining possible pitfalls and useful tactics. First, molecular representation.

2.2.1 Representing a Molecule

A molecule can be anything from a single atom to a large assemblage of atoms. In `ZMC` a molecule is represented as a z-matrix. An example is shown in table 2.1. In this case the molecule is deuterated *para*-terphenyl, shown schematically in figure 2.4. The numbering of atoms in the z-matrix is chosen so that the groups of the molecule can be twisted around the C–C single bonds by changing the dihedral angle of a single appropriate atom. That means that all ensuing atoms have to be forced to depend on the previously defined atoms in the right way, or for example a phenyl ring might get broken in a unphysical way, if all atoms in the ring do not follow the one whose angle is being changed.

The process used to generate the z-matrix depends on having a good quality average crystal structure, with the ability to separate overlapping, superimposed molecules in disordered structures so that the 'building block' unit of the structure, the isolated molecule, is known. It may be that a similar structure without disorder can be used as a guide, and many units in molecules (phenyl rings, for example) can often be usefully approximated by known geometries.

The process is typically done in stages. First, a `.cif` file is obtained (from a database of from a conventional single-crystal Bragg experiment, or constructed by hand if the problem requires this), and the atoms in it reordered such that the expected internal degrees of freedom (for example the angle of the central phenyl ring in the molecule in figure 2.4) are logically defined. It may be useful to define 'dummy' atoms (those beginning with 'x' in table 2.1) to allow more convenient definition of degrees of freedom and of the molecular axes. If a molecule has a centre of symmetry, it may be desirable to place the molecular origin on this point, even if there is no 'real' atom there. In the example given here, this has been taken rather to an extreme; while the molecular origin is on

Table 2.1: The z-matrix for d-*para*-terphenyl. Atoms with labels prefixed 'x' are 'dummy' atoms used to aid in defining the local coordinate system and internal degrees of freedom. The starred (*) value is allowed to vary, and this allows the whole central phenyl ring to rotate as a rigid unit.

	Label	l	Distance from l (Å)	m	Angle with lm (°)	n	Dihedral angle with lmn (°)	
1	x1	–	——	–	——	–	——	
2	C4	1	2.912	–	——	–	——	
3	C5	2	1.406	1	120.510	–	——	
4	C6	3	1.395	2	120.251	1	179.771	
5	C7	4	1.387	3	120.678	2	-1.240	
6	C8	5	1.384	4	119.710	3	1.561	
7	C9	6	1.394	5	120.468	4	-1.155	
8	x2	7	1.409	6	120.519	5	0.432	
9	C4B	8	5.825	7	121.127	6	-179.368	
10	C5B	9	1.406	8	120.510	7	-0.750	
11	C6B	10	1.395	9	120.251	8	-179.771	
12	C7B	11	1.387	10	120.677	9	1.240	
13	C8B	12	1.384	11	119.710	10	-1.561	
14	C9B	13	1.394	12	120.468	11	1.155	
15	x2B	14	1.409	13	120.519	12	-0.432	
16	C2B	15	1.329	14	121.037	13	179.518	
17	C1B	16	1.407	15	121.135	14	-180.000	*
18	C3B	17	1.393	16	121.658	15	-179.878	
19	C2	18	1.406	17	120.534	16	-0.732	
20	C1	19	1.407	18	117.804	17	0.700	
21	C3	20	1.393	19	121.658	18	-0.713	
22	D1	20	1.107	21	118.327	19	-176.663	
23	D2	21	1.061	20	118.909	22	5.629	
24	D1B	17	1.107	18	118.327	16	176.662	
25	D2B	18	1.061	19	120.538	17	-178.415	
26	D3	3	1.064	4	121.032	2	-171.714	
27	D4	4	1.056	5	122.918	3	-178.101	
28	D5	5	1.042	6	118.647	4	176.696	
29	D6	6	1.127	7	118.434	5	170.882	
30	D7	7	1.066	8	122.600	6	174.386	
31	D3B	10	1.064	11	121.033	9	171.714	
32	D4B	11	1.056	12	122.918	10	178.101	
33	D5B	12	1.041	13	118.647	11	-176.696	
34	D6B	13	1.127	14	118.434	12	-170.882	
35	D7B	14	1.066	15	122.600	13	-174.386	

the middle of the central phenyl ring, the desire was to allow only the central ring to rotate relative to the two our ones. Hence the second atom is a very long way away from the first, and the two out rings are defined first, before returning to define the central ring last.

Then Mercury, a program which is part of the CCDC [16] is used to populate the unit cell with contiguous, unbroken molecules. Then that cell is saved as a `.mol2` file, which is processed by a custom-written program called zmat_maker which outputs as many z-matrices as required, along with the coordinates needed

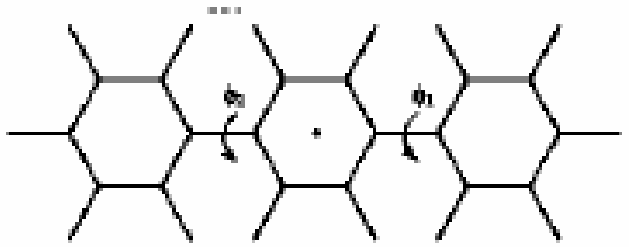


Figure 2.4: Schematic diagram of the *para*-terphenyl molecule, with internal twist angles ϕ_1 and ϕ_2 noted. An example of the H–H *intra*molecular contacts which prefer a non-coplanar configuration of the rings is noted as a dashed line. Note that in practice the model uses only a single rotation angle, ϕ, which gives the angle of the central ring to the plane of the outer two.

to place them into the unit cell, thus providing all information needed to populate a unit cell.

Now, the molecule can be thought of as consisting of a series of variables which fall into two classes—external and internal. The external ones are $\mathbf{x}$, a 3-vector giving the Cartesian position of the origin atom of the z-matrixand $\mathbf{q}$, a quaternion, a 4-vector giving the direction in the unit cell frame of reference of the first bond defined in the molecule. These two things serve to position and orient the molecule in the unit cell.

The internal variables are contained in a vector $\mathbf{i}$ of length N where N is the number of internal degrees of freedom in the molecule. For the example in figure 2.4, $\mathbf{i}$ is of size 1—it contains only the value of the angle the central phenyl ring makes to the plane of the outer rings. Generally only dihedral angles are allowed, since these are likely to be the lowest energy motions; but there is no other reason why bond angles and lengths cannot be allowed to change.

During a MC step (see figure 2.1), the molecule is chosen, its energy is calculated, then random shifts (whose maximum magnitude is either input by the user or the result of an automated optimisation process) are applied to all or some of the components (as specified by the user) of $\mathbf{x}$, $\mathbf{q}$ and $\mathbf{i}$. This will stretch and compress the 'springs' modelling interactions within the molecule and connecting the molecule to its neighbours, changing the molecule's energy (see section 2.2.2), and so when the energy is calculated a second time, the Metropolis MC step [10] proceeds. It should be noted that $\mathbf{q}$ is a normalised vector, so after the random shifts are applied to its components, the q_j, it is renormalised.

2.2.2 A Molecule's Energy

Internal degrees of freedom (the values stored in $\mathbf{i}$) will cause an energy penalty whenever the value of the variable is not some 'ideal' value set by the user. The contribution of this energy, E_{intern}, to the energy of a molecule might be written

$$E_{\text{intern}} = \sum_{j=1}^{N} F_j(\phi_j - \phi_{j0})^2 \tag{2.1}$$

where the molecule has N internal degrees of freedom indexed by j, ϕ_j is the 'instantaneous' or current value of the jth degree of freedom (an angle or length) and ϕ_{j0} is its 'ideal' (zero energy penalty) value. F_j is the associated force constant.

Now, it is quite reasonable that when one segment of the molecule twists one way, for example, another segment might find it energetically favourable to twist another way. Hence the internal degrees of freedom are allowed to interact, though only on a pairwise basis, with interactions of the form:

$$E_{\text{cross}} = \sum_{j=1}^{N} \sum_{k=1}^{j-1} F_{jk}((\phi_j - \phi_{j0}) \times (\phi_k - \phi_{k0})) \tag{2.2}$$

where the subscript 'cross' is used to indicate that these are 'cross-terms' in the internal energy. Many, possibly all, of the interaction constants, the F_{jk}, will be zero.

These two terms combine to give the 'intramolecular' energy of the molecule, $E_{\text{intra}} = E_{\text{intern}} + E_{\text{cross}}$.

The *inter*molecular component of the energy is calculated by assuming the molecule is connected to surrounding molecules *via* Hooke's law springs (harmonic potentials). Harmonic potentials are used because they are simple and fast to implement, they can be parameterised by a single number, they are a reasonable approximation close to the bottom of many types of potential well (Lennard-Jones, Buckingham for example) and they are a weak assumption. They are bad for most of these same reasons—particularly, they will be too weak when the distances become short (repulsion is ultimately extremely strong since atoms are unlikely to invade each others' space) and too strong when the interatomic separations become large—for large separations, atoms are not interacting at all, so to have a large energy penalty there is nonsense. Also, being harmonic they are inherently unable to capture any symmetry-breaking interactions in the system, meaning they cannot model a phase transition. These things can be dealt with by using more complex potentials—at the expense of more computing time and perhaps more free parameters in the modelling (and ultimately in the 'fit'), something which may or may not be valid given the power of the data to discriminate between models. It is intended that `ZMC` be able to allow the user to choose Lennard-Jones potentials in the input file, but as of time of writing this is not implemented in the 'canonical' version of the program. Some of these issues are discussed in chapter 6.

Various strategies can be used when selecting atom-atom interactions to connect molecules. Several studies (amongst many others, [17, 13]) have used a relatively small subset of the possible 'contact vectors' connecting molecules (this is for speed reasons, and again to minimise free parameters in the model). Such models use just enough springs to mutually orient the contacting molecules and/or their segments. Hence the interactions are 'effective' interactions, and do not admit of detailed interpretation (except in rare cases).

Another approach is to use all interactions out to some cutoff length; this will generally imply the existence of hundreds of interactions which, strictly speaking, are not symmetry equivalent and therefore should have different force constants. However, if the force constants can be chosen using some simple procedure (for example a function of the atom types and their expected Van der Waals radii), the number of parameters can again be reduced.

Hooke's law is $F = -kx$ where F is the force, k is the spring constant and x is the extension of the spring. Integrating with respect to x gives the energy associated, $E = \frac{kx^2}{2}$. Recasting this to suit the modeling discussed here gives

$$E_{\text{inter}} = \sum_{cv} F_{cv}(d_{cv} - d_{cv0})^2 \tag{2.3}$$

where d_{cv} is the instantaneous length of the given contact vector, d_{cv0} is its 'equilibrium length' and F_{cv} is its associated force constant (note the '$\frac{1}{2}$' has been absorbed into F_{cv}), and the sum is over all contact vectors (cv) connecting to the given molecule. If the model uses a subset of the nearest neighbour atom-atom contact vectors, the interactions must be considered as 'effective' interactions only, and while the associated F_{cv} may be indicative of the relative strengths of the interactions in different directions, they should primarily be considered as a mechanism for inducing a correlation structure into the displacements.

There is (at least) one more complexity to be added in. This is the size-effect [5]. In brief, the size-effect is the change in expected atom-atom separation when the occupancy changes. Consider a binary alloy. It may contain 50% A atoms and 50% B atoms in the crystal, distributed without long-range ordering (for example they are *not* perfectly clustered or perfectly alternating, or in chains). In this case the 'average' atom will be 50% A and 50% B, and the average atomic separation will be an average of the preferred separation of B–B pairs, A–A pairs and A–B pairs. Yet when a single real site is considered, it will be occupied by A *or* B—there is no such atom as '50% A 50% B' in the periodic table. And the atomic position will depend on how the atom can best satisfy the energetically preferred separations. This means that the preferred interatomic separation (d_{cv0} in equation 2.3) is a function of the occupancies of the connecting sites. Hence equation 2.3 can be modified by incorporating a size-effect term, ϵ, which is a function of the occupancies, S, of the two sites ($\epsilon = \epsilon(S_1, S_2)$):

$$E_{\text{inter}} = \sum_{cv} F_{cv}(d_{cv} - (1+\epsilon)d_{cv0})^2 \tag{2.4}$$

or

$$E_{\text{inter}} = \sum_{cv} F_{cv}(d_{cv} - (d_{cv0}+\epsilon))^2 \tag{2.5}$$

where equation 2.4 uses a fractional size effect and equation 2.5 uses an absolute one.

Hence, the total energy of a molecule in the crystal is $E_{\text{tot}} = E_{\text{inter}} + E_{\text{intra}}$.

In `ZMC` the size-effect is implemented in two ways, allowing the user to choose that which makes most sense. In both cases, the contact vectors connecting the sites of varying occupancy (A or B), which could be called $\mathbf{v}_{\text{AA}}$, $\mathbf{v}_{\text{AB}}$, $\mathbf{v}_{\text{BA}}$ and $\mathbf{v}_{\text{BB}}$, must be treated as separate types, with the user knowing that they are manifestations of the 'same' type (and presumably will therefore all have the same spring constant—although there will be cases when that is a bad approximation). Once that is done, the four (three if $\mathbf{v}_{\text{AB}} = \mathbf{v}_{\text{BA}}$, which is not always the case) can (1) all be given different equilibrium lengths in the list of contact vectors (2) all be given the same length but different size effect or (3) a combination of the two. The advantage of the 'size-effect' approach is that that number can be altered in the input file easily, whereas if implemented in the contact vector list, it must be altered for each occurrence of that type of vector.

To put this another way, as noted, by having a separate class of contact vectors for each combination of occupancies: In the binary alloy case discussed above there are four possible nearest neighbour contact vectors, which could be called $\mathbf{v}_{\mathrm{AA}}$, $\mathbf{v}_{\mathrm{AB}}$, $\mathbf{v}_{\mathrm{BA}}$ and $\mathbf{v}_{\mathrm{BB}}$. There are then three possible preferred interatomic distances, d_{AA}, d_{AB} and d_{BB}, where plainly $d_{\mathrm{AB}} = d_{\mathrm{BA}}$ (plainly in the case of A and B being simple atoms, but if they are complex molecules this may not be true). These could either be considered as four classes, each with one of three possible lengths, or as four classes, all of the same length but with different size-effects. Both these ways of considering the problem can be implemented in `ZMC`. `ZMC` uses an absolute size-effect (equation 2.5). It allows implementation of a size-effect as well as changing the vector lengths themselves because the size-effect required to model the data is generally not known before starting, and it is easier when interacting with the program to change one size-effect parameter in the input file than to edit the contact vector list.

The average separation between average atomic positions is known, so the size-effects should be set to maintain this. If considering the binary alloy case, the occupancy variables for two sites (S_1 and S_2) will be 2-valued variables, whose two values could be 0 or 1, $+1$ and -1, 1 or 2, A or B as above, or anything else that suits. Here they will be denoted $+1$ and -1, or $+$ and $-$ for short. Then if bonds (or contact vectors) rather than sites are considered for a moment, there will be four kinds of contact vectors (this is the same as noted above in the different notation): $\mathbf{v}_{++}$, $\mathbf{v}_{+-}$, $\mathbf{v}_{-+}$ and $\mathbf{v}_{--}$. There are three possible preferred interatomic distances, d_{++}, d_{+-} and d_{--}, where plainly (in this simple case) $d_{+-} = d_{-+}$. The actual number of each type of vector will depend on the correlation structure of the occupancies. Figure 2.5 shows some possible cases for a simple square lattice binary alloy. The figure shows that the number of each type of contact vector in the model crystal depends on the occupancy correlations (positive in figure 2.5a, negative (-1 in fact) in 2.5b, zero in 2.5c and negative but of magnitude less than unity in 2.5d.

If N_{++} is the number of $++$ contact vectors, and similarly for $--$, $+-$ and $-+$, then the concentration of that vector type can be defined as

$$c_{++} = \frac{N_{++}}{N_{++} + N_{+-} + N_{-+} + N_{--}} \tag{2.6}$$

and then by definition

$$c_{++} + c_{+-} + c_{-+} + c_{--} = 1. \tag{2.7}$$

If it is allowed that the average separation must be maintained, then if some vectors are made longer others must be made shorter. If the size-effect on vector $\mathbf{v}_{++}$ is ϵ_{++} for example, then that would imply the constraint

$$c_{++}\epsilon_{++} + c_{+-}\epsilon_{+-} + c_{-+}\epsilon_{-+} + c_{--}\epsilon_{--} = 0, \tag{2.8}$$

and in the binary alloy case, $c_{-+} = c_{+-}$ and $\epsilon_{+-} = \epsilon_{-+}$. This may seem self evident, but if for example '+' means a molecule is in one orientation and '$-$' means it is in another, then the symmetry of the molecule may well mean that $\epsilon_{+-} \neq \epsilon_{-+}$. Consider the four pairs of (simple) molecules drawn in figure 2.6. Note also that this development is for crystals in which the interacting objects have only two possible configurations. When more states are possible, the equations must be appropriately generalised.

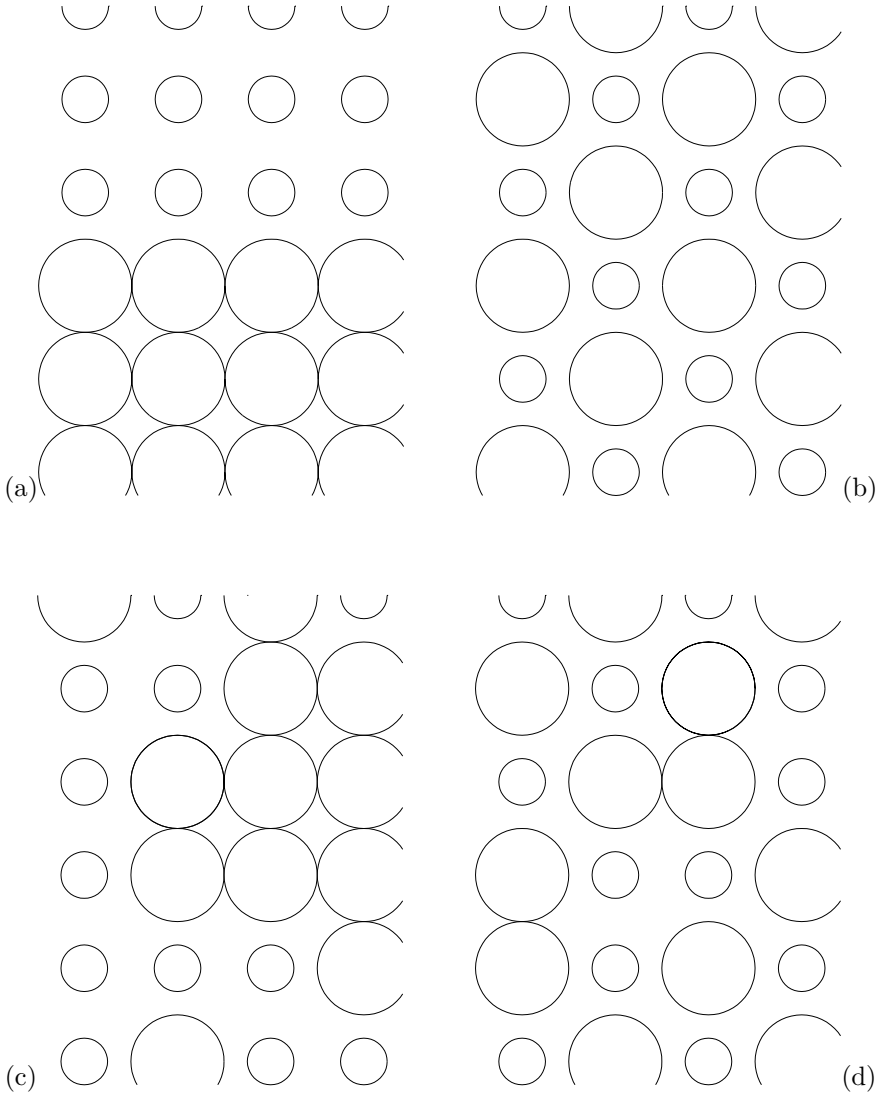


Figure 2.5: Four possible ways of ordering 'big' ($S = +1$) and 'small' (S=-1) atoms in two dimensions on a square grid. Note that in (a) phase segregation has occurred and so very few contact vectors between unlike atoms are present. In (b) there are no contacts between like atoms due to the perfect alternation. In (c) there is random occupancy (when drawing the diagram, big or small atom was chosen by tossing a coin) and so about 1/4 of the vectors are small to small, 1/4 are big to big and about half are mixed. (d) shows a SRO system which obeys none of these extreme cases, although atoms tend to alternate.

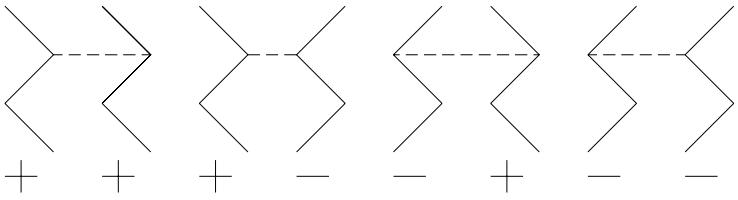


Figure 2.6: Four possible ways of ordering flips on a molecule. '+' means the molecule zig-zags one way, and '−' means is zig-zags the other. The contact vectors noted by the dashed lines are equivalent, and it is plain that $\epsilon_{+-} \neq \epsilon_{-+}$, whereas it is quite likely that $\epsilon_{++} = \epsilon_{--}$.

The size-effect discussion assumes that all the vectors that are equivalent (which interchange upon changing the occupancy—for example all the dotted lines in figure 2.6—have the same spring (force) constant, F. This may not be a terribly physical thing to do—plainly in a molecule, different contacting atoms may have different bonding properties, whether the contact vector is modelling Van der Waals forces, covalent bonding or something else. Given that the model crystal will have cyclic boundary conditions and fixed lattice parameters, it is a moot point whether care must be rigorously taken to obey equation 2.8 or not. Anecdotal experience suggests it is not a problem unless the parameters used are completely unreasonable.

Size-effect implementation in `ZMC` might be conceptualised as a shift of the interatomic harmonic potential such that the forces between the various pairs of interacting atoms are not going to be zero when all atoms are at their average positions. It will be energetically favourable for some pairs to have longer than average separations, and for others to have shorter, depending on their occupancy; this is what size-effect means.

Since the molecule may be subject to conflicting demands, and unable to satisfy all interatomic potentials at once, the resulting actual shifts away from the global average separations induced by the size-effect parameters will in general be *much* smaller than the size-effect parameters themselves (perhaps an order of magnitude smaller). Indeed, depending on the other energy terms in the system, the shift may not even be of the same sign. Plainly this will partly depend on the relative sizes of the spring constants as well.

This all may sound complicated, but it is one of the strengths of the MC approach that, notionally at least, the various key aspects of the chemistry of the problem should be able to be put into the model (and in particular into the various terms in the molecular energy) and then allowed to interact *via* the MC algorithm. This *should* allow the resulting sometimes quite complex behaviour to arise 'naturally'. This is what would happen in a real system. In practice, it is very often just these conflicts between different aspects of the system that give rise to complex SRO and resulting diffuse scattering because they prevent the system reaching a nice, long-range-ordered state which can be essentially completely described by a conventional structure analysis.

Here one quality of the MC approach becomes apparent: It is almost infinitely flexible, but often it is very difficult to know what needs to go into the model in the first place. `ZMC` implements a simple subset of the infinite num-

ber of possible energy terms that can be abstracted from real systems. It is possible to imagine energy being modelled by multipole expansions, multi-body interactions, differently shaped potentials, hard-sphere models, empirical energy terms that depend on bond valence sums [18], energetics of excluded volumes, Ising-like energy terms, arbitrary rules that can be written as a series of IF statements in a computer program... the alternatives are endless. ZMC uses a handful of these approaches—really only one of them, with a little bit of flexibility built in—that are deemed appropriate for the class of problems it was designed to tackle, namely modelling SRO in molecular crystals. That is not to say it cannot be used for other things. A nail may be driven in with the back of a screwdriver.

2.2.3 Occupancy Simulations

If a model requires occupancies, that is not normally handled by ZMC. What is meant by occupancies in this case is that two or more types of objects have to be ordered, or the same object has to be ordered in two (or more) conformations or orientations that will not be able to interconvert thermally during the ZMC modelling. It may be desirable to model cases that *can* interconvert with occupancies also.

ZMC can take a given molecule and rotate it, shift it or let it twist, bend and stretch around internal degrees of freedom. It cannot just flip it over or swap it for another molecule. Hence, what the program does is read in a file which tells it what type of molecule is on each position in the model crystal, and what orientation it is in; if there is no disorder of this type, such a model might be referred to as 'a purely thermal model'. In a purely thermal model in which all molecules are the same and there are no 'occupancies', both the molecule number (z-matrix number) and the orientation number will always be unity — in each case there is only one possibility. In a binary alloy, where one 'molecule' is an atom of type A and a second is an atom of type B, then there is only one possible orientation, but two possible molecules on a given site, so the orientation number would always be unity, but the molecule number could vary (and show SRO). Or there could be a molecular system in which the one molecule is found but it can flip over (see for example figure 2.6) in which case the molecule number would always be unity but the orientation number could vary (and show SRO). Or there could be instances where there are multiple types of molecules with multiple possible orientations. In one problem, there are two different types of molecules, one of which can flip over and the other of which cannot [19, 20].

All this means is, a means of producing that file which tells ZMC what type of molecule is on each position in the model crystal needs to be developed. For purely thermal problems, this is simple — in fact, strictly no such file is needed, as it is known what molecule is on each site, and what orientation it is in, and only the displacive part needs to be done. ZMC deals with these cases automagically, because when it interrogates the input files, it works out what can go where and only looks for an occupancy file (as it will be referred to) if it finds out that a one or more positions can have more than one molecule type and/or more than one orientation—and it *should* complain if it cannot find one. But if occupancies are present, that file does need to be produced, and this section outlines a couple of strategies for doing that—not all of them, though.

Further, while later sections of this document will detail the way in which `ZMC` operates, it will not detail the way in which occupancy simulations work in the same detail. Probably.

There are a few broad ways of inducing occupancy structures. One is the idea of a growth model [5] in which the occupancy structure is assembled from a series of rules, as if growing from some starting point. These rules generally take the form of conditional probabilities. Occupy some site according to the occupancies of the already occupied sites, but do not do it completely prescriptively or the resulting structure will be long-range ordered (or frustrated, possibly).

A second approach is to again use MC, but this time on an array of variables representing the molecular species and/or orientations. The simplest case here is the binary case, where the energy of the ith molecule might be calculated as

$$E_i = \sum_j w_{ij} S_i S_j \tag{2.9}$$

where the ith molecule has N interacting neighbours indexed by j. This is analogous to the Ising model from magnetism, so that the S will take on values of +1 or -1. The simplest case is if it is assumed that only nearest neighbours (NN) interact. This means that the energy becomes

$$E_i = w_{\mathrm{NN}} \sum_j S_i S_j \tag{2.10}$$

and plainly, the sign of w_{NN} will determine whether the S tend to correlate positively or negatively. This can be seen by considering an MC program using equation 2.10 as the energy.

If w_{NN} is positive, then $w_{\mathrm{NN}} S_i S_j$ will be negative (a lesser contribution to the system's energy) if S_i and S_j are of opposite sign. Hence, equilibrating at non-zero temperature, T, will cause the S to be correlated negatively, with the actual strength of the correlation dependent on the ratio of T to w_{NN}, and also on the composition of the system (if there is not a 50:50 ratio of atoms of type A and B, for example in the binary alloy case, then perfect alternation is impossible and the correlation can never achieve -1). Similarly, on a triangular lattice there can be frustration if negative correlation is desired (see figure 2.7a).

The model can be complexified by allowing more distant interactions—there could be second nearest neighbour (2NN) interactions, for example, such that equation 2.10 becomes

$$E_i = w_{\mathrm{NN}} \sum_j S_i S_j + w_{\mathrm{2NN}} \sum_k S_i S_k \tag{2.11}$$

where k indexes the second nearest neighbours. This introduces further possibilities for frustration. If both w_{NN} and w_{2NN} are positive (so trying to induce negative correlation) then the system may not be able to satisfy this (see figure 2.7b).

Plainly, high degrees of complexity are possible. There could be anisotropy so that neighbours in a given plane interact more strongly or with different signs of correlation to those in other directions. Entities may have more than two possible states [13]. These things all require some sort of energy term which can be used in an MC algorithm to select certain configurations as favourable.

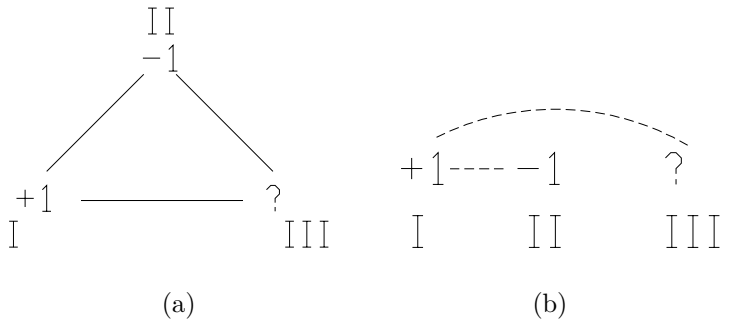


Figure 2.7: Two ways the desire for strong correlation can be frustrated. In (a) it is because of the triangular lattice. If all sites (I, II & III) want to be of opposite value to their neighbours, what can be put on the unoccupied site (site III)? In (b) it is because of the existence of second nearest neighbour interactions—if I wants to be of opposite value to II, and III wants to be opposite to both, what can be put on III?

It could be as crude as a series of IF statements, each adding a different energy penalty to the system energy. It could be more complicated—like the use of bond valence sums [18] to determine whether one atom is better suited to a given site than another. The configurations may be treated as vectors and have an energy dependent on their dot product—for those familiar with magnetism this would be something like using a Heisenberg rather than an Ising Hamiltonian. It might be that if the vectors are parallel (dot product unity for unit vectors), the energy is low (or high) and the reverse if perpendicular (dot product zero). In [13] this is the sort of approach used, where the vector pointed to the NO_2 on a phenyl ring where the other five carbons all had chlorines attached.

A useful tactic here is swapping of occupancies. If the overall composition is known—which in general it will be from Bragg studies or from chemistry—then this can be maintained in the occupancy simulation by exchanging atoms or molecules. For example, go to one site, find the energy of whatever is there. Go to a second site which as a different atom/molecule/whatever and find its energy. Add the two to get, say, E_1 . Now interchange the atom types, and find the two energies again. Add them to get E_2. Now perform the MC step using E_1 and E_2 as 'before' and 'after'. This ensures that if the model had correct composition to begin, it will always maintain correct composition.

2.2.4 A Simple Ising Simulation

Below, the complete code of a simple MC simulation is shown. The program consists of two loops, an inner and an outer. The model is a simple two-dimensional square grid of atoms `A` and `B`. The is a 'model crystal'. The correlation is a simple first neighbour correlation, specified by the user (`corr1`). The interaction constant, w_{NN} (`weight` in the code) is estimated as 1. 100 MC cycles are performed, and the resulting correlation is calculated. The value of `weight` is then adjusted based on the calculated correlation compared to the desired, and this continues for 100 adjustments. Both these numbers are much larger than prob-

ably required. It should also be noted that the output file output2.txt is in the correct format for `ZMC` to read. It is a simple and verbose format, described formally in section 3.1.2.

`simpleI.f`, a simple program to use an Ising-like energy term to induce short-range order in an array of -1 and +1.

```
c A mimimal MC simulation to do nearest neghbour correlation
c on a square array.

  program occupancy_MC

  implicit none

  integer crystal(256,256),i,j,wrap(-(256-1):256*2)
  integer vec(4,2), swap(2),ia(2),jb(2)
        integer loop1,loop2
  real corr1, weight, rr(1), Enew, Eold, Ediff, corrcalc
  real rrr(2)

  write(6,*)' What is NN correlation?'
        read(5,*)corr1
  write(6,*)' Trying to obtain NN correlation',corr1
  weight = 10.0

c Initialise random number generator
  call rseed(12311,30037)

c Initialise crystal -- 50:50 -1 and +1
  do i=1,256
     do j=1,256
        call rannum(rr,1)
        if (rr(1).gt.0.5) then
     crystal(i,j)=-1
        else
     crystal(i,j)=1
        endif
     end do
  end do

  write(6,*)' Crystal Initialised'

c Set up boundary conditions
  j=0
  do i=-(256-1),256*2
     j=j+1
     wrap(i)=j
     if (j.eq.256) j=0
  end do

c Set up connectivity
```

```
      vec(1,1) = 1
      vec(1,2) = 0
      vec(2,1) = 0
      vec(2,2) = 1

      vec(3,1) = -1
      vec(3,2) = 0
      vec(4,1) = 0
      vec(4,2) = -1


      do loop1 = 1,100
         do loop2 = 1,100*256*256
c           Select two sites of different occupancy
            call rannum(rrr,2)
            ia(1) = int(256.0 * rrr(1)) + 1
            jb(1) = int(256.0 * rrr(2)) + 1
 101           call rannum(rrr,2)
            ia(2) = int(256.0 * rrr(1)) + 1
            jb(2) = int(256.0 * rrr(2)) + 1
            if(crystal(ia(1),jb(1)).eq.
     &           crystal(ia(2),jb(2))) goto 101
c           Calculate the initial energy, Eold
            call energy(crystal,ia,jb,wrap,vec,weight,Eold)
c           Swap the occupancies
            swap(1) = crystal(ia(1),jb(1))
            swap(2) = crystal(ia(2),jb(2))
            crystal(ia(1),jb(1)) = swap(2)
            crystal(ia(2),jb(2)) = swap(1)
c           Calculate the new energy, Enew
            call energy(crystal,ia,jb,wrap,vec,weight,Enew)
            Ediff = Enew - Eold
c           Do MC step
            if(Ediff.gt.0) then
         call rannum(rr,1)
         if(rr(1).gt.exp(-1.*Ediff))then
c           reject the MC move
            crystal(ia(1),jb(1)) = swap(1)
            crystal(ia(2),jb(2)) = swap(2)
         end if
            end if
         end do
c           Adjust the interaction constant
         call corr(crystal,vec,wrap,corrcalc)
         weight = weight + (corrcalc - corr1)
         write(6,*)loop1,corr1,corrcalc,weight
      end do

c Write out the results, twice!
      open(unit=1,file='output.txt')
```

```
  do i=1,256
     write(1,'(256i3)')(crystal(i,j),j=1,256)
  end do
  close(unit=1)

  open(unit=1,file='output2.txt')
  do i=1,256
    do j=1,256
     write(1,*)i,j,1,1,(crystal(i,j)+3)/2,1
    end do
  end do
  close(unit=1)
  stop

  end program occupancy_MC

c-----------------------------------------------------------------------

  subroutine corr(crystal,vec,wrap,corrcalc)

  implicit none

  integer crystal(256,256),i,j,k,wrap(-(256-1):256*2)
  integer vec(4,2)
  real corrcalc, S1, S2, sumS1,sumS2,sumS12,sumS22,sumS1S2,N

  N = 0.
  sumS1 = 0.
  sumS2 = 0.
  sumS12= 0.
  sumS22= 0.
  sumS1S2= 0.
  do i = 1,256
     do j = 1,256
        S1 = crystal(i,j)
c       Only look at neighbours on positive side to
c       avoid double counting.
        do k = 1,2
     S2 = crystal(wrap(i+vec(k,1)),wrap(j+vec(k,2)))
     N = N + 1.
     sumS1 = sumS1 + S1
     sumS2 = sumS2 + S2
     sumS12 = sumS12 + S1*S1
     sumS22 = sumS22 + S2*S2
     sumS1S2 = sumS1S2 + S1*S2
        end do
     end do
  end do
        corrcalc = (N*sumS1S2 - sumS1*sumS2)/
     &     sqrt((N*sumS12 - sumS1*sumS1)*(N*sumS22 - sumS2*sumS2))
```

```
  return

  end subroutine corr

c-----------------------------------------------------------------------

  subroutine energy(crystal,ia,jb,wrap,vec,weight,E)

  implicit none

  integer crystal(256,256),i,j,wrap(-(256-1):256*2)
  integer vec(4,2),ia(2),jb(2)
  real S1, S2, weight, E

  E = 0.
  do i = 1,2
     S1 = crystal(ia(i),jb(i))
     do j = 1,4
        S2 = crystal(wrap(ia(i)+vec(j,1)),wrap(jb(i)+vec(j,2)))
        E = E + weight * real(S1 * S2)
     end do
  end do
  return

  end subroutine energy

c-----------------------------------------------------------------------
```

A single MC step is as follows: A site is chosen at random. Its energy is calculated according to equation 2.10. A second site of opposite occupancy is selected, and its energy calculated. The sum of these is the 'initial energy', `Eold`. The two occupancies are exchanged and the energy is recalculated , `Enew`. If `Enew` < `Eold`, the new configuration is kept. If not, it may still be kept according to a simple temperature-dependent probability.

An MC cycle is a number of steps sufficient to visit each site in the model crystal once (or twice since the algorithm visits two sites per step in this case).

The program presented here is not the most efficient that can be imagined. Using 0 and 1 instead of -1 and 1 for the two values increases the speed of the correlation calculation, for example; but it is relatively simple to parse, and the correlation calculation is more generalisable.

Figure 2.8 shows two diffraction patterns, calculated from this model with (1) nearest neighbour (NN) correlation of -0.4 and (b) of +0.4. The calculation was performed using the code in appendices A, B and C. Cutting out the above code and compiling it, and running the results through the compiled version of that in the appendices will produce a binary output file. It is then processed with bin2gray to produce a portable gray map (sic) (a `.pgm` file) which can be viewed with many programs. ImageJ [21] is recommended. The code of the Fortran 77 version of bin2gray is shown in appendix D. This version assumes that the output from the DIFFUSE calculation is named `intensity.bin`.

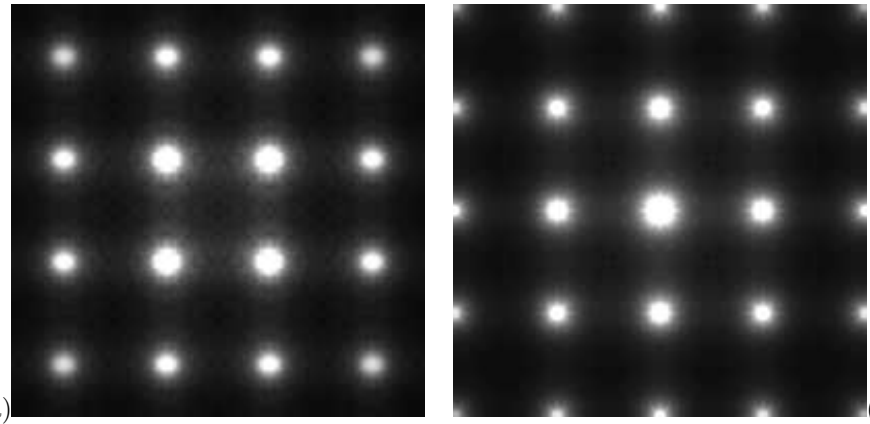
(a)  (b)

Figure 2.8: (a) Shows the diffraction pattern of a simple cubic binary alloy with correlations of -0.4 in the occupancies, and no other disorder. (b) Shows the pattern for correlations of +0.4; the spots here fall where the Bragg peaks would be, although they are much broader. The calculations were performed on the output of the code listed in section 2.2.4, using DIFFUSE.

Figure 2.9 shows the convergence of the simple Ising simulation. It plots the achieved correlation and the associated weight on the same axes, showing how the values oscillate and then settle down.

Figure 2.10 plots sections of the model crystals, (a) for negative correlation and (b) for positive. It shows clearly that in (a) atoms tend to alternate and in (b) they cluster, and these are the real-space pictures corresponding to the reciprocal space ones in figure 2.8. They were plotted simply by adding a line to the `readat.f` such that it outputted a simple text file of the form `x y S` where `S` is ± 1. This file was read into a spreadsheet, sorted on the basis of the `S` values and then two `y` versus `x` plots were superimposed, with different colour codings.

These are the 'real' equivalents of two of the more schematic diagrams shown in figure 2.5.

2.2.5 Further Comments on Occupancy Simulations

Once the occupancy model has produced its output file, this is read into `ZMC` and the model crystal is displacively relaxed. Size-effects are incorporated, the diffuse scattering patterns are calculated and the model can be evaluated. To change the occupancy correlations, change the w_{ij}. In general, as in section 2.2.4, the correlation structure is specified and an automatic adjustment of the w_{ij} is used to produce output with these correlations. An example is plotted in figure 2.9. Essentially, the MC program has an inner loop which equilibrates the model subject to the w_{ij}, and an outer loop which calculates the resulting correlations (most statistics text books will have the definition and computational formulae for correlation coefficients) and adjusts the w_{ij} through some simple algorithm (if correlation is too negative or positive, make its interaction constant larger or smaller (if need be change its sign) relative to T; often $T = 1$ and does no appear explicitly). By repeated adjustment eventually the model

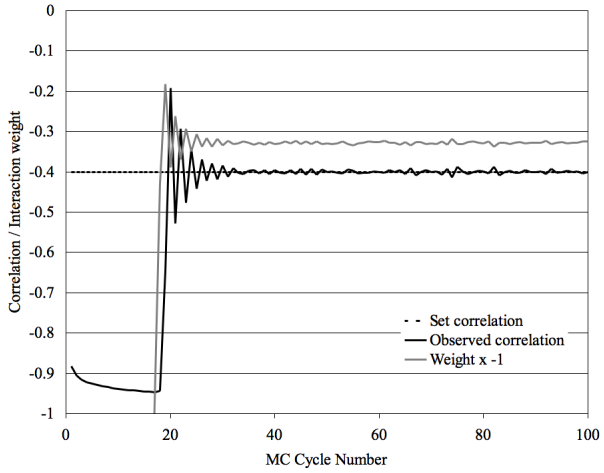
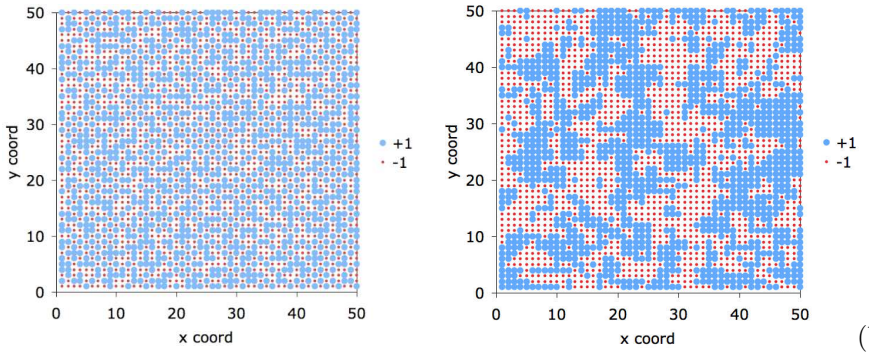


Figure 2.9: Shows the set nearest neighbour correlation (-0.4, dashed line), the actual correlation present in the model (black line) and the negative of the force constant w_{NN} as a function of MC cycle number for the 'simple Ising' model in section 2.2.4.



(a) (b)

Figure 2.10: The real space crystals generated by the simple Ising simulation. (a) Correlations of -0.4 in the occupancies, and no other disorder. (b) Correlations of +0.4.

has the occupancy correlations desired (if they are physically possible), and it is ready to be passed to ZMC. Of course the occupancy correlations desired may not match those present in the real system, so the whole process has to be refined, something noted in figure 2.2.

Note that often an occupancy model does not need to capture much 'reality' from the system. It can he highly abstract—nothing more than an array of +1 and -1 for example, with the geometry of the real system essentially incorporated into how these things interact. As long as an array of occupancy variables (molecule numbers and/or orientation numbers) is found and can then be impressed onto the displacive simulation in ZMC, it does not much matter how it came about.

Also, while this example has used integers, it is equally possible to use a correlation between real numbers. When this is done, the reason for using a more tedious correlation coefficient calculation than strictly necessary for the special case of ± 1 becomes apparent. In a molecular system, adjacent molecules might have a twist angle on a bond, or some other continuous quantity that you want to correlate. An example is *para*-terphenyl in section 4.3 [22, 23]. In that case one way of tackling the problem was to populate the twist angles from a distribution taken from the literature, and then swap twist angles using a similar algorithm to that discussed above, only with real numbers in it.

2.3 Displacements

As noted, ZMC is essentially a program for displacive simulations. This is a bigger task than it might seem because it tackles flexible molecules and occupancies. Leaving the occupancy simulation to be external seemed a reasonable compromise, given the need for flexibility.

In this section, I will simply show some sample displacive results that form a more complete picture when combined with the simple Ising model above; what I will show is just the same two-dimensional array, but under the following conditions:

1. (a) All atoms the same, connected to nearest neighbours by Hooke's law springs.

2. (b) As above, but with a slightly weaker spring also connecting to second nearest neighbours.

3. (c) Second neighbour interaction only.

4. (d) Two kinds of atoms, negatively correlated occupancies (see figure 2.10 and others above), no size effect.

5. (e) Two kinds of atoms, positively correlated occupancies (see figure 2.10 and others above), no size effect.

6. (f) Two kinds of atoms, negatively correlated occupancies (see figure 2.10 and others above), like atoms want to be 10% closer together than the average, unlike want to be 10% further apart.

7. (g) Two kinds of atoms, negatively correlated occupancies (see figure 2.10 and others above), big scatterers want to be 10% further apart than the average, small want to be closer together, big–small is the average.

8. (h) Two kinds of atoms, negatively correlated occupancies (see figure 2.10 and others above), small scatterers want to be 10% closer than the average, big want to be further apart. Big–small is the average.

So for each of these, a real-space and reciprocal-space picture (crystal versus diffraction pattern) can be shown.

Whole bunch of figures.

2.4 Calculating Diffuse Scattering

The calculation of diffuse scattering patterns from the simulated crystal is not central to this document, but it is appropriate to say a few words. ZMC is primarily designed to mate to DIFFUSE [3].

DIFFUSE calculates the diffuse scattering from a model crystal by Fourier transforming many small randomly chosen regions of the crystal (referred to as 'lots') and adding the resulting intensities. Transforming the whole model crystal results in high-frequency noise and edge-effects.

To use DIFFUSE a subroutine to give it the atomic positions and types has to be constructed. This reads in the files output from ZMC and converts the z-matrix-related numbers into atomic coordinates which are fed to DIFFUSE.

However, other programs exist, most particularly the DISCUS suite of programs, and they can also be used to calculate diffuse scattering plots. Hence ZMC has (or will have) an option to output files suitable for DISCUS to read.

Whether being read by DIFFUSE or DISCUS, they tend to be very big files, and ZMC tends to assume you have a fair bit of RAM and a large hard drive, by which I mean you may need to tailor the size of your model crystal to match your computing power. In 2004 we typically used a $32 \times 32 \times 32$ unit cells model crystal (for a system with say 120 atoms in a cell) but this scales up over time as computing power increases. A more significant issue can be that the Fortran compilers sometimes have limits on the sizes of the arrays they can allocate. Chapter 3 and section 3.3 will have more to say about this in a little more detail.

Chapter 3

The Programs

This section is really the 'instruction manual' part of the document, and as such the most crucial part. The previous sections discuss the problem that the program tackles, and the outlines of the method the program uses. This section indicates the commands used to implement models and generate diffuse scattering plots. The next chapter outlines process to go through to implement a model.

Here we talk about formats of input files, sensible values to put into those input files, the suite of ancillary programs needed to sensibly tackle the problem.

The main program is `ZMC`. It began in frustration at having to write a new program to tackle every problem. For example, the original work done on benzil, $C_{14}H_{10}O_2$, was done using a custom-written program [24, 25], partly written by Prof. W. I. F. David, to whom I am indebted and who is in no way responsible for any idiocies and misinterpretations I might have introduced! Later studies of benzil and other compounds have used (various early versions of) `ZMC`, inlcuding buut not limited to [26, 27, 17, 13, 28, 29, 20, 30, 31, 22, 23, 32].

Generally, getting a sensible working simulation using `ZMC` has proved much quicker than writing a new program would have, with a working simulation being ready in a few hours of intensive work. Having said that it is a very large step from a working simulation (in the sense that the program runs and produces output) to a good model.

As stated, `ZMC` does not do occupancy simulations, so some words were expended in section 2.2.3 a typical program for implementing these. However, even if the problem does not require an occupancy simulation, before `ZMC` can be used one must generate z-matrices, lists of contact vectors, diagnostic plots and potentially other bits and pieces, so there is a range of programs required.

Generally, the order will be:

1. Get a good `.cif` file for your structure. This may come from a database, or may come from your own conventional single-crystal Bragg scattering experiments.

2. Identify which atoms belong to a single molecule. This is easy when you do not have a disordered structure with overlapping molecules, but can be non-trivial when you are trying to extract actual molecules from blurry electron density (as given by X-rays) or scattering-length density (for example neutron diffraction). It may be necessary to split the `.cif`

into several files, one for each overlapped molecule. Can you take the mlecular shape from some other structure and disorder it an put it into your cell?

3. Think about the internal flexibility you want the molecules to have, and reorder the atoms such that the z-matrix idea will allow you to implement those. That means, as discussed in section 2.2.1, that the molecule does not get 'broken' if a twist is allowed around a bond, for example.

4. Think about whether ohne or more 'dummy' atoms might be useful, either in defining the internal degrees of freedom or in defining the molecular origin, such that the origin is on a centre of symmetry for example. This latter may not be strictly necessary, but can aid in making the model easier to 'debug'.

5. Once you have the `.cif` file(s) you need, run zmat_maker to generate the z-matrices you will need, along with the quaternions and molecular origin coordinates. These numbers should come out in a file of the correct format; but if you have had to split up the `.cif` because of overlapping molecules, some manual reassembly may be required.

6. If the crystal shows occupancies (see section 2.2.3), it is a good idea at this point to set up a representative occupancy structure. This could be random at this stage (and a very simple program exists for creating such a random occupancy file, `make_random_occ.f90`), but it should be indicative of what molecules in what orientations are allowed to occur at what places in the unit cell. These places are referred to as 'locations' in `ZMC` to differentiate them from 'sites', because 'site' is always taken to refer to an atomic position, whereas location is used to refer to a molecular position (which may be occupied by one of a number of molecules in one of a number of orientations, or by a vacancy, but can only contain one of these things at a time).

7. You can now set up the part of the input files pertaining to the crystal geometry, but you need a set of contact vectors (see section 2.2.2). `ZMC` has a subroutine that can be called on the command line for doing this (see section sec:zmc:contacts), but there is also a program with a graphical interface, courtesy of Dr A. P. Heerdegen which does the same job more interactively. See section 3.4.

8. Classifying contact vectors into groups, each of which will have the same size-effect (see section 2.2.2) and force constant. The simplest way to do this is to simply make all symmetry-equivalent vectors of the same type (doing otherwise is not really defensible). Symmetry inequivalent vectors that you want to give the same parameters to can either be given the same type number, or just given the same parameters in the input file; the latter is to be preferred as it is more flexible, and the input file format is designed to make the information compact.

9. Define the internal degrees of freedom you want the molecule to have.

10. Decide on various other parameters that are explained below. Things like temperature, force constants, size-effects, sizes of random variable shifts and so on.

11. Set up the input files for DIFFUSE to allow calculation of the diffraction patterns (cuts through reciprocal space) that are desired.

12. Run the program

13. Puzzle over the results.

Something for the future would be to arrange a simple means of writing out the simulation as far as it has got. Something like creating a small file that it tests for each MC cycle and aborting if the file 'STOPZMC' exists.

3.1 ZMC

ZMC is really what this document is about. It has built-in significant help, obtained by running ZMC −−help and ZMC −−help2, shown in appendices E and F). The notes below should be examined in conjunction with the detailed example simulations outlined in the following chapter.

3.1.1 The ZMC command line

This section outlines the fields on the command line, the command line options and filenames that need to be given in the various modes in which ZMC can be run. ZMC has three main modes of operation. (1) Generate a list of contact vectors according to some simple rules supplied by the user (−−getcontacts). (2) Plot the model crystal (crudely) (−−plot). (3) Do an MC simulation (neither of the above is given).

All these require differing amounts of information to be available in the input file.

Here is an extract from the built-in help:

```
 |-----------------------------------------------------------------------|
 | Usage:                                                                |
 |                                                                       |
 | ZMC [--option_1] [--option_2] ... [--option_n] infile [outfile]       |
 |                                                                       |
 | infile contains the parameters and additional filenames to run        |
 | the MC simulation.                                                    |
...
etc
...
 |-----------------------------------------------------------------------|
```

−−getcontacts: For contact vector calculation, ZMC needs a unit cell, z-matrix, .qxyz file and some information on which atoms to connect to and what length limits to use.

Refer to appendices E and F) for the gritty detail. I note here that it may be useful to generate a relatively small subset of vectors at a time then combine them into a final list; or not, if you want an exhaustive list.

−−plot: For plotting, , the user is asked some simple interactive questions and a garishly coloured PostScript plot (or in fact set of plots) is the result. Filenames are hard-wired in and so care must be taken to avoid over-writing

previous plots. The plotting is supposed to use as much information as is given; it will need the model crystal dimensions (keyword CELL) and if initial randomness is specified, it will incorporate it into the plot (and mention this as it does so). The plotting routine is acknowledged as crude and limited, with many choices hard-wired in. It is a simple incorporation of what was an *ad hoc* simulation testing and diagnostic tool into the main ZMC code, and this shows; at present there is no intention of developing this part of the program beyond minor refinements and big fixes.

Other command line arguments that could use a little more discussion include:

−−crystal: Causes the simulation to output a `.crystal` file, that contains the variables for each z-matrix; in other words, writes out **x**, **q** and **i** for each molecule in the model. In conjunction with the z-matrix itself and the MC input file, this provides enough information to resume the simulation.

The first few lines of such a file might be:

```
DCDNB from the CSD
1 1 1 1  5.572  2.986  5.257  0.160  0.864  0.424  0.215 F -0.41615 -2.17374
1 1 1 2  1.878  6.279  1.801  0.424 -0.215 -0.160  0.864 F -1.15394 -2.31949
1 1 1 3 -0.260  3.599  8.859  0.160  0.864  0.424  0.215 T -0.23322 -1.93374
1 1 1 4  3.433  0.306 12.315  0.424 -0.215 -0.160  0.864 T -0.18090 -2.12341
1 1 2 1  5.572  2.986  5.257  0.160  0.864  0.424  0.215 F -0.20370 -2.88468
1 1 2 2  1.878  6.279  1.801  0.424 -0.215 -0.160  0.864 F -0.13101 -2.54319
...
etc
...
```

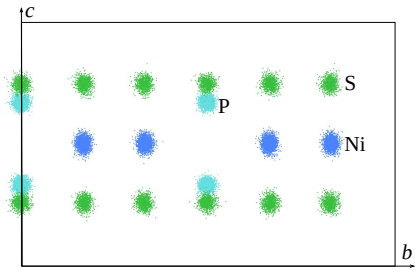
Where plainly we have a header, then many lines, each of the format: unit cell indices (3 integers), location number , one integer, **x** (3 reals), **q** (4 reals and one character) and **i** (some number of reals, possibly zero). The character in the quaternion indicates whether the rotation it represents is 'improper' — if it is improper, the flag is set to `T`.

This file may also be useful in interrogating the simulation and drawing out its conclulsions.

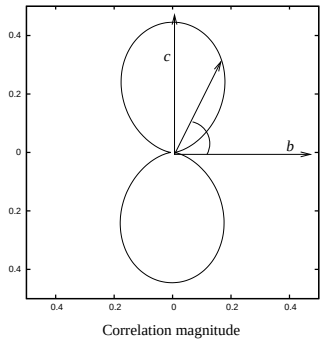
−−diffuse: Causes ZMC to put out a file suitable for the accompanying diffuse scatteing calculation program.

−−pairs: This goes to a molecule, writes its variables to a file (as for the −−crystal option, more or less) but then also goes to the molecule at the end of each contact vector and writes out its variables. The result is a series of files with names `pairs_L_C_outfile.out` where L is the location number of the origin molecule C is contact number. Many of these files will be duplicates. Why? Because such a file can then be examined for correlations between adjacent molecules, exactly the sort of result diffuse scattering is supposed to give. For a simple example, you could simply calculate (in a spreadsheet for example) the correlation coefficient between the x coordinates of adjacent molecules to see if they are 'pushing' each other along in the x direction, and compare this to other correlations.

−−fracs, −−cartsn, −−cartst: Causes the simuation to write out in simple format the x, y and z coordinates of all atoms in the simulation, plus their indexing information (cell, location, z-matrix, etc). Useful for then plotting, (for example figure 3.1a) or exploring for correlations or local structure.



(a)



(b)

Figure 3.1: (a) A scatter plot of (a subset of) the atom positions in the model crystal of $NiPS_3$ [33]. (b) A diagram representing the correlations of the atomic displacements in the *bc* plane of S atoms in neighbouring layers. These figures show that the distributions of displacements are relatively isotropic in the *bc* plane,even though the *correlations* in the displacements are not.

−−corro, −−corra: Plots the correlation coefficient of the component of molecular displacement along a given direction on adjacent molecules as a function of the direction. Best seen in an example, in this case from [33], figure 3.1b.

−−energy: Outputs the MC energy of each molecule to `outfile.energy`. Just how meaningful the MC energy is is anyone's guess.

−−cif: Outputs a crude `.cif` file to `outfile.cif`. Probably a deeply imperfect `.cif`, especially in terms of the anisotropic ADPs, but potentially useful even so. A brand new routine to perform this task is a desirable refinement in future.

3.1.2 Description of the input files

This section outlines the formats of the various input files, gives examples for the various modes in which `ZMC` can be run, and tries to indicate possible pitfalls.

3.1.3 Generating contact vectors

This section discusses using `ZMC` to generate a list of contact vectors (see section 2.2.2).

3.1.4 Exploring the results

This section discusses using `ZMC` to output some information about the resulting simulation.

3.2 zmat_maker

3.3 DIFFUSE

3.4 jContacts

Aidan?

Chapter 4

Detailed Examples

4.1 A minimal simulation in ZMC

It seems sensible to outline, perhaps briefly, a minimal implementation of a simulation in ZMC. The simplest unit cell is a simple cubic with a single atom at the origin.

The z-matrix here becomes very simple, and indeed can be constructed by inspection:

```
d:\zmc_win7\nothing>cat n1.zma
Random header
1
C 0   0.00 0 0.00 0 0.00
```

And similarly, the .qxyz file:

```
d:\zmc_win7\nothing>cat N1.QXY
1 1 1 1 0.0 0.0 0.0 1.0 F 0.0 0.0 0.0
```

Immediately we can generate some contact vectors, with a simple input file:

```
d:\zmc_win7\nothing>cat MC_INPUT.INP
HEADER Minimal
ZMATFILE 1 n1.zma
QXYZFILE 1 n1.qxy
CELL 3.0 3.0 3.0 90.0 90.0 90.0
!!
VMIN 2.0
VMAX 10.0
CONTATOMS ZMAT 1 1
NEWCONTACTS minimal.con
```

Which gives an output (sorted and classified on length) like this:

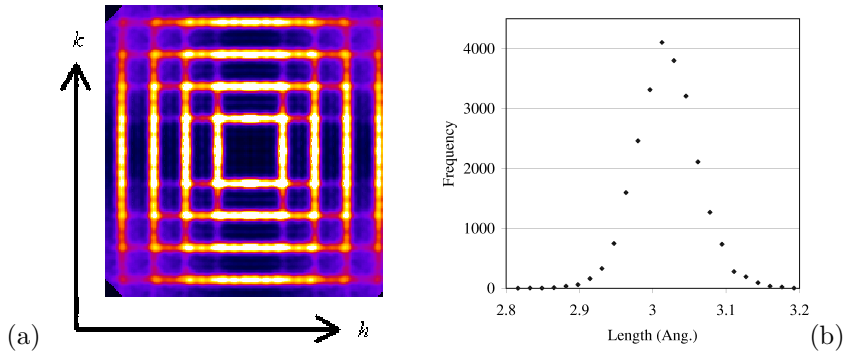
```
ol   oz   om   oat     da   db   dc   dl   dz
dm   dat     length  type
    1    1    1    1   -1    0    0    1    1    1    1    3    1
```

```
    1    1    1    1    0   -1    0    1    1    1    1    3    1
    1    1    1    1    0    0   -1    1    1    1    1    3    1
    1    1    1    1    0    0    1    1    1    1    1    3    1
    1    1    1    1    0    1    0    1    1    1    1    3    1
    1    1    1    1    1    0    0    1    1    1    1    3    1
    1    1    1    1   -1   -1    0    1    1    1    1 4.24264    2
    1    1    1    1   -1    0   -1    1    1    1    1 4.24264    2
    1    1    1    1   -1    0    1    1    1    1    1 4.24264    2
    1    1    1    1   -1    1    0    1    1    1    1 4.24264    2
    1    1    1    1    0   -1   -1    1    1    1    1 4.24264    2
    1    1    1    1    0   -1    1    1    1    1    1 4.24264    2
    1    1    1    1    0    1   -1    1    1    1    1 4.24264    2
    1    1    1    1    0    1    1    1    1    1    1 4.24264    2
    1    1    1    1    1   -1    0    1    1    1    1 4.24264    2
    1    1    1    1    1    0   -1    1    1    1    1 4.24264    2
    1    1    1    1    1    0    1    1    1    1    1 4.24264    2
    1    1    1    1    1    1    0    1    1    1    1 4.24264    2
    1    1    1    1   -1   -1   -1    1    1    1    1 5.19615    3
    1    1    1    1   -1   -1    1    1    1    1    1 5.19615    3
    1    1    1    1   -1    1   -1    1    1    1    1 5.19615    3
    1    1    1    1   -1    1    1    1    1    1    1 5.19615    3
    1    1    1    1    1   -1   -1    1    1    1    1 5.19615    3
    1    1    1    1    1   -1    1    1    1    1    1 5.19615    3
    1    1    1    1    1    1   -1    1    1    1    1 5.19615    3
    1    1    1    1    1    1    1    1    1    1    1 5.19615    3
    1    1    1    1   -2    0    0    1    1    1    1    6    4
    1    1    1    1    0   -2    0    1    1    1    1    6    4
...
etc
...
    1    1    1    1    3    1   -1    1    1    1    1 9.94987   10
    1    1    1    1    3    1    1    1    1    1    1 9.94987   10
```

And in this case it is easy to picture the vectors without doing any plots, since plainly first neighbour ('NN'), second neighbour ('2NN') and so on are pretty simple on a simple cubic cell.

So the first MC uses the folllowing input file:

```
d:\zmc_win7\nothing>cat MC_INPUT.INP
HEADER Minimal
CONTACTFILE minimal_con_sort.prn
CRYSTAL 20 20 20
TEMPERATURE 1.0
ZMATFILE 1 n1.zma
QXYZFILE 1 n1.qxy
MCCYCLES 100
SPRCON 0.0
SPRCON 1.0 1
XYZINITW 0.3
QINITW 0.0
```



(a) (b)

Figure 4.1: (a) $hk0$ cut for this 'minimal' model, nearest-neighbour only interactions, aiming for a global B_{iso} of 1.0. (b) The histogram of lengths for contact vectors of type 1; output from the −−summary option of ZMC.

```
XYZWIDTH 0.1
BADJUST 1 1.0
INCUPDATE 1
CELL 3.0 3.0 3.0 90.0 90.0 90.0
!!
VMIN 2.0
VMAX 10.0
CONTATOMS ZMAT 1 1
NEWCONTACTS minimal.con
```

So in other words only the NN interaction is non-zero. We expect this to simply mean that atoms will have their motions correlated in longitudinal chains — they will 'push and pull' on each other in the directions of their motions. A chain of correlated atoms will be like a small one dimensioanl (1d) crystal embedded in the larger crystal. If a 3d corrlated crystal collapses the scattering into 0 dimennsional Bragg spots, a 1d crystal will collapse it into 2d sheets of intensity. Figure 4.1 shows the $hk0$ cut of the reciprcal space of this model; the strong lines of scattering are indeed sections through planes. Figure 4.1a. And we can see the resulting distribution of contact lengths for this interaction (or any other) by examining the output of the −−summary switch. Figure 4.1b.

We can then turn on 2NN instead of NN by changing SPRCON

```
SPRCON 1.0 1
```

to

```
SPRCON 1.0 2
```

and so on. We can allow force constants on NN and 2NN to be non-zero together

```
SPRCON 1.0 1 2
```

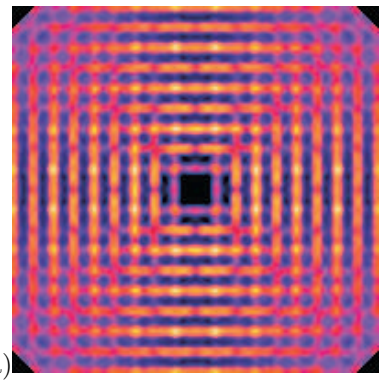
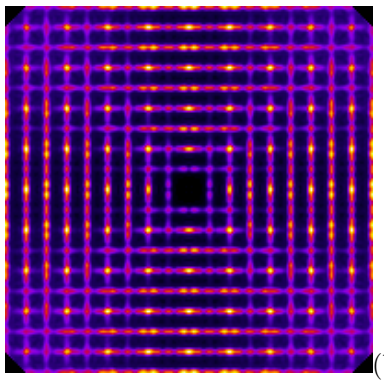
(a)   (b)

Figure 4.2: (a) $hk0$ cut for this 'minimal' model, nearest-neighbour only interactions, aiming for a global B_{iso} of 1.0. (b) The same but with 1000 MC cycles and 120 lots of size $9 \times 9 \times 9$ instrad of 100 MC cycles and 20 lots of $5 \times 5 \times 5$ unit cells.

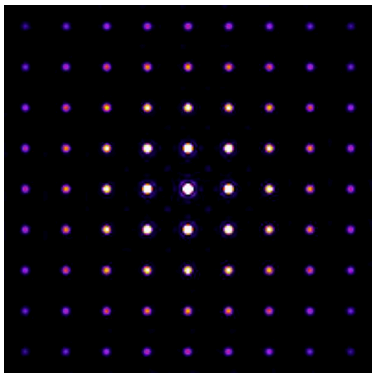


Figure 4.3: Bragg spot calculation.

Or do silly things like have neighbours 2 unit cells away interacting but not the nearest nieghbours. Neighbours two cells away will be 6Å away, which is contact vector type 4. This would seem to be likely to result in little 1d 'crystals' of lattice parameter 6Å embedded in the model, and we might expect it to give planes of scattering therefore at half-integer positions. Does it? Figure 4.2a would suggest it does. We may also want to look at the effect of number of lots, lot sizes, and number of MC cycles. As one example, (b) shows the same calcualtion except with 1000 MC cycles and 120 lots of size $9 \times 9 \times 9$. Qualitatively the same, but different in detail, and this must be born in mind when quantitatievly comparing calculation with experiment.

As a note, the Bragg spots are shown in figure 4.3.

Lastly, even though we are not looking at occupancy, we can look at size-effect by messing with the SIZE keyword.

The 'standard' MC input file for this excursion is:

```
HEADER Minimal
```

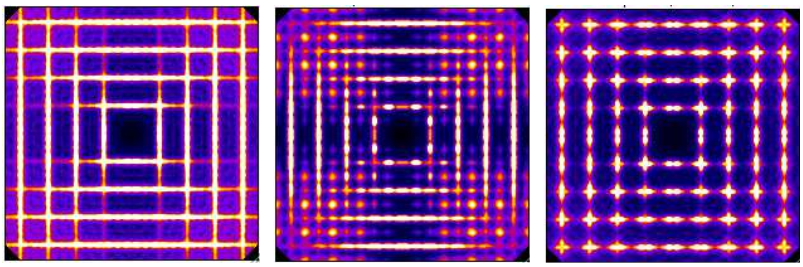


Figure 4.4: Some exploration of size-effect. See table 4.1.

```
CONTACTFILE minimal_con_sort.prn
CRYSTAL 20 20 20
TEMPERATURE 1.0
ZMATFILE 1 n1.zma
QXYZFILE 1 n1.qxy
MCCYCLES 200
SPRCON 0.0
SPRCON 200.0 1 4
!!SIZE 0.3 1
!!SIZE -0.3 4
XYZINITW 0.1
QINITW 0.0
XYZWIDTH 0.1
BADJUST 1 1.0
INCUPDATE 1
CELL 3.0 3.0 3.0 90.0 90.0 90.0
```

Where possible size-effect terms have been commented ('!!') out. The combinations oulined in table 4.1 were explored.

Table 4.1: Size=effects explored. See figure 4.4.

Label	s.e. on 1	s.e. on 4
(a)	0	0
(b)	0.3	-0.3
(c)	-0.3	0.3

And the associated patterns are shown in figure 4.4.

I note here that the bin2gray command used for all of these was effectively

$$> bin2gray –hmirror –vmirror –norm=NORM –rotave=4 FILENAME

where NORM was chosen to be just a little bigger than the maximum value (as revealed by a premilinary bin2gray run without specifying a normalisation value) and that the symmetry before the rotational averaging was not always perfect; this is often improved by using 'e' or 'E' for Bragg spot removal. The diffuse.hk0 file used for these calculations typcally looked like this:

```
'minimal'
12645 9676
3.0 3.0 3.0 0.0 0.0 0.0
20 20 20
Y
-4.5 -4.5 0.0
 4.5 -4.5 0.0 200
-4.5  4.5 0.0 200
-4.5 -4.5 0.0  1
1.0
9 9 9
20
1
1
Y
'C '
2.31 20.844 1.02 10.2075
1.5886 0.5687 0.865 51.65 0.2156
0.0 0.0
```

4.2 DCDNB 1

This molecule was chosen pseudorandonly from the database and implemented. The aim is to walk through the setting-up, making some common errors and showing the range of tools used to put a simulation toegther. It is more realistic than the simulation in 4.1 and correspondingly more complicated, although still simple by many standards; for example it does not deal with occupancies.

So here we are setting up a simulation of DCDNB using `ZMC`; at this point I am just getting something to run, not worrying about whether it is 'right' as far as the SRO in DCDNB is concerned. I will use `$$>` to indicate a command line prompt.

Here is the `.cif` as downloaded from the database, `METXOZ_CCDC_633654.cif`:

```
##############################################################################
#
#                 Cambridge Crystallographic Data Centre
#                                 CCDC
#
##############################################################################
#
#  This CIF contains data generated directly from one or more entries in
#  the Cambridge Structural Database and will include bibliographic,
#  chemical, crystal, experimental, refinement, and atomic coordinate data,
#  as available.
#
#  Copyright  2011  The Cambridge Crystallographic Data Centre
#
#  This CIF is provided on the understanding that it is used for bona fide
#  research purposes only.  It may contain copyright material of the CCDC
```

```
#  or of third parties, and may not be copied or further disseminated in
#  any form, whether machine-readable or not, except for the purpose of
#  generating routine backup copies on your local computer system.
#
#  For further information about the CCDC, data deposition and data
#  retrieval see <www.ccdc.cam.ac.uk>. Bona fide researchers may freely
#  download  Mercury and enCIFer from this site to visualise CIF-encoded
#  structures and to carry out CIF format checking respectively.
#
##############################################################################

data_CSD_CIF_METXOZ
_audit_creation_date 2007-02-13
_audit_creation_method CSD-ConQuest-V1
_database_code_CSD METXOZ
_database_code_depnum_ccdc_archive 'CCDC 633654'
_chemical_formula_sum 'C6 H2 Cl2 N2 O4'
_chemical_formula_moiety
;
C6 H2 Cl2 N2 O4
;
_journal_coeditor_code "IUCr CI2241"
_journal_coden_Cambridge 1370
_journal_volume 63
_journal_year 2007
_journal_page_first o177
_journal_name_full 'Acta Crystallogr.,Sect.E:Struct.Rep.Online '
loop_
_publ_author_name
"Zhong-Hua Luo"
"Hong-Jun Zhu"
"Shui-Ping Deng"
"Hong-Sheng Jia"
"Shan Liu"
_chemical_name_systematic
;
1,5-Dichloro-2,4-dinitrobenzene
;
_cell_volume    891.623
_exptl_crystal_colour 'yellow'
_exptl_crystal_density_diffrn 1.766
_exptl_crystal_description 'Needle'
_exptl_crystal_preparation 'ethanol'
_diffrn_ambient_temperature 298
#These two values have been output from a single CSD field.
_refine_ls_R_factor_gt 0.0606
_refine_ls_wR_factor_gt 0.0606
_symmetry_cell_setting monoclinic
_symmetry_space_group_name_H-M 'P 21/c'
_symmetry_Int_Tables_number 14
```

```
loop_
_symmetry_equiv_pos_site_id
_symmetry_equiv_pos_as_xyz
1 x,y,z
2 -x,1/2+y,1/2-z
3 -x,-y,-z
4 x,-1/2-y,-1/2+z
_cell_length_a 9.5900(19)
_cell_length_b 6.5860(13)
_cell_length_c 14.751(3)
_cell_angle_alpha 90
_cell_angle_beta 106.86(3)
_cell_angle_gamma 90
_cell_formula_units_Z 4
loop_
_atom_type_symbol
_atom_type_radius_bond
C 0.68
H 0.23
Cl 0.99
N 0.68
O 0.68
loop_
_atom_site_label
_atom_site_type_symbol
_atom_site_fract_x
_atom_site_fract_y
_atom_site_fract_z
Cl1 Cl 0.86836(16) 0.8072(2) 0.51157(9)
Cl2 Cl 0.52778(16) 0.1606(2) 0.42287(10)
O1 O 0.6239(4) -0.0393(5) 0.2774(3)
O2 O 0.6192(4) 0.1645(6) 0.1615(3)
O3 O 0.9089(5) 0.7816(9) 0.2416(3)
O4 O 1.0529(4) 0.7729(7) 0.3821(3)
N1 N 0.6402(4) 0.1278(6) 0.2447(3)
N2 N 0.9417(4) 0.7272(7) 0.3233(3)
C1 C 0.8048(5) 0.6115(7) 0.4339(3)
C2 C 0.8413(4) 0.5881(7) 0.3509(3)
C3 C 0.7831(4) 0.4338(7) 0.2888(3)
H1 H 0.80500 0.42330 0.23160
C4 C 0.6926(4) 0.2955(6) 0.3121(3)
C5 C 0.6525(5) 0.3152(6) 0.3944(3)
C6 C 0.7088(5) 0.4764(7) 0.4544(3)
H2 H 0.68120 0.49340 0.50940
#END
```

A labelled picture generated by Mercury is in shown figure 4.5.

First thing to do is reorder the atoms. (1) We want a convenient origin. Probably a good place is the centre of the phenyl ring; and if we decide that we want a 'π' electron to interact with, this 'dummy' atom is a useful surrogate.

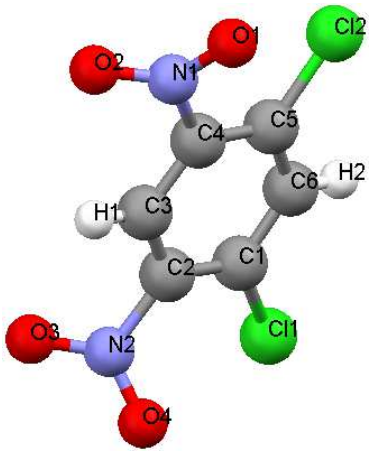


Figure 4.5: Single molecule of DCDNB generated in Mercury from the `.cif` file in `METXOZ_CCDC_633654.cif` (in section 4.2).

(2) We want to make sure that the NO_2 groups can rotate without carrying the whole molecule with them.

So we work out the average of all the C atom positions and put an 'x' atom there.

Now the molecule looks like that shown in figure 4.6

And the `.cif` file looks like (just the changed bit)

```
x1 H 0.7471833 0.453416  0.3724166667
C3 C 0.7831(4) 0.4338(7) 0.2888(3)
H1 H 0.80500 0.42330 0.23160
C2 C 0.8413(4) 0.5881(7) 0.3509(3)
C1 C 0.8048(5) 0.6115(7) 0.4339(3)
Cl1 Cl 0.86836(16) 0.8072(2) 0.51157(9)
C6 C 0.7088(5) 0.4764(7) 0.4544(3)
H2 H 0.68120 0.49340 0.50940
C5 C 0.6525(5) 0.3152(6) 0.3944(3)
Cl2 Cl 0.52778(16) 0.1606(2) 0.42287(10)
C4 C 0.6926(4) 0.2955(6) 0.3121(3)
N1 N 0.6402(4) 0.1278(6) 0.2447(3)
O1 O 0.6239(4) -0.0393(5) 0.2774(3)
O2 O 0.6192(4) 0.1645(6) 0.1615(3)
N2 N 0.9417(4) 0.7272(7) 0.3233(3)
O3 O 0.9089(5) 0.7816(9) 0.2416(3)
O4 O 1.0529(4) 0.7729(7) 0.3821(3)
```

And so now we use Mercury to pack out the unit cell (Calculate -¿ Packing/Slicing), shown in the figure 4.7 the cell.

And we can output this to an editable text file, Using the mol2 format (File -¿ Save As) which I will call `dcdnb_cq1_METXOZ.mol2`, and it looks like this:

```
@<TRIPOS>MOLECULE
```

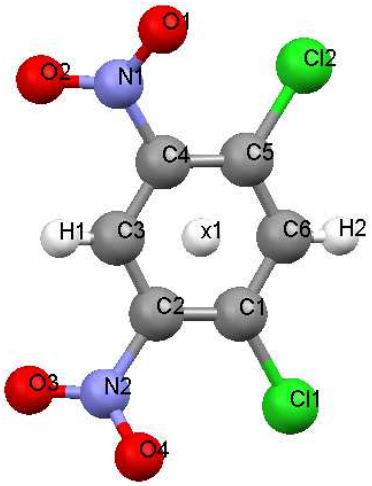


Figure 4.6: Single molecule of DCDNB generated in Mercury from the `.cif` file in `METXOZ_CCDC_633654.cif`, with dummy atom.

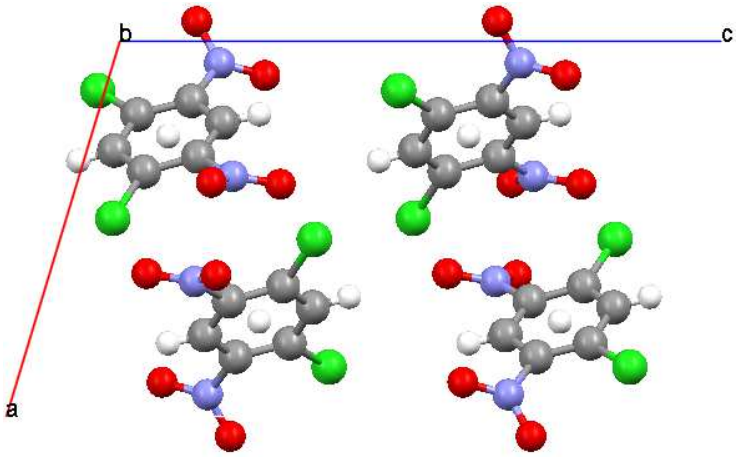


Figure 4.7: A unit cell of DCDNB packed in Mercury, showing the four molecules (Z=4), with dummy atom.

```
METXOZ
    68    64     8     0     0
SMALL
NO_CHARGES
****
Generated from the CSD

@<TRIPOS>ATOM
     1 x1       5.5722   2.9862   5.2574   H         1 RES1   0.0000
     2 x1       1.8787   6.2792   1.8011   H         2 RES2   0.0000
     3 x1      -0.2605   3.5998   8.8596   H         3 RES3   0.0000
     4 x1       3.4330   0.3068  12.3159   H         4 RES4   0.0000
     5 C3       6.2744   2.8570   4.0770   C.2       5 RES5   0.0000
     6 H1       6.7291   2.7879   3.2695   H         5 RES5   0.0000
     7 C2       6.5668   3.8732   4.9536   C.2       5 RES5   0.0000
     8 C1       5.8617   4.0273   6.1253   C.3       5 RES5   0.0000
     9 Cl1      6.1389   5.3162   7.2218   Cl        5 RES5   0.0000
    10 C6       4.8533   3.1376   6.4147   C.2       5 RES5   0.0000
    11 H2       4.3533   3.2495   7.1912   H         5 RES5   0.0000
    12 C5       4.5701   2.0759   5.5677   C.3       5 RES5   0.0000
    13 Cl2      3.2522   1.0577   5.9696   Cl        5 RES5   0.0000
    14 C4       5.3068   1.9462   4.4059   C.2       5 RES5   0.0000
    15 N1       5.0926   0.8417   3.4544   N.3       5 RES5   0.0000
    16 O1       4.7964  -0.2588   3.9160   O.2       5 RES5   0.0000
    17 O2       5.2472   1.0834   2.2799   O.2       5 RES5   0.0000
    18 N2       7.6477   4.7893   4.5640   N.3       5 RES5   0.0000
...
etc
...
    67 O3       5.5436  -1.8546  10.4691   O.2       8 RES8   0.0000
    68 O4       6.3234  -1.7973  12.4526   O.2       8 RES8   0.0000
@<TRIPOS>BOND
     1     5     6    1
     2     5     7   un
     3     5    14   un
     4     7     8    1
     5     7    18    1
     6     8     9    1
     7     8    10    1
...
etc
...
    63    66    67    1
    64    66    68    1
@<TRIPOS>SUBSTRUCTURE
     1 RES1        1 GROUP            0 ****  ****    0
     2 RES2        2 GROUP            0 ****  ****    0
     3 RES3        3 GROUP            0 ****  ****    0
     4 RES4        4 GROUP            0 ****  ****    0
     5 RES5        5 GROUP            0 ****  ****    0
```

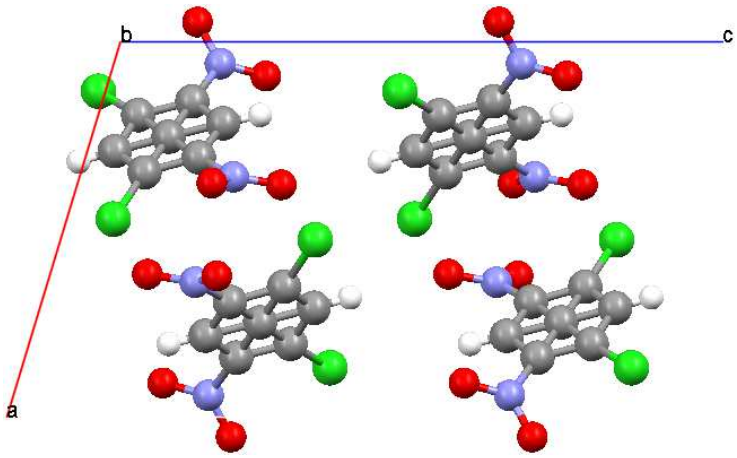


Figure 4.8: A unit cell of DCDNB packed in Mercury, showing the four molecules (Z=4), with dummy atom now bonded.

```
     6 RES6        21 GROUP             0 ****  ****    0
     7 RES7        37 GROUP             0 ****  ****    0
     8 RES8        53 GROUP             0 ****  ****    0
@<TRIPOS>CRYSIN
    9.5900    6.5860   14.7510   90.0000  106.8600   90.0000    14     1
```

But we have a problem here; Mercury is putting the new x1 atom in a separate molecule to the DCDNB. That is because the H atom has too short a bonding distance and is not considered to 'belong' to the molecule; call it a C atom instead and we get figure 4.8 —note the extra bonds to the centre atom.

Now the `.mol2` is:

```
@<TRIPOS>MOLECULE
METXOZ
    68    88     4     0     0
SMALL
NO_CHARGES
****
Generated from the CSD

@<TRIPOS>ATOM
     1 x1        5.5722   2.9862   5.2574   C.3        1 RES1   0.0000
     2 C3        6.2744   2.8570   4.0770   C.3        1 RES1   0.0000
     3 H1        6.7291   2.7879   3.2695   H          1 RES1   0.0000
     4 C2        6.5668   3.8732   4.9536   C.3        1 RES1   0.0000
     5 C1        5.8617   4.0273   6.1253   C.3        1 RES1   0.0000
     6 Cl1       6.1389   5.3162   7.2218   Cl         1 RES1   0.0000
     7 C6        4.8533   3.1376   6.4147   C.3        1 RES1   0.0000
...
etc
...
```

```
    68 O4       6.3234  -1.7973  12.4526   0.2       4 RES4   0.0000
@<TRIPOS>BOND
     1     1     2    1
     2     1     4    1
     3     1     5    1
     4     1     7    1
     5     1     9    1
...
etc
...
    86    63    65    1
    87    66    67    1
    88    66    68    1
@<TRIPOS>SUBSTRUCTURE
     1 RES1        1 GROUP             0 ****  ****    0
     2 RES2       18 GROUP             0 ****  ****    0
     3 RES3       35 GROUP             0 ****  ****    0
     4 RES4       52 GROUP             0 ****  ****    0
@<TRIPOS>CRYSIN
    9.5900    6.5860   14.7510   90.0000  106.8600   90.0000    14      1
```

But we are not done yet; now we need to control the connectivity. We want the centre atom to *only* connect to the atom we want it to. I want to imagine for my own mental picture the molecule rotating around the vector connecting H atoms, so I want to force the first bond to be from x1 to C3. This is from atom 1 to atom 2. That means we have to remove all the other bonds to atom 1. But we do not need to renumber, since mol2 allows duplicate bonds. Further, zmat_maker only uses the first molecule to define the z-matrix, so we don't need to do this manipulation for molecule 2 to 4. So now we get this for the relevant bit of the bond section of the `.mol2` file:

```
@<TRIPOS>BOND
     1     1     2    1
     2     1     2    1
     3     1     2    1
     4     1     2    1
     5     1     2    1
     6     1     2    1
     7     2     3    1
     8     2     4    1
     9     2    11    1
    10     4     5    1
    11     4    15    1
    12     5     6    1
    13     5     7    1

...
etc
...
```

And now we can try to run zmat_maker—but can you spot the deliberate mistake?

Correct! The first three atoms are all in a line and so cannot define a plane and so cannot define a dihedral angle, needed to define the position of the fourth atom. So we need to go back to the `.cif` and renumber it with say C2 as the third atom.

So now we go back to Mercury and pack out the cell again. And then we have to sort out the connectivity again (fun, eh?).

The `.mol2` file with modified connectivity may not read back into Mercury successfully, but will be OK in zmat_maker. So the output of running zmat_maker looks like this:

$$> zmat_maker dcdnb_cq1_METXOZ.mol2

```
  Sub structure: 1
       Z-matrix: 1
     Quaternion:    0.160098   0.864619   0.424569   0.215740
       Improper:    F
COM Translation:    5.572200   2.986200   5.257400
            RSD:    0.000012
  Sub structure: 2
       Z-matrix: 1
     Quaternion:    0.424570  -0.215734  -0.160095   0.864621
       Improper:    F
COM Translation:    1.878642   6.279200   1.801061
            RSD:    0.001022
  Sub structure: 3
       Z-matrix: 1
     Quaternion:    0.160098   0.864619   0.424571   0.215738
       Improper:    T
COM Translation:   -0.260502   3.599800   8.859539
            RSD:    0.000780
  Sub structure: 4
       Z-matrix: 1
     Quaternion:    0.424571  -0.215738  -0.160098   0.864619
       Improper:    T
COM Translation:    3.433066   0.306799  12.315883
            RSD:    0.000957
 Saved quaternion data to dcdnb_cq1_METXOZ.qxyz
 Printed ok
 Writing dcdnb_cq1_METXOZ.zmat
```

Key points are that the RSD ('root square deviation') is nice and small for all the molecules, and that the `.qxyz` and `.zmat` files are written out as noted. They look like this:

`dcdnb_cq1_METXOZ.qxyz`:

```
1  1  1  1  0.1600  0.8646  0.4245  0.2157  F   5.5722  2.9862  5.257400
2  2  1  1  0.4245 -0.2157 -0.1600  0.8646  F   1.8786  6.2792  1.801061
3  3  1  1  0.1600  0.8646  0.4245  0.2157  T  -0.2605  3.5998  8.859539
4  4  1  1  0.4245 -0.2157 -0.1600  0.8646  T   3.4330  0.3067 12.315883
```

dcdnb_cq1_METXOZ.zmat

```
METXOZ : Generated from the CSD
17
x1        0   0.00000    0     0.00000    0     0.00000
C3        1   1.37954    0     0.00000    0     0.00000
C2        2   1.37353    1    59.53489    0     0.00000
H1        2   0.92929    3   120.35904    1   177.96918
C1        3   1.37615    2   121.14223    4   176.96814
Cl1       5   1.71476    3   123.07847    2  -177.27394
C6        5   1.37557    6   118.04285    3  -177.60553
H2        7   0.93031    5   119.45915    6    -0.36510
C5        7   1.38738    8   119.46602    5   179.97775
Cl2       9   1.71322    7   117.94444    8     2.70892
C4        9   1.38178   10   123.47576    7  -175.70216
N1       11   1.47348    9   122.39807    2  -179.44162
O1       12   1.22960   11   117.61214    9   -35.59310
O2       12   1.20904   13   125.05824   11   178.51016
N2        3   1.46948    5   122.19559    2   178.98005
O3       15   1.20818    3   117.34031    5  -136.99116
O4       15   1.20155   16   124.43945    3   177.02079
```

Where the four integers at the front of the `.qxyz` file keep track of which molecule goes on which molecular site and in which orientation (we only have one orientation per site). To differentiate a molecular site from an atomic site (the more usual use of the word 'site' in crystallography) I will sometimes refer to a 'location'.

Now we need to examine the NO_2 groups. Working backwards, we see that O4 depends on O3 (atom 16), so if we allow the dihedral angle of O3 to vary, O4 will follow—good. O2 depends on O1 (atom 13), and nothing depends on O2 (atom 14) and nothing *but* O2 depends on O1. This means that allowing the N1/O1/O2 group to rotate will not cause any other part of the molecule to 'break'.

So we look like we have a sensible molecular z-matrix for our purposes. It is possible to explore the internal motions of the molecule and any unexpected ramifications of the connectivity by using zmat_anim, which animates the motions.

BUGNOTE:

```
\> zmat_anim --param=3 --delta=0.1 --nstep=20 dcdnb_cq1_METXOZ.zmat 13
17 At line 184 of file zmat_anim.f90 (Unit 6)
Traceback: not available, compile with -ftrace=frame or -ftrace=full
Fortran runtime error: Nonnegative width required in format
(A,I0,A,F)
         ^
```

BUGNOTE: I am not sure windows zmat_maker works when there is more than one z-matrix.

Anyway, we can't run zmat_anim, but z-matrix seems OK. Can't run zmatchk either...

BUGNOTE:

```
\> zmatchk dcdnb_cq1_METXOZ.zmat dcdnb_cq1_METXOZ.qxyz
At line 512 of file string_functions.f90
Traceback: not available, compile with -ftrace=frame or -ftrace=full
Fortran runtime error: Bad real number in item 1 of list input
```

So since we are not worrying about flipping molecules or occupancies of any kind, we are just putting together a purely thermal model, or purely displacive to be more precise, since that makes no assumptions about the static or dynamic nature of the displacements.

So we need to generate some contact vectors. The relevant bits of the output from `ZMC --help` and `ZMC --help2` are respectively (full output from `ZMC` version from April 2011 shown in appendices E and F):

```
 |-----------------------------------------------------------------------|
 | Usage:                                                                |
 |                                                                       |
 | zmc [--option_1] [--option_2] ... [--option_n] infile [outfile]       |
 |                                                                       |
 | infile contains the parameters and additional filenames to run        |
 | the MC simulation.                                                    |
 |                                                                       |
 | outfile is the root name for most output; if not given the root       |
 | name will be infile (i.e., outfile = infile)                          |
 |                                                                       |
 | Options always begin with two dashes, and if the option can be        |
 | passed a value, the value must be indicated with an equals sign       |
 | and there must be no spaces.  E.g.: --summary=inline is right,        |
 | "--summary inline" is wrong, "--summary = inline" is wrong.           |
 |                                                                       |
 | Items enclosed in square brackets are optional.                       |
 |                                                                       |
 | Options are:                                                          |
 |                                                                       |
...
snip
...
 |                                                                       |
 | --help    Prints this information and exits.                          |
 |                                                                       |
 | --help2   Prints more help and exits.                                 |
 |                                                                       |
 | --version Prints version information and exits.                       |
 |                                                                       |
 | --plot    Runs interactive plotting of the simulation; crude but      |
 |           sometimes useful. Exits after producing plots.              |
 |                                                                       |
 | --getcontacts                                                         |
 |           Runs a simple routine to generate contact vectors           |
 |           based on parameters specified in keyword file.              |
 |           Then exits.                                                 |
 |                                                                       |
```

```
| --help, --help2 and --version can be run without infile             |
| or outfile.out being specified. --quiet does not work with --help,  |
| --help2 or --version. If --getcontacts is given, will not do MC     |
| or plot.  If --plot is given, will not proceed to MC.               |
|                                                                     |
|---------------------------------------------------------------------|


|---------------------------------------------------------------------|
|                                                                     |
| The main input file is a series of keywords and values.  Some       |
| are mandatory, some are not.  The order in the keyword file does    |
| not matter.  The details are outlined below.                        |
|                                                                     |
| It is best to begin all filenames called within the keyword file    |
| with alphabetical characters, not with numbers or other characters.  |
|                                                                     |
| ZMC uses a z-matrix to describe a molecule.  If carefully           |
| constructed, this allows segmented motion of the molecules in a     |
| simple way.  The convention for naming these files is to give       |
| them the extension ".zmat".  While a z-matrix defines the           |
| molecular geometry, the molecule must be oriented and positioned    |
| within the unit cell.  This is done using a 3-vector (x, y, z)      |
| to specify the origin of the molecule (the position of the first    |
| atom) and a quaternion (a normalised 4-vector (q1,q2,q3,q4)) to     |
| give the orientation.  This information is in files with            |
| extension ".qxyz". If one uses a third vector to hold the values    |
| of the internal degrees of freedom for the molecule (this will      |
| be an n-vector if there are n internal d.f.) then the molecule      |
| is completely specified by the 3-vector, the 4-vector and the n-    |
| vector.  Given that a substantial molecule may have 50 atoms in     |
| it, requiring 150 coordinates, this is a great economy of           |
| variables.                                                          |
|                                                                     |
| A molecule is said to occupy a location rather than a site          |
| simply because site is a word commonly attached to atomic           |
| position, and there is a desire to avoid confusion by using a       |
| word (hopefully) without connotations.                              |
|                                                                     |
| A single Monte Carlo (MC) step consists of choosing a molecule      |
| at random, calculating its energy by summing over any               |
| interactions pertaining to it (whether internal to the molecule     |
| or between the molecule and its environment), then randomly         |
| altering the molecules configuration (putting random shifts on      |
| the three vectors noted above), calculating the new energy and      |
| then accepting or rejecting the new configuration based on a MC     |
| algorithm.                                                          |
|                                                                     |
| Hence the keywords allow the user to specify what the degrees of    |
| freedom are, what the force constants (spring constants) acting     |
```

```
 | in the system are, and how wide the distribution of random          |
 | shifts can be (referred to as various kinds of widths).             |
 |                                                                     |
 | ZMATFILE     Specifies the file containing a z-matrix.  Must        |
 |              be followed by an integer, which is the z-matrix       |
 |              number, and a filename.                                |
 |              e.g. ZMATFILE 1 paraterphenyl.zmat                     |
 |                                                                     |
 | QXYZFILE     Specifies the file containing the variables            |
 |              describing the average origin of the z-matrix          |
 |              (xyz) and its orientation (quaternion, q).  Must       |
 |              be followed by an integer, which is the z-matrix       |
 |              number, and a filename.                                |
 |              e.g.  QXYZFILE 1 paraterphenyl.qxyz                    |
 |                                                                     |
 | CELL         Specifies unit cell parameters (angles in              |
 |              degrees). Must be followed by 6 reals.                 |
 |              e.g.  CELL 12.34 5.126 8.902 90.0  109.35 90.0         |
 |                                                                     |
 |                                                                     |
 | VMIN         When creating some contact vectors, this is the        |
 |              minimum length (Angstrom). e.g. VMIN 1.9               |
 |                                                                     |
 | VMAX         When creating some contact vectors, this is the        |
 |              maximum length.                                        |
 |                                                                     |
 | NEWCONTACTS  Specifies the new contact vector file name.            |
 |              e.g. NEWCONTACTS newfilename.out                       |
 |                                                                     |
 | CONTATOMS    Specified which atoms to look at when generating       |
 |              contact vectors.  "CONTATOMS ZMAT 1 17 25" would       |
 |              look for contacts between atoms 17 and 25 on           |
 |              ZMAT 1.  To look for contacts involving more than      |
 |              one z-matrix type, use multiple instances of           |
 |              CONTATOMS.                                             |
...
etc
...
 |---------------------------------------------------------------------|
```

So from this we have a first try at assembling an input file that will give us a list of contact vectors (interactions between non-bonded atoms, and in this case ones on different molecules as well – *inter*molecular not *intra*). First, what atoms will be use? Terminal atoms are H, Cl and O. These are atoms number: 6, 10 (Cl), 4, 8 (H) and 13, 14, 16, 17 (O). Let's use these; there are a range of strategies for choosing the interactions. Earlier studies of ours used slimmed down molecules and the fewest possible interactions as 'effective' interactions [24, 19]. More recent studies have used more exhaustive lists of interactions [34], have played around with intramolecular contacts and so on [23] but for an example we'll just generate a list of 'some' interactions; we're not worried about

doing good science, we just want a calculation that works, for now.

We also need to decide on a VMIN and VMAX. Interactive exploration in Mercury suggests that VMIN can be very short to make sure we pick up the short contacts, and that VMAX could be around 4.3Å.

So here is our initial input file, `MC_DCDNB.inp`:

```
HEADER DCDNB from the CSD
!!
!! Basic geometry needed
!!
CELL 9.5900    6.5860   14.7510   90.0000  106.8600   90.0000
ZMATFILE 1 dcdnb_cq1_METXOZ.zmat
QXYZFILE 1 dcdnb_cq1_METXOZ.qxyz
!!
!! Stuff related to getting contacts:
!!
VMIN  0.5
VMAX  4.3
NEWCONTACTS DCDNB_1.contacts
CONTATOMS ZMAT 1 6 10 4 8 13 14 16 17
```

Here is the command that was run:

$$> ZMC −−getcontacts MC_DCDNB.inp ¿ MC_DCDNB_getcontects.screen

Note that the output redirection will be stymied if the contact vector file already exists, since there will be an interactive prompt asking for overwrite confirmation...

And here is the screen output, `MC_DCDNB_getcontects.screen`:

```
 ------------------------------------------------------------------------------
 This is ZMC version dated April 20 2011, a program for implementing
 Monte Carlo simulations of crystal structures, primarily for the
 analysis of diffuse scattering.

 All use should reference the paper:
 D. J. Goossens, A. P. Heerdegen, E. J. Chan & T. R. Welberry, 'Monte
 Carlo Modelling of Diffuse Scattering from Single Crystals: The
 Program ZMC' Metallurgical and Materials Transactions A, vol 42A,
 (2011) 23-31.  DOI:10.1007/s11661-010-0199-1

 Invoke it with ZMC --help for some help, and ZMC --help2 for more help.
 ------------------------------------------------------------------------------
 Input file is MC_DCDNB.inp
 ------------------------------------------------------------------------------
 Keyword filename will be used as root name for output files.
 Output files have root name MC_DCDNB.
 ------------------------------------------------------------------------------
 Cell parameters are:   9.5900  6.5860 14.7510 90.0000106.8600 90.0000
 ------------------------------------------------------------------------------
 Value for NUMLOCS not specified explicitly.
 ------------------------------------------------------------------------------
```

```
 Value for NUMZMATS not specified explicitly.
 ------------------------------------------------------------------------
 Number of z-matrix files to read:     1
 Reading z-matrix   1 of   1 from :              dcdnb_cq1_METXOZ.zmat
 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
METXOZ : Generated from the CSD
17
x1        0   0.00000    0     0.00000    0     0.00000
C3        1   1.37954    0     0.00000    0     0.00000
C2        2   1.37353    1    59.53489    0     0.00000
H1        2   0.92929    3   120.35904    1   177.96918
C1        3   1.37615    2   121.14223    4   176.96814
Cl1       5   1.71476    3   123.07847    2  -177.27394
C6        5   1.37557    6   118.04285    3  -177.60553
H2        7   0.93031    5   119.45915    6    -0.36510
C5        7   1.38738    8   119.46602    5   179.97775
Cl2       9   1.71322    7   117.94444    8     2.70892
C4        9   1.38178   10   123.47576    7  -175.70216
N1       11   1.47348    9   122.39807    2  -179.44162
O1       12   1.22960   11   117.61214    9   -35.59310
O2       12   1.20904   13   125.05824   11   178.51016
N2        3   1.46948    5   122.19559    2   178.98005
O3       15   1.20818    3   117.34031    5  -136.99116
O4       15   1.20155   16   124.43945    3   177.02079
 Printed ok
 ------------------------------------------------------------------------
 Interrogation of qxyz files suggests there are    4 locations in unit cell.
 Interrogation of qxyz files suggests there are
 up to    1 instances of a z-matrix on a location.
 ------------------------------------------------------------------------
 Quaternions and translations for z-matrix    1
 dcdnb_cq1_METXOZ.qxyz

                     Location):    1
  Sub structure (per location):    1
                    Quaternion:    0.160098   0.864619   0.424569   0.215740
                      Improper:    F
               COM Translation:    5.572200   2.986200   5.257400

                     Location):    2
  Sub structure (per location):    1
                    Quaternion:    0.424570  -0.215734  -0.160095   0.864621
                      Improper:    F
               COM Translation:    1.878642   6.279200   1.801061

                     Location):    3
  Sub structure (per location):    1
                    Quaternion:    0.160098   0.864619   0.424571   0.215738
                      Improper:    T
               COM Translation:   -0.260502   3.599800   8.859539
```

```
                    Location):    4
 Sub structure (per location):    1
                  Quaternion:    0.424571  -0.215738  -0.160098   0.864619
                    Improper:    T
               COM Translation:    3.433066   0.306799  12.315883
------------------------------------------------------------------------------
Location    1 can have    1 type(s) of z-matrix(ices).
These are types 1
Type    1 is in    1 orientation(s) on that location.
These are orientation(s):    1
Location    2 can have    1 type(s) of z-matrix(ices).
These are types 1
Type    1 is in    1 orientation(s) on that location.
These are orientation(s):    1
Location    3 can have    1 type(s) of z-matrix(ices).
These are types 1
Type    1 is in    1 orientation(s) on that location.
These are orientation(s):    1
Location    4 can have    1 type(s) of z-matrix(ices).
These are types 1
Type    1 is in    1 orientation(s) on that location.
These are orientation(s):    1
------------------------------------------------------------------------------
Value for VMIN specified explicitly is:  0.5
------------------------------------------------------------------------------
Value for VMAX specified explicitly is:  4.3
------------------------------------------------------------------------------
Coordinates for location:     1
Occupied by zmatrix     :     1
Which is ordinal number :     1 on that location.
In orientation number   :     1 for that location.
   1   5.572  2.986  5.257
   2   6.274  2.857  4.077
   3   6.567  3.873  4.954
   4   6.729  2.788  3.269
   5   5.862  4.027  6.125
   6   6.139  5.316  7.222
   7   4.853  3.138  6.415
   8   4.353  3.249  7.191
   9   4.570  2.076  5.568
  10   3.252  1.058  5.970
  11   5.307  1.946  4.406
  12   5.093  0.842  3.454
  13   4.796 -0.259  3.916
  14   5.247  1.083  2.280
  15   7.648  4.789  4.564
  16   7.683  5.148  3.411
  17   8.463  5.090  5.394
------------------------------------------------
```

```
Coordinates for location:     2
Occupied by zmatrix     :     1
Which is ordinal number :     1 on that location.
In orientation number   :     1 for that location.
   1   1.879  6.279  1.801
   2   1.176  6.150  2.981
   3   0.884  7.166  2.105
   4   0.722  6.081  3.789
   5   1.589  7.320  0.933
   6   1.312  8.609 -0.163
   7   2.598  6.431  0.644
   8   3.098  6.542 -0.133
   9   2.881  5.369  1.491
  10   4.199  4.351  1.089
  11   2.144  5.239  2.653
  12   2.358  4.135  3.604
  13   2.654  3.034  3.142
  14   2.204  4.376  4.779
  15  -0.197  8.082  2.494
  16  -0.232  8.441  3.648
  17  -1.012  8.383  1.664
-------------------------------------------------
Coordinates for location:     3
Occupied by zmatrix     :     1
Which is ordinal number :     1 on that location.
In orientation number   :     1 for that location.
   1  -0.261  3.600  8.860
   2  -0.963  3.729 10.040
   3  -1.255  2.713  9.163
   4  -1.417  3.798 10.847
   5  -0.550  2.559  7.992
   6  -0.827  1.270  6.895
   7   0.458  3.448  7.702
   8   0.958  3.337  6.926
   9   0.742  4.510  8.549
  10   2.059  5.528  8.147
  11   0.005  4.640  9.711
  12   0.219  5.744 10.663
  13   0.515  6.845 10.201
  14   0.065  5.503 11.837
  15  -2.336  1.797  9.553
  16  -2.371  1.438 10.706
  17  -3.151  1.496  8.723
-------------------------------------------------
Coordinates for location:     4
Occupied by zmatrix     :     1
Which is ordinal number :     1 on that location.
In orientation number   :     1 for that location.
   1   3.433  0.307 12.316
   2   4.135  0.436 11.135
```

```
    3   4.428 -0.580 12.012
    4   4.590  0.505 10.328
    5   3.723 -0.734 13.184
    6   4.000 -2.023 14.280
    7   2.714  0.155 13.473
    8   2.214  0.043 14.250
    9   2.431  1.217 12.626
   10   1.113  2.235 13.028
   11   3.168  1.347 11.464
   12   2.953  2.451 10.513
   13   2.657  3.552 10.974
   14   3.108  2.210  9.338
   15   5.509 -1.496 11.622
   16   5.544 -1.855 10.469
   17   6.323 -1.797 12.453
 ------------------------------------------------
 Initial Cartesian positions calculated.
 ------------------------------------------------------------------------------
 Exiting normally.
 ------------------------------------------------------------------------------
```

And here is our initial contact vector list, `DCDNB_1.contacts`:

```
 ol  oz  om oat  da  db  dc  dl  dz  dm dat  length  type
  1   1   1   6   0   0   0   3   1   1  10 4.18846        1
  1   1   1   6   0   1   0   1   1   1  10 3.91387        2
  1   1   1   6   0   1   0   1   1   1  13 3.70847        3
  1   1   1   6   0   1   0   4   1   1   4 3.89845        4
  1   1   1   6   0   1   0   4   1   1  16 3.35289        5
  1   1   1   6   1   0   0   3   1   1  17 4.11577        6
  1   1   1   6   1   1   0   3   1   1   6 3.66621        7
  1   1   1   6   1   1   0   3   1   1  17 3.16091        8
  1   1   1  10   0  -1   0   1   1   1   6 3.91387        9
...
etc
...
  4   1   1  17   1  -1   1   2   1   1   8 3.12604      283
  4   1   1  17   1   0   0   3   1   1  16 3.78436      284
```

Unsorted, with each vector given its own type. Plainly this is silly; at the very least all vectors that are symmetry-equivalent should be of the same type. There my be good chemical reasons for combining more vectors than that, although when using effective interactions, since it is not possible to know in detail what interactions the chosen ones are 'standing in for', it is tricky to make any strong assumptions.

Sorting these on length, and then assigning a type to each symmetry-equivalent set gives this list:

```
ol oz om oat da db dc dl dz dm dat length type
1 1 1 4 1 -1 0 2 1 1 17 2.64139 1
2 1 1 17 -1 1 0 1 1 1 4 2.64139 1
```

```
3 1 1 4 -1 1 0 4 1 1 17 2.6414 1
4 1 1 4 1 0 0 3 1 1 17 2.6414 1
3 1 1 17 -1 0 0 4 1 1 4 2.6414 1
4 1 1 17 1 -1 0 3 1 1 4 2.6414 1
...
etc
...
4 1 1 10 0 0 0 3 1 1 16 4.26201 36
3 1 1 16 0 0 0 4 1 1 10 4.26201 36
4 1 1 16 0 -1 0 3 1 1 10 4.26201 36
2 1 1 10 0 0 0 1 1 1 16 4.26202 36
1 1 1 16 0 0 0 2 1 1 10 4.26202 36
```

And we'll save this to `DCDNB_1_sort.contacts`. Now, the first thing to do with this is to plot the contacts. I use the crude '−−plot' option of ZMC; it is *extremely* crude, but useful. The other thing to do is to think about ways to classify the vectors; this helps with developing the model. Classification schemes include chemistry (H-H, H-O, H-Cl etc) or which neighbour they connect to, or what direction they 'point' in and so on. You may want to make different groups weaker or stronger depending on any of these criteria, in an attempt to fit your data.

Here is the ZMC input file set up for plotting; I have left the contact vector generation stuff in at the bottom.

```
HEADER DCDNB from the CSD
!!
!! Basic geometry needed
!!
CELL 9.5900    6.5860   14.7510   90.0000  106.8600   90.0000
ZMATFILE 1 dcdnb_cq1_METXOZ.zmat
QXYZFILE 1 dcdnb_cq1_METXOZ.qxyz
CONTACTFILE  DCDNB_1_sort.contacts
CRYSTAL 32 32 32
!!
!! Stuff related to getting contacts:
!!
VMIN  0.5
VMAX  4.3
NEWCONTACTS DCDNB_1.contacts
CONTATOMS ZMAT 1 6 10 4 8 13 14 16 17
```

By interactively responding to the prompts, the first lot of files I have generated are simply views of the structure down three principal directions; these are called `no_cont_?.ps`, where ? = x, y or z and inspection of these is a good idea to see if the geometry is being set properly by ZMC (also a good idea to compare contact lengths output from ZMC with those explored in Mercury for additional verification). Figure 4.9.

Next plots are `con_mul_?.ps`, and I can plot any subset of the 36 types, but here I plot them all just to show the complete bird's nest they add up to. Figure 4.10.

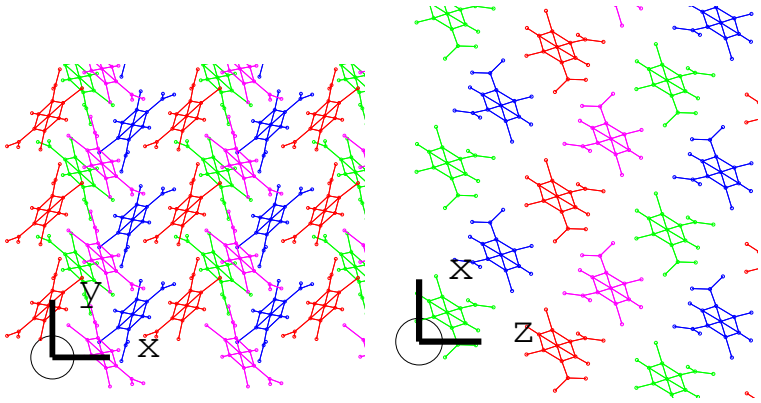


Figure 4.9: Two views of part of the model crystal of DCDNB showing multi-coloured molecules and no contact vectors; output from `ZMC`.

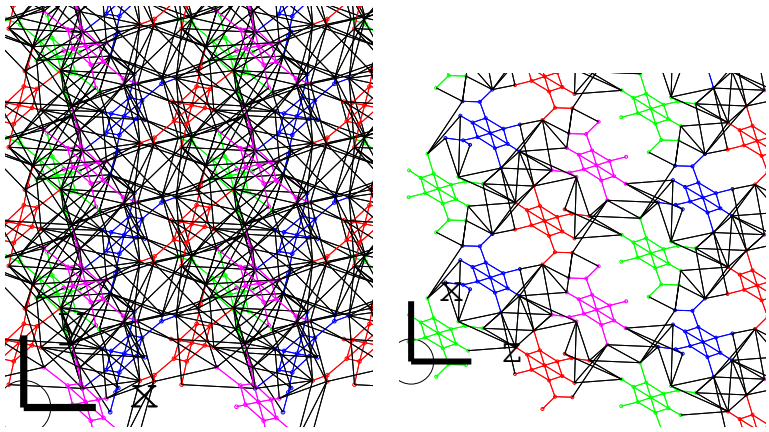


Figure 4.10: Two views of part of the model crystal of DCDNB showing multi-coloured molecules and all contact vectors; output from `ZMC`.

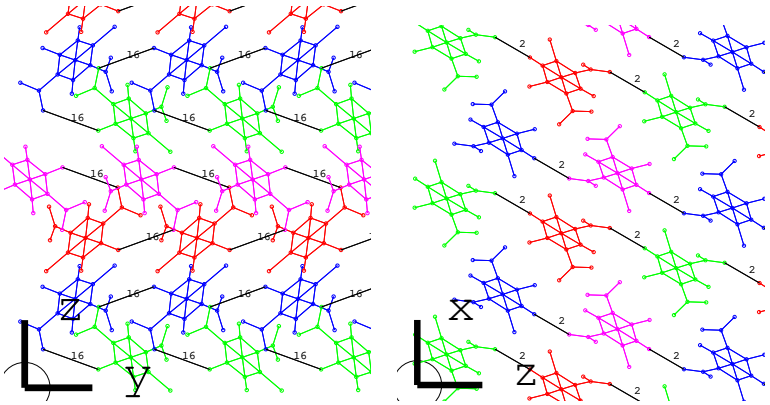


Figure 4.11: Left: View down x showing vector 16, right: View down y showing vector 2.

While this is not terribly useful, it already shows that there are some directions along which the network is thick with vectors and others are relatively more sparse. This reinforces the word 'effective' in the phrase 'effective interactions'(!).

A useful thing to do is plot all the vectors one by one. Get `ZMC` to number them, but say no to 'plot multiple vectors on same plot' or whatever it exactly asks. This will generate 3×36 plots which I suggest combining into three files, one for each of x, y and z. This can be done (at least on Linux) simply by `cat`ting the required files together then running `ps2ps` on the output.

For example

```
cat con_???_y.ps > cony.ps
ps2ps cony.ps y.ps
```

will produce a 36 page PostScript file called `y.ps`, one plot per page.

Note that the colours will be awful! `ZMC` uses a crude algorithm to assign colours purely in an attempt to differentiate molecules based on 'location' and type. It is interesting and useful to note vectors that form chains or sheets of molecules, as often this is a useful way of thinking about the crystal. A simple example here is given by `con_016_x.ps`, where the vector seems to give chains, albeit rather indirectly linked, running along y. Similarly vectors 28 and 29 and presumably others. Vector 2 gives chains running approximately along $z - x$. Figure 4.11.

These inspections are also significant in that if one has sorted solely on length,there may be symmetry inequivalent vectors with coincidentally similar lengths that have been given the same type and should not have.

Anyway, for now I just want to calculate a pattern. So the next step is to run an MC simulation (at last!). Below I show the MC input file, with all force constants (SPRCON) set to unity, with a requested overall average B-factor of 2 and every cycle the random shifts are adjusted to try for 50% rejections. This does not have any effect at this time because I am doing just one MC cycle, because I am simply going to explore the effect of different kinds of randomness.

In this example, I have no randomness on the quaternions, just small random shifts on x, y and z.

The MC file is:

```
HEADER DCDNB from the CSD
!!
!! Basic geometry needed
!!
CELL 9.5900    6.5860   14.7510   90.0000  106.8600   90.0000
ZMATFILE 1 dcdnb_cq1_METXOZ.zmat
QXYZFILE 1 dcdnb_cq1_METXOZ.qxyz
CONTACTFILE  DCDNB_1_sort.contacts
CRYSTAL 32 32 32
!!
!! MC variables
!!
MCCYCLES 1
TEMPERATURE 1.0
XYZINITW 0.3
QINITW 0.0
!!
XYZWIDTH 0.0
QWIDTH 0.0
SPRCON 1.0
BADJUST 1 2.0
INCUPDATE 1
!!
!! Stuff related to getting contacts:
!!
VMIN  0.5
VMAX  4.3
NEWCONTACTS DCDNB_1.contacts
CONTATOMS ZMAT 1 6 10 4 8 13 14 16 17
```

And the command line was:

$$> ZMC --diffuse --cif MC_DCDNB.inp Random_xyz

where I asked for the `.cif` file (`Random_xyz.cif`) just to check how big the U (or B) from the XYZINITW value was. The `.cif` opened successfully in Mercury; I am not sure that the $U_{\rm aniso}$ calculation is correct, I must note. I am pretty happy with $U_{\rm iso}$ and the U_{ii}.

So now we need the input files for the DIFFUSE calculation; these are the same as for the DIFFUSE program outlined (though not in enormous detail) in [3], although I usually call the diffuse associated with ZMC 'DZMC'. And one input file suitable for the $hk0$ cut of reciprocal space for this compound is shown below. This file outlines the bit of reciprocal space being considered, whether to subtracts out Bragg scattering or not, what sorts of atoms there are and their scattering factors.

`diffuse.hk0`:

```
'DCDNB hk0  '                        ! Run description
12645,9676                           ! Random number seeds
```

```
9.590 6.586 14.7510 0.0 -0.290 0.0   ! Cell coords (cos of angles)
32 32 32                             ! Size of crystal simulation (unit cells)
Y                                    ! Periodic Boundary?
-8.7367,-6.0000, 0.0000              ! Origin of calculation volume
 8.7367,-6.0000, 0.0000, 400         ! u (horizontal) axis
-8.7367, 6.0000, 0.0000, 400         ! v
-8.7367,-6.0000, 0.0000, 1           ! w (integer at end gives number of slices)
0.62                                 ! sin(theta)/lambda maximum
8,12,5                               ! Lot size
10                                   ! Number of lots
64                                   ! Number of atom sites per cell
5                                    ! Number of atom types
e                                    ! Subtract average lattice?
'C '                                 ! Atom label
2.31,20.8439,1.02,10.2075            ! Scattering coeff (1)
1.58860,0.5687,0.865,51.6512,0.2156  ! Scattering coeff (2)
0.0,0.0                              !fprime, f-double-prime
'O '
3.0485,13.2771,2.2868,5.7011
1.5463,0.3239,0.867,32.9089,0.2508
0.0,0.0
'N'
12.2126,0.00570,3.13220,9.89330
2.01250,28.9975,1.16630,0.58260,-11.529
0.0,0.0
'Cl'
11.4604,0.01040,7.19640,1.16620
6.25560,18.5194,1.64550,47.7784,-9.5574
0.0,0.0
'H '
0.0000,0.0000,0.0000,00.0000
0.0000,00.0000,0.0000,0.0000,000.000
0.0,0.0
```

So this file has X-ray scattering factors taken from the International Tables for Crystallography [35], but could just as easily have neutron scattering lengths [36]. The 'e' tells DIFFUSE to subtract off the scattering due to Bragg peaks, by calculating the Bragg scattering from 5% of the model crystal. 'E' (upper case) would tell it to calculate the Bragg spots from the whole model crystal and 'Y' says calculate the Bragg scattering from the $B_{\rm iso}$ values; this is the fastest but least precise.

The same calculation can then be done with a 'N' in place of the 'e' to leave the Bragg spots in; this is a good way of making sure you are calculating the bit of reciprocal space you think you asked for.

The command line (assuming interactive use of DZMC) is:

$$> DZMC Random_xyz.diffuse

Then give the name of the input file (`diffuse.hk0`) and then the name you want for the output file (in this case I chose `Random_xyz_hk0`

For automation, the easiest thing is to put the name of the input file (in this case `diffuse.hk0`) and the desired output file into a small text file and use

redirected input and output. Something like this:

$$> DZMC Random_xyz.diffuse ¡ hk0.input ¿ hk0.screen

where the filenames are in `hk0.input` and the stuff that would usually go to the screen goes to `hk0.screen`.

Unix utility tee is also useful if you want output to the screen *and* a file.

Here is `diffuse.0kl`, with an 'N' for 'subtract Bragg peaks?', meaning that all the scattering you can see is either Bragg spot or artifact (finite size of the calculation) with any diffuse crushed down into the background.

I note that bin2gray seems to be flaky on Windows, but usable with careful selection of command line parameters.

```
'DCDNB 0kl  '                          ! Run description
12645,9676                             ! Random number seeds
9.590 6.586 14.7510 0.0 -0.290 0.0     ! Cell coords (cos of angles)
32 32 32                               ! Size of crystal simulation (unit cells)
Y                                      ! Periodic Boundary?
 0.0000,-6.0000,-13.441                ! Origin of calculation volume
 0.0000, 6.0000,-13.441, 400           ! u (horizontal) axis
 0.0000,-6.0000, 13.441, 400           ! v
 0.0000,-6.0000,-13.441, 1             ! w (integer at end gives number of slices)
0.62                                   ! sin(theta)/lambda maximum
8,12,5                                 ! Lot size
10                                     ! Number of lots
64                                     ! Number of atom sites per cell
5                                      ! Number of atom types
N                                      ! Subtract average lattice?
'C '                                   ! Atom label
2.31,20.8439,1.02,10.2075              ! Scattering coeff (1)
1.58860,0.5687,0.865,51.6512,0.2156    ! Scattering coeff (2)
0.0,0.0                                !fprime, f-double-prime
'O '
3.0485,13.2771,2.2868,5.7011
1.5463,0.3239,0.867,32.9089,0.2508
0.0,0.0
'N'
12.2126,0.00570,3.13220,9.89330
2.01250,28.9975,1.16630,0.58260,-11.529
0.0,0.0
'Cl'
11.4604,0.01040,7.19640,1.16620
6.25560,18.5194,1.64550,47.7784,-9.5574
0.0,0.0
'H '
0.0000,0.0000,0.0000,00.0000
0.0000,00.0000,0.0000,0.0000,000.000
0.0,0.0
```

This diffuse input file calculates the $0kl$ cut but does not subtract out Bragg scatter. It was run:

```
DZMC Random_xyz.diffuse
```

```
 DIFFUSE -  Ver. 2.1a  - May 11, 1995

 diffuse.in filename?
diffuse.0kl


   The computation volume is defined by:
     (  0.00, -6.00,-13.44) => (  0.00,  6.00,-13.44)
     (  0.00, -6.00,-13.44) => (  0.00, -6.00, 13.44)
     (  0.00, -6.00,-13.44) => (  0.00, -6.00,-13.44)
   Image size is  400  X  400  X  1
   sin(theta)/lambda maximum =  0.62
   Random number seeds are:  12645 9676
   Crystal size is:  32 32 32
   Periodic boundary? Y
   Number of atom sites per cell :  64
   Number of Atom types:  5
   Lot Size is:  8 12 5
   Number of Lots to Compute:  10
   Subtract Average Lattice? N

 intensity file?
Random_xyz_0kl_N

etc
```

and the output goes into `Random_xyz_0kl_N`. this is then turned into a viewable `.pgm` file by running bin2gray. Symmetrising ups the smoothness of the image, but you should always run bin2gray without any symmetrising first to make sure your image is as symmetrical as it ought to be — helps cache errors.

$$> bin2gray –hmirror –vmirror –rotave=2 –norm=3400000 Random_xyz_0kl_N

and this creates `Random_xyz_0kl_N.pgm`, a 16bit `.pgm` file that can be opened in many packages; I use ImageJ [21]. Putting on the 'red hot' colour palette and pulling up the levels a bit gives figure 4.12.

The `diffuse.hk0` treated the same way gives us figure 4.13; no Bragg spots this time, and since the displacements are uncorrelated, no highly structured diffuse either. Essentially we are seeing scattering from uncorrelated molecular structure factors. Figure 4.13.

The effect of more lots can be seen by comparing figure 4.14a with figure 4.13; figure 4.14a uses 100 lots where figure 4.13 uses just 10. Figure 4.14b shows the $0kl$ cut with the Bragg scattering subtracted and 100 lots calculated, for comparison.

Figure 4.15 shows the same two cuts of reciprocal space after the MC has been run for 100 cycles and 1000 cycles, where a cycle is the number of individual MC steps needed to on average visit each molecule once. Key parameters are that all spring constants (SPRCON) have been set to 1.0, and that the system is automatically adjusting to achieve a global B_{iso} of 2.0, and that we are dynamically adjusting the widths if the intervals from which the random

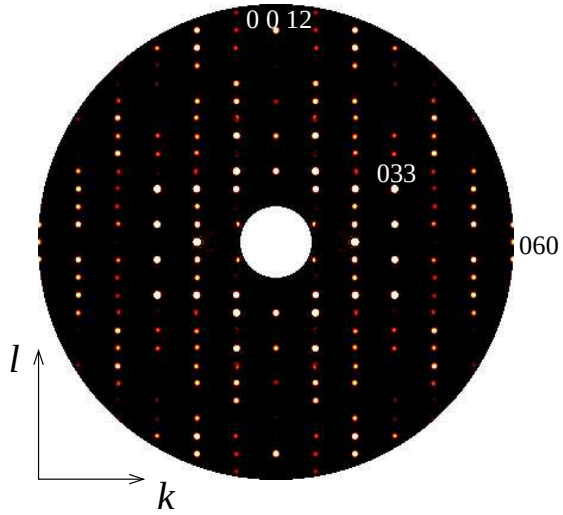


Figure 4.12: The $0kl$ cut of reciprocal space of DCDNB, without the Bragg scattering removed; some peaks noted for scale, and their indices should be compared with the values in the file `diffuse.0kl` shown above.

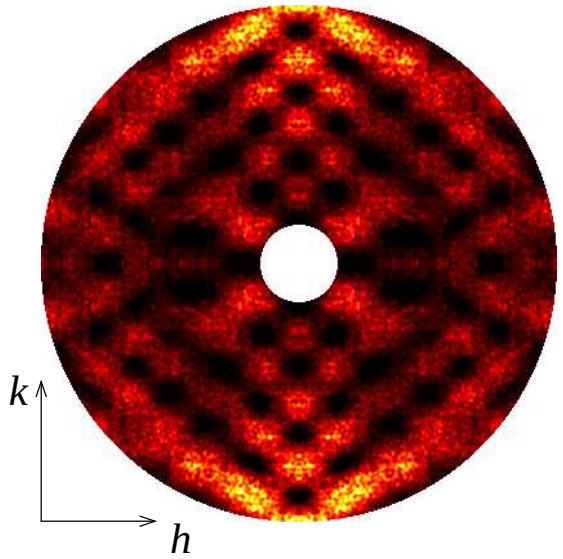


Figure 4.13: The $hk0$ cut of reciprocal space of DCDNB, with the Bragg scattering removed; random displacements on x, y and z coordinates of all molecules, no randomness in quaternions or internal degrees of freedom. Only 10 lots used in the DIFFUSE calculation.

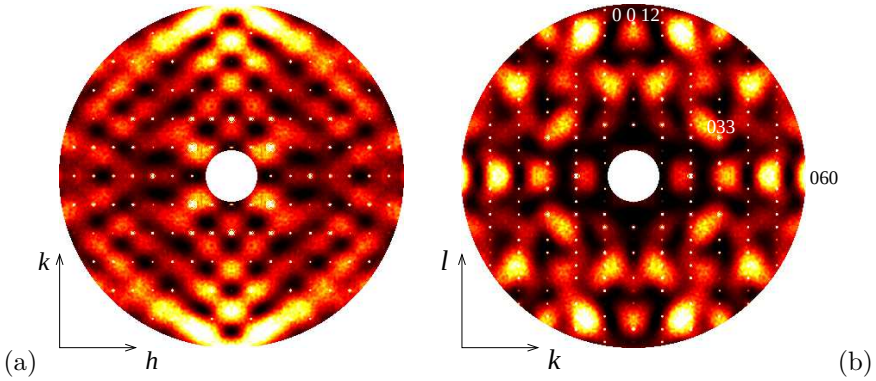


Figure 4.14: (a) The $hk0$ cut of reciprocal space of DCDNB, with the Bragg scattering removed; random displacements on x, y and z coordinates of all molecules, no randomness in quaternions or internal degrees of freedom. 100 lots used in the DIFFUSE calculation. (b) is the same for $0kl$.

increments on the molecular origin (x, y and z, the vector $\mathbf{x}$) and on the four components of the quaternion that governs overall molecular orientation ($\mathbf{q}$). Also, the molecules are being treated as rigid. 100 lots calculated. Bragg peaks were calculated separately and then added on to these images to give a more representative looking dataset.

By capturing the screen output, it is possible to observe the convergence of the B_{iso} to its chosen value and of the acceptance:rejection ratio to a ratio of 50% of moves rejected. Figure 4.16.

Another variation is to classify the contact vectors and modify their force constants. Simple inspection of the contact vector list shows that, for example, for some types of vectors we always have `da`=`dc`=0 but this is not necessarily true of `db`. In other words, these vectors form a chain of contacts running along the b axis. These vectors are numbers: 5,9,12,14,16,18,19,20,22,23,27,28,29,31 and 36. A plot of the structure with just these included is shown in figure 4.17.

So now we can look at what happens for example if we make these much stronger than any other interactions. For example:

```
HEADER DCDNB from the CSD
...
etc
...
SPRCON 1.0
SPRCON 100.0 5,9,12,14,16,18,19,20,22,23,27,28,29,31,36
INSPR 0.0 ZMAT 1 INT 1 2
...
etc
...
```

We would expect this to cause the scattering to be more collapsed along b ($\sim b^*$), giving the scattering a sort of banded appearance. Figure 4.18.

So now we clearly have a working simulation. Some points to note include:

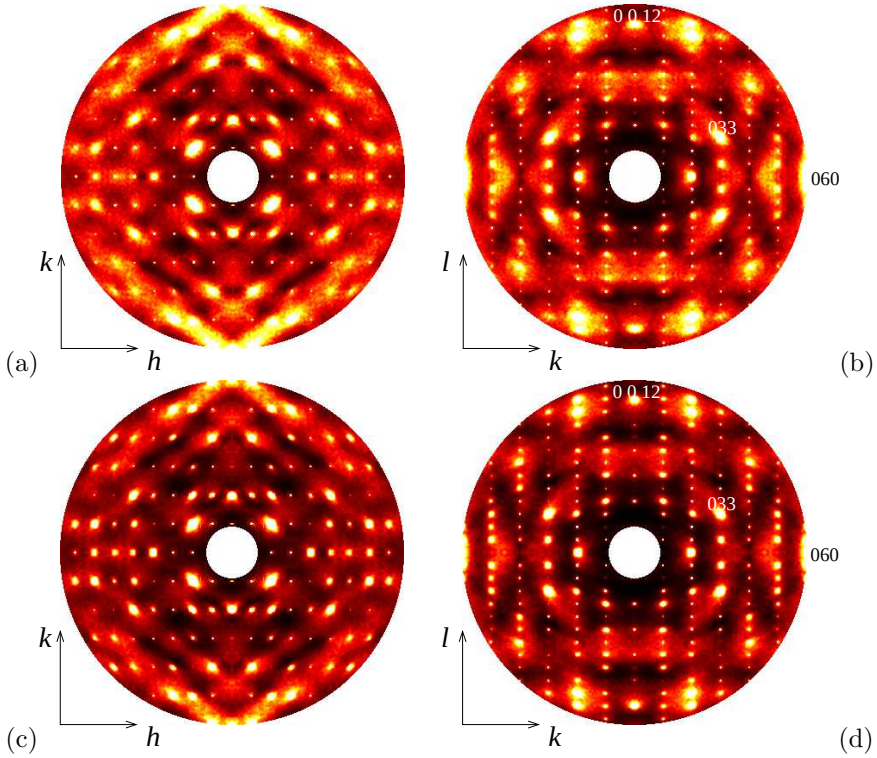


Figure 4.15: (a) and (c): The $hk0$ cut of reciprocal space of DCDNB, with the Bragg scattering removed then added in from a separate calculation. After 100 (a) and 1000 (c) cycles of MC. (b) and (d): Same but $0kl$

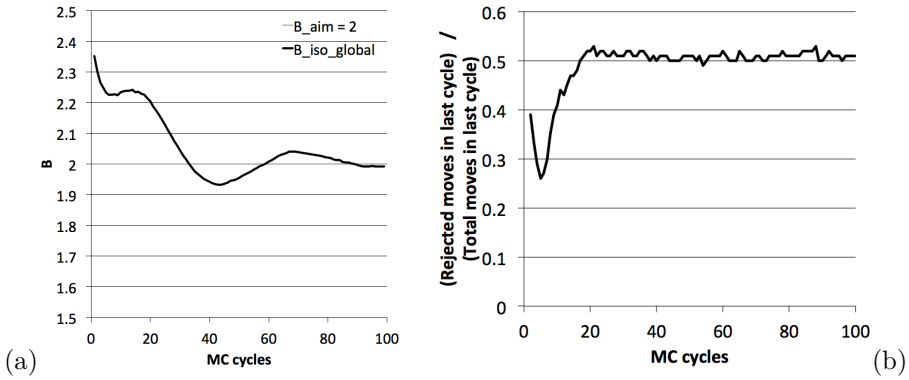


Figure 4.16: (a) The convergence of the global B_{iso} to the requested. (b) The convergence of the rejection ratio (rejected MC moves ÷ total moves) to 0.5.

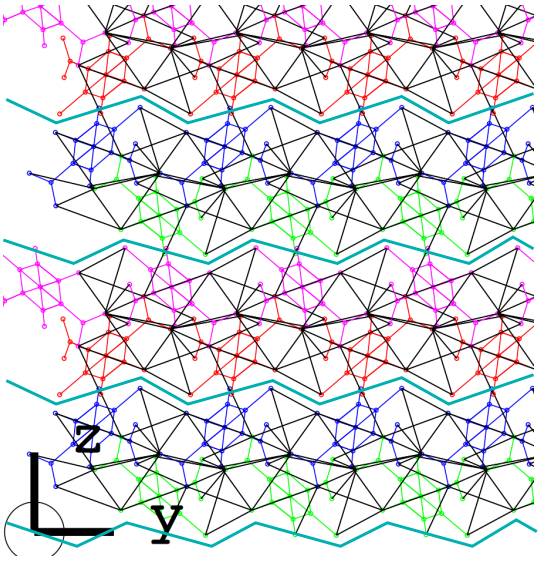


Figure 4.17: Classes of contacts for which `da`=`dc`=0 is always true. These therefore operate in chains along b, outlined by thick lines.

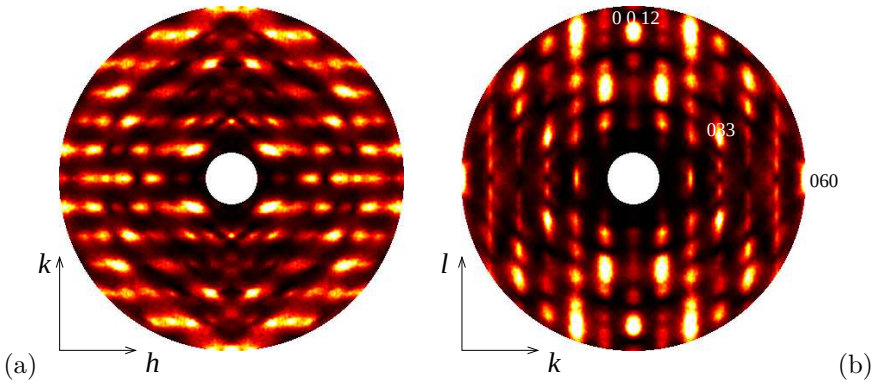


Figure 4.18: (a) The $hk0$ and (b) the $0kl$ scattering froim a model in which the classes of contact vectors for which `da`=`dc`=0 is always true are 100 times stronger than the rest.

- The notable sharpening of features when we go from 100 to 1000 MC cycles — how mush sharper can it get?
- The fact that I have chosen. two easy sections to set up input files for — $h0l$ for example is trickier since β is not 90°.
- Note that the 'random' calculations give a good indication of *where* in the pattern scattered intensity is 'allowed' to go according to the molecular structure factor. What this effectively means is that there are regions of the diffraction pattern which will always be dark simply because the molecular structure factor is small there. Such regions cannot therefore be good tests of a model of the SRO, and this needs to be considered when evaluating and refining models.
- After 100 cycles the global B_{iso} is still oscillating, although the oscillation amplitudes are small and soon after 100 MC cycles it settles down to the desired value.
- Adjusting the rejection ratio and the B_{iso} at the same time results in interaction between the two. Further, if they fail to converge, that usually means a subtle flaw or at least trick in the model; one example occurred when one species had been removed form the occupancy structure to run some simplified tests, but its existence had been left in the input files; the actual MC did not care, it simply did not encounter this species when calculating energies; but when it came to calculating a B_{iso} across this second species, numbers were undefined and this caused a breakdown in the algorithm. Some tests have been added to combat this sort of thing, but who's to say there are not other unusual cases not yet encountered?
- Both the adjusted increments ('widths') and the adjusted spring constants are outputted to the screen at the end of the simulation. These can be copied back into the input file and these algorithms 'turned off' for subsequent simulations, as long as the subsequent simulations are substantially the same as the initial one and can therefore use similar SPRCON and width values.
- All the widths are generated from *uniform* distributions of random numbers, *not* Gaussian; this is because `ZMC` does *not conserve* the overall average inputted distribution — atoms may start with huge ADPs and end with very small ones, or the system may be started in a completely ordered state and thermally disordered. A future project is to make a version of `ZMC` that conserves the average displacements as set up at the start. This would have to be a swapping algorithm, and would use Gaussian random numbers to set up the initial widths.

So far, we have done essentially two simulations; we have put random shifts in $\mathbf{x}$, and we have done a simple MC simulation in which $\mathbf{x}$ and $\mathbf{q}$ are allowed to move. We have done some simple exploration of this as a function of the number of lots calculated and the number of MC cycles done. Other parameters I am not going to explore here include lot size and shape, 'N' versus 'e' versus 'E' versus 'Y' in the Bragg peak subtraction, and the effect on the pattern of randomness in $\mathbf{q}$. These are left as exercises for the reader! (No, I never do those things either.)

What we will look at next because it explores an important aspect of the modelling is internal degrees of freedom. For DCDNB as implemented here, that means rotations of the NO_2 groups. Referring back to the z-matrix we can see that the two atoms that have to be allowed to change their dihedral angles are O1 and O3, or atom numbers 13 and 16. Hence if we want to generate some randomness and not do much MC on these, analogous to what we did above for **x**, then the input file for `ZMC` now looks like this:

```
HEADER DCDNB from the CSD
!!
!! Basic geometry needed
!!
CELL 9.5900    6.5860   14.7510   90.0000  106.8600   90.0000
ZMATFILE 1 dcdnb_cq1_METXOZ.zmat
QXYZFILE 1 dcdnb_cq1_METXOZ.qxyz
CONTACTFILE  DCDNB_1_sort.contacts
CRYSTAL 32 32 32
!!
!! MC variables
!!
MCCYCLES 1
TEMPERATURE 1.0
XYZINITW 0.25
QINITW 0.0
ININITW 35.0
INTERNAL ZMAT 1 INT 1 dihedral 13
INTERNAL ZMAT 1 INT 2 dihedral 16
!!
XYZWIDTH 0.0
QWIDTH 0.0
SPRCON 1.0
INSPR 0.0 ZMAT 1 INT 1 2
BADJUST 1 2.0
INCUPDATE 1
!!
!! Stuff related to getting contacts:
!!
VMIN  0.5
VMAX  4.3
NEWCONTACTS DCDNB_1.contacts
CONTATOMS ZMAT 1 6 10 4 8 13 14 16 17
```

And the resulting `.cif` file gives a unit cell that looks like figure 4.19.

Now, I note that the ADPs result in the distribution on the O atoms looking like ellipsoids, but in truth they will be more like bananas. By plotting the coordinates of these atoms atop each other in a scatter plot, this becomes more apparent, and incidentally shows how misleading conventional ADPs can be in these segmented-molecule modes. Figure 4.20 was created by establishing an MC input file in which the NO_2 group dihedral angles were allowed random shifts of at most around 35° and no other variation was allowed. `ZMC` was then

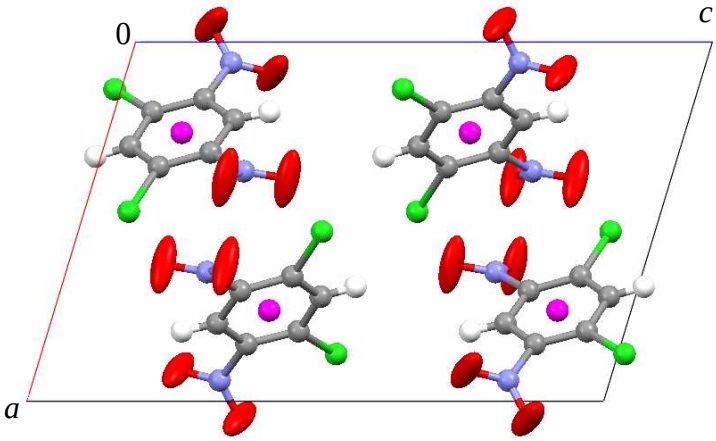


Figure 4.19: A unit cell of DCDNB packed in Mercury, showing the four molecules (Z=4), with some randomness on **x** and on the internal degrees of freedom — rotations of the NO_2 groups, the dihedral angles of which are stored in the vector **i**. Molecules also have some randomness in **x**.

run with the −−cartsn option to output a list of atomic Cartesian coordinates *without* unit cell translations. This is a big file by some standards (> 100MB). I ran this file through Unix sort, sorting by location (sort −g −k 4 infile > outfile) then imported the first few thousand rows into a spreadsheet and plotted y versus x, z versus y and z versus x and then manipulated the plot in xfig [37]. I am sure there are better ways but that suited the computer I was working on at the time; gnuplot [38] and R [39] are better plotting solutions for many users — scriptable, for one thing.

This use of the −−cartsn option brings to notice the other output flags for ZMC. Their syntax can be observed by running ZMC with the −−help flag. This output is largely self-explanatory, but is included in appendix E, with the output of −−help2 in F, for anyone is looking at this document while not in front of a computer.

DCDNB as modelled here possesses two internal degrees of freedom. In the input files shown above, the torsional force constant on these is set to zero by the keyword INSPR in the line:

```
INSPR 0.0 ZMAT 1 INT 1 2
```

So this line says that INTernal degrees of freedom 1 and 2 on ZMATrix 1 have a torsional force ('spring') constant of 0.0. This means that the rotations of these groups are hindered only by the network con contact vectors to other molecules — by the *inter*molecular interactions. The energy penalty associated with rotating the group, in other words, comes from stretching and compressing intermolecular contact vectors away from their equilibrium lengths (equation 2.3). But other restraints could be added; plainly a torsional force constant could be placed on these dihedral angles simply by changing the line above to

```
INSPR 10.0 ZMAT 1 INT 1 2
```

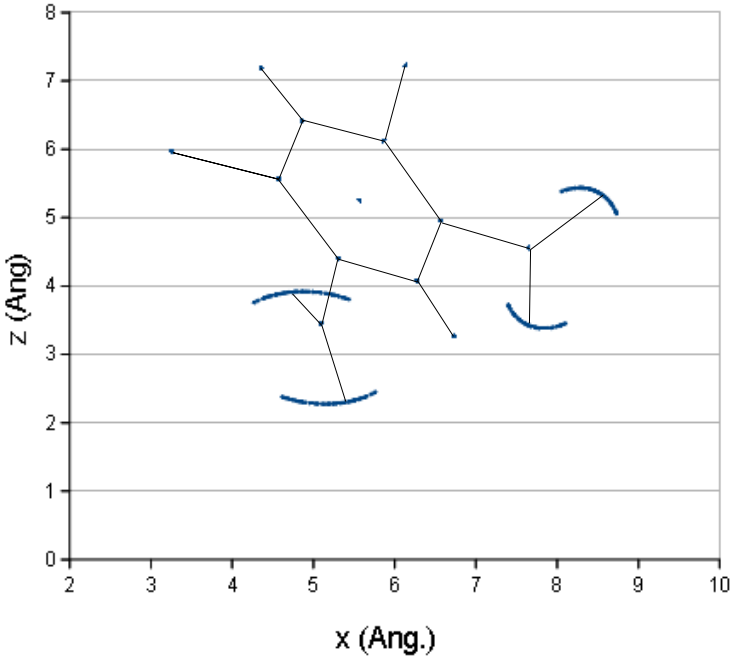


Figure 4.20: A scatter plot of DCDNB with large random variations on the dihedral angles of the NO_2 groups and no other randomness.

for example. The other possibility is *intramolecular* contact vectors. ZMC with the −−getcontacts option ***does not*** generate intramolecular contact vectors. However, since these are trivial to generate manually, or in Mercury, this is not a major issue, and may be built into ZMC in a future version.

In the discussion of *para*-terphenyl in section 4.3, we will look at how combining these things can be used to generate simple molecular conformations and thus, in a sense, a crude form of molecular dynamics.

The other energy that may be needed as interaction *between* internal degrees of freedom. These can be implemented on a pair-wise basis and are referred to as 'cross terms' (equation 2.2). It may be that the groups tend to rotate in parallel (in phase, if you prefer) or not, and this could be implemented using a cross term — although perhaps an intramolecular contact vector would be just as sensible, depending on the details of the system being modelled. ZMC tries to give you the option.

This section has implemented a MC simulation of a relatively simple molecular crystal, chosen (almost) at random. We have seen how to arrange the molecular connectivity, generate a z-matrix and then contact vectors, manipulate the input files to produce various kinds of behaviour and calculate the diffuse scattering. We have not looked at occupancies, although an earlier section has done this to some extent (section 2.2.4). This sort of system is often called a 'thermally' disordered system. In some senses it is not disordered at all; there are no split atomic sites, occupancy simulation needs to be applied. Yet there are many systems of this ilk that show strong, highly structured diffuse

scattering [24, 25] that is plainly informative about the correlated molecular motions in the system.

Indeed, since the correlations in the 'thermal motions' come about through intermolecular interactions, the fitting of the diffuse scattering to a sufficiently realistic model may offer an alternative means of determining intermolecular potentials that may have broader uses, in the calculation of IR spectra, for example; it is to be born in mind that this work is written with a focus on diffuse scattering, but that ultimately we are putting together a model crystal from which, if sufficiently realistic, we may be able to calculate a range of physical properties and compare with experiment — Young's moduli? Lattice contribution to the heat capacity? I am not yet sure...

4.3 *para*-terphenyl

PTP

4.4 Other Stuff

, PCNB, LiNbOF

Chapter 5

Optimising the Model

Least squares, genetic algorithms, calibrated eyeball.

Chapter 6

Parameterising the Force Constants

This discusses Eric's equation for parameterising the force constants, and the idea of using all the contacts out to some length. It can also discuss use of Lennard-Jones, and my idea of Lennard-Jones with a single potential for a given type of atom pair, and not necessarily centred on the observed equilibrium length.

Bibliography

[1] D. J. Goossens, A. P. Heerdegen, E. J. Chan, and T. R. Welberry. *Metallurgical and Materials Transactions A*, 42A:23–31, 2011.

[2] D. J. Goossens and T. R. Welberry. In R. I. Barabash, G. E. Ice, and P. E. A. Turchi, editors, *Diffuse Scattering and the Fundamental Properties of Materials*, pages 181–209. Momentum Press, New York, N. Y., 2009.

[3] B. D. Butler and T. R. Welberry. Calculation of diffuse scattering from simulated disordered crystals: a comparison with optical transforms. *J. Appl. Cryst.*, 25:391–399, 1992.

[4] Th. Proffen and R. Neder. Discus, a program for diffuse scattering and defect structure simulation. *J. Appl. Cryst.*, 30:171–175, 1997.

[5] T. R. Welberry. *Diffuse X-ray Scattering and Models of Disorder*. IUCr Monographs on Crystallography. Oxford University Press, Oxford, 2004.

[6] `http://software.intel.com`.

[7] `http://www.g95.org`.

[8] `http://gcc.gnu.org/fortran/`.

[9] `http://adoxa.110mb.com/tde`.

[10] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[11] T. R. Welberry and B. D. Butler. Interpretation of Diffuse X-ray Scattering *via* Models of Disorder. *Journal of Applied Crystallography*, 27:205–231, 1994.

[12] W. Schweika. *Disordered Alloys: Diffuse Scattering and Monte Carlo Simulations*. Springer, 1997.

[13] L. H. Thomas, T. R. Welberry, D. J. Goossens, A. P. Heerdegen, M. J. Gutmann, S. J. Teat, C. C. Wilson, P. L. Lee, and J. M. Cole. *Acta Crystallographica B*, 63:663–673, 2007.

[14] T. R. Welberry, Th. Proffen, and M. Bown. Analysis of single-crystal diffuse X-ray scattering via automatic refinement of a Monte Carlo model. *Acta Crystallogr.*, A54:661–674, 1998.

[15] Thomas Weber and Hans-Beat Bürgi. Determination and refinement of disordered crystal structures using evolutionary algorithms in combination with Monte Carlo methods. *Acta Crystallographica Section A*, 58(6):526–540, Nov 2002.

[16] `http://www.ccdc.cam.ac.uk/products/csd/`.

[17] D. J. Goossens, T. R.Welberry, A. P. Heerdegen, and A. G. Beasley. *Int. J. Pharmaceutics*, 343:59–68, 2007.

[18] N. E. Brese and M. O'Keeffe. Bond-valence parameters for solids. *Acta Crystallographica*, B47:192–197, 1991.

[19] D. J. Goossens and T. R. Welberry. Monte carlo study of disorder in HMTA. *Computer Physics Commun.*, 142:387–390, 2001.

[20] D. J. Goossens, A. P. Heerdegen, and T. R. Welberry. X-ray diffuse scattering from HMTA: analysis *via* a Monte Carlo model. *Acta Crystallographica Section B*, 64(4):456–465, Aug 2008.

[21] `http://rsbweb.nih.gov/ij/`.

[22] D J Goossens, A G Beasley, T R Welberry, M J Gutmann, and R O Piltz. Neutron diffuse scattering in deuterated para -terphenyl, $C_{18}D_{14}$. *Journal of Physics: Condensed Matter*, 21(12):124204, 2009.

[23] D. J. Goossens and M. J. Gutmann. Revealing how interactions lead to ordering in para-terphenyl. *Phys. Rev. Lett.*, 102(1):015505, Jan 2009.

[24] T. R. Welberry, D. J. Goossens, A. J. Edwards, and W. I. F. David. Diffuse X-ray scattering from benzil, $C_{14}H_{10}O_2$: analysis via automatic refinement of a Monte Carlo model. *Acta Crystallogr.*, A57:101–109, 2001.

[25] T. R. Welberry, D. J. Goossens, W. I. F. David, M. J. Gutmann, M. J. Bull, and A. P. Heerdegen. Diffuse neutron scattering in benzil, $C_{14}D_{10}O_2$, using the time-of-flight laue technique. *J. Appl. Cryst.*, 36:1440–1447, 2003.

[26] D. J. Goossens, T. R. Welberry, and A. P. Heerdegen. *Z. Krist.*, 220:1035–1042, 2005.

[27] D. J. Goossens, T. R. Welberry, A. P. Heerdegen, and M. J. Gutmann. Simultaneous fitting of X-ray and neutron diffuse scattering data. *Acta Crystallographica Section A. Foundations of Crystallography*, A63:30–35, 2007.

[28] D. J. Goossens, A. P. Heerdegen, T. R. Welberry, and M. J. Gutmann. *Physica B*, 385-386:1352–1354, 2006.

[29] P. A. Altin and D. J. Goossens. Diffuse scattering from optically pure ibuprofen. *Proceedings of the 31st Annual Condensed Matter and Materials Meeting*, www.aip.org/publications, 2007.

[30] A. G. Beasley, T. R. Welberry, D. J. Goossens, and A. P. Heerdegen. A room-temperature X-ray diffuse scattering study of form (II) of the trimorphic molecular system *p*-(*N*-methylbenzylidene)-*p*-methylaniline. *Acta Crystallographica Section B*, 64(5):633–643, Oct 2008.

[31] E. J. Chan, T. R. Welberry, D. J. Goossens, A. P. Heerdegen, A. G. Beasley, and P. J. Chupas. Single-crystal diffuse scattering studies on polymorphs of molecular crystals. I. The room-temperature polymorphs of the drug benzocaine. *Acta Crystallographica Section B*, 65(3):382–392, Jun 2009.

[32] E. J. Chan, T. R. Welberry, A. P. Heerdegen, and D. J. Goossens. Diffuse scattering study of aspirin forms (I) and (II). *Acta Crystallographica Section B*, 66(6):696–707, Dec 2010.

[33] D J Goossens, D James, J Dong, R E Whitfield, L Norn, and R L Withers. Local order in layered $NiPS_3$ and $Ni_{0.7}Mg_{0.3}PS_3$. *Journal of Physics: Condensed Matter*, 23(6):065401, 2011.

[34] E. J. Chan, T. R. Welberry, D. J. Goossens, and A. P. Heerdegen. A refinement strategy for Monte Carlo modelling of diffuse scattering from molecular crystal systems. *Journal of Applied Crystallography*, 43(4):913–915, Aug 2010.

[35] `http://it.iucr.org/`.

[36] `http://www.ill.eu/quick-links/publications/`.

[37] `http://www.xfig.org/`.

[38] `http://www.gnuplot.info/`.

[39] `http://www.r-project.org/`.

Index

Appendix A

diffuse.f.

This appendix includes the code for `diffuse.f`, the Fortran 77 version of the diffuse scattering calculation program DIFFUSE [3]. It must be compiled with the `readat.f` shown in appendix B and the requested input file is shown in appendix C. The compile command would look something like:

```
$$> g77 -o Diffuse diffuse.f readat.f
```

```
      PROGRAM DIFFUSE

c       Computes the diffuse diffracted intensity from a simulated
c       disordered crystal.  Input parameters are given in the file
c       'diffuse.in' - output is to the file 'intensity.bin'.
An
c       interface routine that provides access to information in the
c       simulation must be linked to this program.

c       ------------------------------------------
c       Version 2.1a  May 11, 1995    *BDB*
c       Version 2.1 adds option to subtract exact ave. lattice
c       version 2.0b computes Biso with orthoganal axes.
c       ------------------------------------------

        parameter (MI=400,MJ=400,MT=10,MS=155,MAT=6000)
        parameter (MASK=2**14-1)
        complex csf(MI,MJ),acsf(MI,MJ),tcsf(MI,MJ)
        complex cex(0:MASK),cfact(0:1999)
        real dsi(MI,MJ),xat(MAT,3),cell(6)
        real ah_o(3),ah_u(3),ah_v(3),ah_w(3),uin(3),vin(3),win(3)
        real a1(MT),b1(MT),a2(MT),b2(MT),a3(MT),b3(MT)
        real a4(MT),b4(MT),c1(MT),fp(MT),fpp(MT)
        real Biso(MS,MT),wt(MS,MT),ax(MS,MT),ay(MS,MT),az(MS,MT)
        integer istl(MI,MJ),csize(3),ls_xyz(3),lbeg(3),l(3)
        character*70 descrip
        character*4 c_at(MT),cat
        character*1 qalat,qbnd
        character*32 fname
```

```
      common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax
      common/tabels/cex,istl,cfact
      common/scatter/a1,b1,a2,b2,a3,b3,a4,b4,c1,fp,fpp
      common/crystal/csize,cell
      common/info1/ls_xyz,iseed,jseed,nlots,nsites,ntypes
      common/info2/ah_u,ah_v,ah_w
      common/info3/c_at,descrip,qalat,qbnd

c     Find out everything we need to know to run the program ...
       call READINF

c     Write the input info to standard output ...
       call WRITEINF

c     Initialize pseudo-random number generator ...
       call RSEED(iseed,jseed)

c     Initialize the complex exponent table ...
       call cexpt(cex)

c     Open the output file and write the appropriate header ...
c     For 4byte record size (i.e. DEC) use these lines
c     irlen=numu
c     if (irlen.lt.20) irlen=20
c     Else use these lines
       irlen=numu*4
       if (irlen.lt.74)irlen=74
c
  write(6,*) 'intensity file?'
  read(5,1122) fname
  write(6,1122) fname
1122  format(a32)
c        open(unit=1,file='intensity.bin',status='unknown',
        open(unit=1,file=fname,status='unknown',
     *       form='unformatted',access='direct',recl=irlen)
         write(1,rec=1)irlen,descrip
         write(1,rec=2)ah_o,ah_u,ah_v,ah_w,numu,numv,numw
         write(1,rec=3)cell,stlmax
         write(1,rec=4)iseed,jseed,csize,qbnd,nsites,ntypes
         write(1,rec=5)ls_xyz,nlots,qalat
        close(unit=1)

c     Get average atom positions, occupancies, and Debye factors ...
         print *,'  Average Structure ...'
         do 10, isite=1,nsites
          print *,'   Site # ',isite,' :'
          do 10, itype=1,ntypes
           cat=c_at(itype)
           call AVINFO(isite,cat,xx,yy,zz,wx,B)
```

```
              ax(isite,itype)=xx
              ay(isite,itype)=yy
              az(isite,itype)=zz
              wt(isite,itype)=wx
              Biso(isite,itype)=B
              if (nint(wx*100).gt.0) then
               print 101,cat,xx,yy,zz,wt(isite,itype),Biso(isite,itype)
101            format (6x,A,'atom at (',3f6.3,'), xocc = ',
     *                 f5.2,', Biso = ',f6.3)
              end if
10          continue

c        If there is no periodic boundary limit the crystal size ...
         if ( (qbnd.ne.'y').and.(qbnd.ne.'Y') ) then
           csize(1)=csize(1)-ls_xyz(1)
           csize(2)=csize(2)-ls_xyz(2)
           csize(3)=csize(3)-ls_xyz(3)
         end if

c        ------------------------------------------------------------
c        For each reciprocal plane along the w-axis compute scattering
c        ------------------------------------------------------------
         do 100, nw=1,numw
           print *,'  '
           print *,' ----------------------------'
           print *,'  Reciprocal plane # ',nw
           print *,' ----------------------------'

c          Initialize table of sin(theta)/lambda ...
           call STLTAB(istl,cell,nw)

c          Zero some arrays ...
           do 110, j=1,Numv
            do 110, i=1,Numu
             csf(i,j)=cmplx(0.d0,0.d0)
             acsf(i,j)=cmplx(0.d0,0.d0)
             dsi(i,j)=0.d0
110        continue

c         Calculate the average structure factor if asked to do so...
          if( (qalat.eq.'y').or.(qalat.eq.'Y') ) then
            print *,'  Computing Average Scattering using Biso ...'
c           SCATTERING FROM AVERAGE CELL
            do 120, isite=1,nsites
cthp            print *,'   Site # ',isite,' :'
             do 120, itype=1,ntypes
              cat=c_at(itype)
              xat(1,1)=ax(isite,itype)
              xat(1,2)=ay(isite,itype)
              xat(1,3)=az(isite,itype)
```

```
                wx=wt(isite,itype)
                B=Biso(isite,itype)
          natm=1
          if (nint(wx*1000).gt.1) then
                call SCATF(cfact,itype)
                call DEBYE(cfact,B,wx)
                call STRUCF(tcsf,xat,natm,nw)
                do 125, j=1,Numv
                 do 125, i=1,Numu
                  acsf(i,j)=acsf(i,j)+tcsf(i,j)
125             continue
               end if
120           continue

c              INTERFERENCE FUNCTION OF LOT SHAPE
               xx=0.
               yy=0.
               zz=0.
               call GETAV(xat,natm,ls_xyz,xx,yy,zz)
               do 130, i=0,1999
          cfact(i)=cmplx(1.,0.)
130       continue
        call STRUCF(tcsf,xat,natm,nw)
               do 131, j=1,Numv
                do 131, i=1,Numu
                 acsf(i,j)=acsf(i,j)*tcsf(i,j)
131       continue
               print *,' '
             end if

c           A more exact method if asked ...
             if( (qalat.eq.'e').or.(qalat.eq.'E') ) then
        if(qalat.eq.'e') then
          mp=( csize(1)*csize(2)*csize(3) )/20
                print *,' Average Scattering from 5% of Crystal ...'
cthp                print *,' This could take some time ...'
c               SCATTERING FROM AVERAGE CELL
                do 140, isite=1,nsites
cthp                 print *,'  Site # ',isite,' :'
                 do 140, itype=1,ntypes
                  cat=c_at(itype)
                  call SCATF(cfact,itype)
      natm=0
      do 145, ir=1,mp
        call RANLOC(csize,l)
        call READAT(l,isite,cat,inum,x,y,z)
        if(inum.eq.1) then
          natm=natm+1
          xat(natm,1)=x
          xat(natm,2)=y
```

```
          xat(natm,3)=z
          if(natm.eq.MAT) then
                        call STRUCF(tcsf,xat,natm,nw)
                        do 146,  j=1,Numv
                         do 146, i=1,Numu
                           acsf(i,j)=acsf(i,j)+tcsf(i,j)
146                     continue
           natm=0
          end if
        end if
145               continue
                  call STRUCF(tcsf,xat,natm,nw)
                  do 147,  j=1,Numv
                   do 147, i=1,Numu
                     acsf(i,j)=acsf(i,j)+tcsf(i,j)
147               continue
140             continue
         else
          l1=csize(1)
          l2=csize(2)
          l3=csize(3)
          mp=l1*l2*l3
                print *,' Computing EXACT Average Scattering ...'
cthp                print *,' This could take a looooooooong time ...'
c               SCATTERING FROM AVERAGE CELL
                do 150, isite=1,nsites
cthp                 print *,'   Site # ',isite,' :'
                 do 150, itype=1,ntypes
                   cat=c_at(itype)
                   call SCATF(cfact,itype)
       natm=0
       do 155, kk=1,l3
        l(3)=kk
        do 155, jj=1,l2
         l(2)=jj
         do 155, ii=1,l1
          l(1)=ii
          call READAT(l,isite,cat,inum,x,y,z)
          if(inum.eq.1) then
            natm=natm+1
            xat(natm,1)=x
            xat(natm,2)=y
            xat(natm,3)=z
            if(natm.eq.MAT) then
                        call STRUCF(tcsf,xat,natm,nw)
                        do 156,  j=1,Numv
                         do 156, i=1,Numu
                           acsf(i,j)=acsf(i,j)+tcsf(i,j)
156                     continue
             natm=0
```

```
          end if
        end if
155      continue
                  call STRUCF(tcsf,xat,natm,nw)
                  do 157,  j=1,Numv
                   do 157, i=1,Numu
                     acsf(i,j)=acsf(i,j)+tcsf(i,j)
157               continue
150           continue
            end if
c           INTERFERENCE FUNCTION OF LOT SHAPE
            xx=0.
            yy=0.
            zz=0.
            call GETAV(xat,natm,ls_xyz,xx,yy,zz)
            do 160, i=0,1999
         cfact(i)=cmplx(1.,0.)
160      continue
       call STRUCF(tcsf,xat,natm,nw)
       denom=1./mp
            do 161, j=1,Numv
             do 161, i=1,Numu
              acsf(i,j)=acsf(i,j)*tcsf(i,j)*cmplx(denom,0.)
161      continue
            print *,' '
          end if

c         Now compute the diffuse scattering by averaging the total
c         intensities from 'nlots' regions of the simulated crystal.
c         Loop over all atom types and then over all atom sites.
          do 200 nlot=1,nlots
            call RANLOC(csize,lbeg)
c             print *,'    Lot number = ',nlot,' (l,m,n) = (',
c     *              (lbeg(i),i=1,3),')'
            do 210, itype=1,ntypes
               cat=c_at(itype)
               call SCATF(cfact,itype)
               do 210, isite=1,nsites
          if (nint(wt(isite,itype)*1000).gt.1) then
              call GETATM(xat,natm,lbeg,csize,ls_xyz,isite,cat,ncell)
c              print *,'       # of ',c_at(itype),' atoms on lattice'
c     *               ,' site ',isite,' = ',natm
              call STRUCF(tcsf,xat,natm,nw)
c             Add this part of the structure factor to the total ...
              do 215, j=1,Numv
               do 215, i=1,Numu
                csf(i,j)=csf(i,j)+tcsf(i,j)
215           continue
             end if
210         continue
```

```
c             Subtract average (Bragg) scattering amplitude ...
               do 220, j=1,Numv
                do 220, i=1,Numu
                  csf(i,j)=csf(i,j)-acsf(i,j)
220             continue

c             Convert to intensity, add to total, and zero csf() ) ...
               do 230, j=1,Numv
                do 230, i=1,Numu
                  dsi(i,j)=dsi(i,j)+real( csf(i,j)*conjg(csf(i,j)) )
                  csf(i,j)=cmplx(0.d0,0.d0)
230             continue

c           Save the diffuse intensity to disk ...
c           Comment out if you don't want to save this often.
c           There is a pretty big performance penalty on VP!!
c             open(unit=1,file='intensity.bin',status='old',
c     *            form='unformatted',access='direct',recl=irlen)
c              xnorm=1./(real(ncell*nlot))
c              do 240 j=1,Numv
c               irec=numv*(nw-1)+j+5
c               write(1,rec=irec)(dsi(i,j)*xnorm, i=1,Numu)
c240            continue
c             close(unit=1)

c             Tell the file 'cur.txt' where we are ...
  if((nlot/50)*50.eq.nlot.or.nlots.lt.200) then
               open(unit=1,file='cur.txt',status='unknown')
                write(1,*)' Now at lot # ',nlot,'/',nlots
                write(1,*)' Now at plane # ',nw,'/',numw
               close(unit=1)
  endif
200         continue
c           Finished doing 'nlots' regions on this one plane.

c           Check to see if we did zero lots and do average I
     if (nlots.eq.0) then
               write(*,*)' This will only have the average in it!'
               do 300, j=1,Numv
                do 300, i=1,Numu
                  dsi(i,j)=real( acsf(i,j)*conjg(acsf(i,j)) )
300             continue
                nlot=1
        ncell=1
     end if

c           Save the diffuse intensity to disk ...
c             open(unit=1,file='intensity.bin',status='old',
             open(unit=1,file=fname,status='old',
```

```
     *           form='unformatted',access='direct',recl=irlen)
                 xnorm=1./(real(ncell*nlots))
                 do 170 j=1,Numv
                  irec=numv*(nw-1)+j+5
                  write(1,rec=irec)(dsi(i,j)*xnorm, i=1,Numu)
170              continue
                close(unit=1)

100       continue
c         Finished.
          print *,'  All done!'
          print *,' '
c
  errcnt=0
556               open(unit=1,file='cur.count',status='unknown',err=555)
                  read(1,*) icount
          rewind 1
          icount=icount+1
c
                  write(1,*) icount
                 close(unit=1)
    goto 558
555 continue
  errcnt=errcnt+1
  if(errcnt.lt.100) goto 556
  print *, 'unable to write cur.count'
c
558 continue
      END

      SUBROUTINE AVINFO(isite,cat,xx,yy,zz,wx,Biso)
c       Takes a look at all of the unit cells in the simulated
c       crystal and computes the average position (xx,yy,zz) of
c       an atom labeled 'cat' on the lattice site 'isite'.
It
c       also returns the occupation fraction 'wx' and an
c       isotropic static displacement Debye factor 'Biso'

          integer l(3),csize(3)
          real cell(6)
          double precision sumx,sumy,sumz,sumua,sumub,sumuc
          character*4 cat

          common/crystal/csize,cell

c         These are the cell dimensions and cosines
           a1=cell(1)
           a2=cell(2)
           a3=cell(3)
           c1=cell(4)
```

```
          c2=cell(5)
          c3=cell(6)
      s3=sqrt(1.-c3**2)
      ugh=(1.-c2**2+c1**2*s3**2)

c         Define some constants (using double precision for variables
c         that loop real sums) ...
          xpi2=(4.*atan(1.))**2
          ntot=0
          sumx=0.d0
          sumy=0.d0
          sumz=0.d0
          sumua=0.d0
          sumub=0.d0
          sumuc=0.d0

c         Loop over all cells and do the sums ...
           do 10, kk=1,csize(3)
            l(3)=kk
            do 10, jj=1,csize(2)
             l(2)=jj
             do 10, ii=1,csize(1)
              l(1)=ii
c             This is a call to the user supplied subroutine.
We pass
c             it a cell 'l()', site 'isite', and atom description 'cat'
c             and it tells us if there is such an atom there (i.e.
c             inum=1) and if so where (x,y,z).
               call READAT(l,isite,cat,inum,x,y,z)
               if (inum.eq.1) then
                 ntot=ntot+1
                 sumx=sumx+x
                 sumy=sumy+y
                 sumz=sumz+z
                 sumua=sumua+( a1*x +a2*y*c3 + a3*z*c2 )**2
                 sumub=sumub+( a2*y*s3 + a3*z*c1*s3 )**2
                 sumuc=sumuc+(a3*z)**2*ugh
                else if (inum.ne.0) then
               end if
10          continue

c         Normalize, compute Debye factor, and get outa here ...
           wx=real(ntot)/real(csize(1)*csize(2)*csize(3))
      if (wx*1000.gt.1) then
       denom=1./real(ntot)
             xx=sumx*denom
             yy=sumy*denom
             zz=sumz*denom
             sumua=sumua*denom
             sumub=sumub*denom
```

```
            sumuc=sumuc*denom
            sumaa=( a1*xx +a2*yy*c3 + a3*zz*c2 )**2
            sumab=( a2*yy*s3 + a3*zz*c1*s3 )**2
            sumac=(a3*zz)**2*ugh
            Ba=8.*xpi2*(sumua-sumaa)
            Bb=8.*xpi2*(sumub-sumab)
            Bc=8.*xpi2*(sumuc-sumac)
            Biso=(Ba+Bb+Bc)/3.
         end if
         RETURN
         END

         SUBROUTINE GETAV(xat,natm,ls_xyz,xx,yy,zz)
c          Composes a list of atom positions 'xat()' for a site position
c          (xx,yy,zz). There are 'natm' positions returned and these lie
c          inside unit cells with centers inside an elipsoid with major
c          axes (along the crysallographic (a,b,c) directions) of ls_xyz
c          unit cells.

           parameter (MAT=6000)
           real xat(MAT,3)
           integer ls_xyz(3)

           x01=real(ls_xyz(1))/2.
           x02=real(ls_xyz(2))/2.
           x03=real(ls_xyz(3))/2.
           natm=0

c          Loop over all cells that might be in the elipsoid but only
c          keep those cells that have a center inside the elipsoid.
            do 20, kk=0,ls_xyz(3)-1
              xtest3=( real(kk)-x03+0.5 )**2/x03**2
              do 20, jj=0,ls_xyz(2)-1
               xtest2=( real(jj)-x02+0.5 )**2/x02**2
               do 20, ii=0,ls_xyz(1)-1
                xtest1=( real(ii)-x01+0.5 )**2/x01**2
                xtest=xtest1+xtest2+xtest3
                if (xtest.le.1.) then
                 natm=natm+1
                 xat(natm,1)=real(ii)+xx
                 xat(natm,2)=real(jj)+yy
                 xat(natm,3)=real(kk)+zz
                end if
20           continue
         RETURN
         END

         SUBROUTINE GETATM(xat,natm,lbeg,csize,ls_xyz,isite,cat,ncell)
c          Composes a list of 'natm' atom positions 'xat()' for a site
c          position 'isite' and atom label 'cat' that lie inside inside
```

```
c       unit cells with centers inside an elipsoid with major axes
c       (along the crysallographic (a,b,c) directions) of ls_xyz unit
c       cells.  The ellipsoid center is placed using the variable
c       'lbeg()'.  The crystal size 'csize()' is used to avoid over-
c       running the array and the number of unit cells 'ncell'
c       contained in the elipsoid is also returned so that we know
c       what to normalize to later.

        parameter (MAT=6000)
        real xat(MAT,3)
        integer lbeg(3),csize(3),ls_xyz(3),natm
        integer l(3)
        character*4 cat

c       Initialize a few variables ...
         x01=real(ls_xyz(1))/2.
         x02=real(ls_xyz(2))/2.
         x03=real(ls_xyz(3))/2.
         natm=0
         ncell=0

c       Loop over all cells that might be inside the ellipsoid ...
         do 100, kk=0,ls_xyz(3)-1
           l(3)=kk+lbeg(3)
           if(l(3).gt.csize(3)) l(3)=l(3)-csize(3)
           xtest3=( real(kk)-x03+0.5 )**2/x03**2
           do 100, jj=0,ls_xyz(2)-1
            l(2)=jj+lbeg(2)
            if(l(2).gt.csize(2)) l(2)=l(2)-csize(2)
            xtest2=( real(jj)-x02+0.5 )**2/x02**2
            do 100, ii=0,ls_xyz(1)-1
             l(1)=ii+lbeg(1)
             if(l(1).gt.csize(1)) l(1)=l(1)-csize(1)
             xtest1=( real(ii)-x01+0.5 )**2/x01**2

c            Is it inside the ellipsoid?
              xtest=xtest1+xtest2+xtest3
              if (xtest.le.1.) then
                ncell=ncell+1
                call READAT(l,isite,cat,inum,x,y,z)
                if(inum.eq.1) then
                  natm=natm+1
                  xat(natm,1)=real(ii)+x
                  xat(natm,2)=real(jj)+y
                  xat(natm,3)=real(kk)+z
                 else if (inum.ne.0) then
                  stop 'The variable inum from readat must be 0 or 1'
                end if
              end if
100      continue
```

```
      RETURN
      END

      SUBROUTINE CEXPT(cex)
c       This routine computes a table of exp(i*2pi*delta) that is used
c       to save time in computing trig functions in the routine STRUCF.
c       The spacing of the values in the table is 1/2**N where N is
c       chosen to compromise between precision and the size of the
c       table.  Too large of a table size will slow down the computation
c       considerably.

        parameter (i2pi=2**14,MASK=2**14-1)
        complex cex(0:MASK)
        double precision twopi,xmult,xarg,xt

        xt=1.d0/i2pi
        twopi=8.d0*datan(1.d0)
        do 10, i=0,MASK
          xmult=real(i)*xt
          xarg=twopi*xmult
          cex(i)=cmplx( cos(xarg),sin(xarg) )
10      continue

      RETURN
      END

      SUBROUTINE STLTAB(istl,cell,nw)
c       Calculates the value of sin(theta)/lambda for each element
c       of the diffuse scattering array.  The cell parameters (a,b,c,
c       cos(bc),cos(ac),cos(ab)), computation spacings and dimmensions,
c       and the reciprocal section must be provided.  This table is
c       used for quick lookup of the scattering factor curves.
c       The output table is integer with istl=nint(stl*1000).

        parameter (MI=400,MJ=400)
        integer istl(MI,MJ)
        real ah_o(3),cell(6)
        real uin(3),vin(3),win(3)

        common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax

c       The cell parameters ...
        a1=cell(1)
        a2=cell(2)
        a3=cell(3)
        c1=cell(4)
        c2=cell(5)
        c3=cell(6)
        s1=sin(acos(c1))
```

```
        s2=sin(acos(c2))
        s3=sin(acos(c3))

c       I got this out of Culity ...
        V=a1*a2*a3*sqrt(1.-c1**2-c2**2-c3**2+2.*c1*c2*c3)
        S11=(a2**2)*(a3**2)*(s1**2)
        S22=(a1**2)*(a3**2)*(s2**2)
        S33=(a1**2)*(a2**2)*(s3**2)
        S12=(a1)*(a2)*(a3**2)*(c1*c2-c3)
        S23=(a2)*(a3)*(a1**2)*(c2*c3-c1)
        S13=(a1)*(a3)*(a2**2)*(c1*c3-c2)

c       Loop over the all elements in the output image ...
        do 10, jj=1,Numv
          do 10, ii=1,Numu
            xh1=ah_o(1)+real(ii-1)*uin(1)+real(jj-1)*vin(1)+nw*win(1)
            xh2=ah_o(2)+real(ii-1)*uin(2)+real(jj-1)*vin(2)+nw*win(2)
            xh3=ah_o(3)+real(ii-1)*uin(3)+real(jj-1)*vin(3)+nw*win(3)
            stl=sqrt(S11*xh1**2+S22*xh2**2+S33*xh3**2
     *                 +2.*S12*xh1*xh2+2.*S13*xh1*xh3+2.*S23*xh2*xh3)
            stl=0.5*stl/V
            istl(ii,jj)=nint(stl*1000.)
            if (istl(ii,jj).gt.1999) then
              write(*,*)ii,jj,istl(ii,jj)
              stop ' sin(theta)/lambda is greater than 2!'
            end if
10      continue

        RETURN
        END

        SUBROUTINE SCATF(cfact,itype)
c       This routine computes the complex atomic scattering factor as
c       a function of sin(theta)/lambda for the element itype given
c       the parameters in the common block.  If this has already been
c       computed for this atom type then this routine just returns the
c       previously computed values.  (That is what the extra array
c       cftab() holds and what the test is all about.)

        parameter (xinc=0.001)
        parameter (MT=10)
        complex cfact(0:1999)
        real a1(MT),b1(MT),a2(MT),b2(MT),a3(MT),b3(MT)
        real a4(MT),b4(MT),c1(MT),fp(MT),fpp(MT)
        complex cftab(0:1999,MT)
        integer itest(MT)

        common/scatter/a1,b1,a2,b2,a3,b3,a4,b4,c1,fp,fpp
        data itest/10*0/
        save cftab,itest
```

```
c       Test if I have already computed it for this element
        if (itest(itype).eq.1234) then
          do 5, i=0,1999
           cfact(i)=cftab(i,itype)
5         continue
         else
c         I haven't done this element yet so compute it ...
          do 10, i=0,1999
           stl=float(i)*xinc
           stl2=stl**2
           sf =      a1(itype)*exp(-1.*b1(itype)*stl2)
           sf = sf + a2(itype)*exp(-1.*b2(itype)*stl2)
           sf = sf + a3(itype)*exp(-1.*b3(itype)*stl2)
           sf = sf + a4(itype)*exp(-1.*b4(itype)*stl2) + c1(itype)
           sfp=fp(itype)
           sfpp=fpp(itype)
           cfact(i)=cmplx(sf+sfp,sfpp)
           cftab(i,itype)=cfact(i)
10         continue
c          Now I have computed it so let future call know this ...
           itest(itype)=1234
        end if

      RETURN
      END

      SUBROUTINE DEBYE(cfact,Biso,wx)
c       Multiplies a table of scattering lengths 'cfact()' by a
c       static displacement Debye factor and by a weighting factor
c       'wx' which is usually a site occupancy fraction.

        parameter (xinc=0.001)
        complex cfact(0:1999)

c       Loop over the whole tabel cfact() ...
        B=-1.*Biso
        do 10, i=0,1999
         stl=float(i)*xinc
         stl2=stl**2
         cfact(i)=wx*cfact(i)*exp(B*stl2)
10       continue

      RETURN
      END

      SUBROUTINE RANLOC(csize,lbeg)
c       Returns a pseudo-random cell from within the simulated crystal
c       which is 'csize()' cells on edge.
```

```
      integer csize(3),lbeg(3)
      real xran(3)

      call rannum(xran,3)
      lbeg(1)=int(xran(1)*csize(1))+1
      lbeg(2)=int(xran(2)*csize(2))+1
      lbeg(3)=int(xran(3)*csize(3))+1

      RETURN
      END

      SUBROUTINE RSEED(ij,kl)
c       This is the initialization routine for the random number
c       generator rannum() and must be called once prior to rannum().
c       NOTE: The seed variables must have values between:
c       0 <= IJ <= 31328  0 <= KL <= 30081

      real U(97), C, CD, CM
      integer I97, J97

      common /raset1/ U, C, CD, CM, I97, J97
      SAVE

      if( IJ .lt. 0  .or.  IJ .gt. 31328  .or.
     *    KL .lt. 0  .or.  KL .gt. 30081 ) then
        print '(A)', ' The first random number seed must have a ',
     *                'value between 0 and 31328'
        print '(A)',' The second seed must have a value between ',
     *               '0 and 30081'
        stop
      endif

      i = mod(IJ/177, 177) + 2
      j = mod(IJ    , 177) + 2
      k = mod(KL/169, 178) + 1
      l = mod(KL,     169)

      do 2 ii = 1, 97
       s = 0.0
       t = 0.5
       do 3 jj = 1, 24
          m = mod(mod(i*j, 179)*k, 179)
          i = j
          j = k
          k = m
          l = mod(53*l+1, 169)
          if (mod(l*m, 64) .ge. 32) then
             s = s + t
          endif
          t = 0.5 * t
```

```
3         continue
          U(ii) = s
2        continue

         C = 362436.0 / 16777216.0
         CD = 7654321.0 / 16777216.0
         CM = 16777213.0 /16777216.0

         I97 = 97
         J97 = 33

      RETURN
      END

      SUBROUTINE RANNUM(rvec, len)
c      in Florida State University Report: FSU-SCRI-87-50.
It was
c      slightly modified by F. James to produce an array of pseudo-
c      random numbers.

         real rvec(*)
         real U(97), C, CD, CM
         integer I97, J97
         integer ivec

         common /raset1/ U, C, CD, CM, I97, J97
         save

          do 100 ivec = 1, LEN
           uni = U(I97) - U(J97)
           if( uni .lt. 0.0 ) uni = uni + 1.0
           U(I97) = uni
           I97 = I97 - 1
           if(I97 .eq. 0) I97 = 97
           J97 = J97 - 1
           if(J97 .eq. 0) J97 = 97
           C = C - CD
           if( C .lt. 0.0 ) C = C + CM
           uni = uni - C
           if( uni .lt. 0.0 ) uni = uni + 1.0
           RVEC(ivec) = uni
100       continue

      RETURN
      END

      SUBROUTINE READINF
c      This routine reads in all the information necessary to run
c      this program from the file 'diffuse.in' and passes all of this
c      information, through common blocks, back to the main routine
```

```
c       for use as it sees fit.

        parameter (MI=400,MJ=400)
        parameter (MT=10,MS=155)
        real ah_o(3),ah_u(3),ah_v(3),ah_w(3)
        real cell(6),uin(3),vin(3),win(3)
        integer csize(3),ls_xyz(3)
        real a1(MT),b1(MT),a2(MT),b2(MT),a3(MT),b3(MT)
        real a4(MT),b4(MT),c1(MT),fp(MT),fpp(MT)
        character*70 descrip
        character*4 c_at(MT)
        character*1 qalat,qbnd
  character*32 diffin

        common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax
        common/scatter/a1,b1,a2,b2,a3,b3,a4,b4,c1,fp,fpp
        common/crystal/csize,cell
        common/info1/ls_xyz,iseed,jseed,nlots,nsites,ntypes
        common/info2/ah_u,ah_v,ah_w
        common/info3/c_at,descrip,qalat,qbnd

c       Find out everything we need to know to run the program ...
         print *,' '
         print *,'DIFFUSE -  Ver. 2.1a  - May 11, 1995'
         print *,' '
c         open(unit=1,file='diffuse.in',status='old')
  write(6,*)'diffuse.in filename?'
  read(5,*)diffin
        open(unit=1,file=diffin,status='old')
c         Get the run description
          read(1,*)descrip
c         Random Number Seeds [ < 31328 and 30081 respectively ]
          read(1,*)iseed,jseed
c         Get the Lattice Parameter Info (Axial lengths, Cosines)
          read(1,*)(cell(i), i=1,6)
c         Get the Crystal Size in Unit Cells
          read(1,*)(csize(i), i=1,3)
c         Does this crystal contain a periodic boundary?
          read(1,'(A1)')qbnd
c         Get the origin of the image computation
          read(1,*)(ah_o(i),i=1,3)
c         Get the maximum horizontal point and number of divisions
          read(1,*)(ah_u(i),i=1,3),numu
c         Get the maximum vertical point and Number of divisions
          read(1,*)(ah_v(i),i=1,3),numv
c         Get the maximum w-axis point and Number of divisions
          read(1,*)(ah_w(i),i=1,3),numw
          if ( (Numu.gt.MI).or.(Numv.gt.MJ) ) then
            stop '  Your image is too big!'
          end if
```

```
c          Get maximum sin(theta)/lambda
           read(1,*)stlmax
c          Get the Lot Size (Size of Lots << csize() )
           read(1,*)(ls_xyz(i), i=1,3)
c          Get the Number of Lots
           read(1,*)nlots
c          How many Atom Site Positions are there per Cell?
           read(1,*)nsites
           if (nsites.gt.MS) stop ' Too many sites!'
c          How many Atom Types are we to Deal With? (<10)
           read(1,*)ntypes
           if (ntypes.gt.MT) stop ' Too many atom types!'
c          Do we Want to Subtract an Average Lattice (Y/N) ?
           read(1,'(A1)')qalat
c          Read in the Scattering Factor Info ...
           do 10, k=1,ntypes
c            Get these from Table 2.2B, p. 99, Vol. 3, Int. Tables
             read(1,*)c_at(k)
             read(1,*)a1(k),b1(k),a2(k),b2(k)
             read(1,*)a3(k),b3(k),a4(k),b4(k),c1(k)
c            And the Dispersion Corrections
             read(1,*)fp(k),fpp(k)
10         continue
         close(unit=1)

c        Compute the increments along the u,v,w directions
c        This statement prevents a divide by zero on the vp2200
*vocl loop,nopreex
          do 20, i=1,3
           uin(i)=0.d0
           vin(i)=0.d0
           win(i)=0.d0
           if (numu.gt.1) uin(i)= (ah_u(i)-ah_o(i))/real(numu-1)
           if (numv.gt.1) vin(i)= (ah_v(i)-ah_o(i))/real(numv-1)
           if (numw.gt.1) win(i)= (ah_w(i)-ah_o(i))/real(numw-1)
20        continue
      RETURN
      END

      SUBROUTINE WRITEINF
c      Takes the information that was read from the file 'diffuse.in'
c      and writes it back out to standard output so that a record of
c      the run can be kept.

       parameter (MT=10)
       real ah_o(3),ah_u(3),ah_v(3),ah_w(3)
       real cell(6),uin(3),vin(3),win(3)
       integer csize(3),ls_xyz(3)
       real a1(MT),b1(MT),a2(MT),b2(MT),a3(MT),b3(MT)
       real a4(MT),b4(MT),c1(MT),fp(MT),fpp(MT)
```

```
      character*70 descrip
      character*4 c_at(MT)
      character*1 qalat,qbnd

      common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax
      common/scatter/a1,b1,a2,b2,a3,b3,a4,b4,c1,fp,fpp
      common/crystal/csize,cell
      common/info1/ls_xyz,iseed,jseed,nlots,nsites,ntypes
      common/info2/ah_u,ah_v,ah_w
      common/info3/c_at,descrip,qalat,qbnd

      print *,descrip
      print *,' '
      print *,'  The computation volume is defined by:'
      print 101,(ah_o(i), i=1,3),(ah_u(i), i=1,3)
      print 101,(ah_o(i), i=1,3),(ah_v(i), i=1,3)
      print 101,(ah_o(i), i=1,3),(ah_w(i), i=1,3)
      print *,'  Image size is ',numu,' X ',numv,' X ',numw
      print *,'  sin(theta)/lambda maximum = ',stlmax
      print *,'  Random number seeds are: ',iseed,jseed
      print *,'  Crystal size is: ',(csize(i), i=1,3)
      print *,'  Periodic boundary? ',qbnd
      print *,'  Number of atom sites per cell : ',nsites
      print *,'  Number of Atom types: ',ntypes
      print *,'  Lot Size is: ',(ls_xyz(i), i=1,3)
      print *,'  Number of Lots to Compute: ',nlots
      print *,'  Subtract Average Lattice? ',qalat
      print *,' '
101   format(5x,'(',2(f6.2,','),f6.2,') => (',2(f6.2,','),f6.2,')' )

      RETURN
      END

      SUBROUTINE STRUCF(csf,xat,n,nw)
c      Computes the complex stucture factor 'csf()' of n identical
c      atoms at the positions 'xat()' on the reciprocal plane 'nw'.
c      This is the work-horse routine of the program DIFFUSE. Any
c      real speed improvement will come from improving the inner loop
c      of this subroutine.

      parameter (MI=400,MJ=400,MAT=6000)
      parameter (i2pi=2**14,MASK=2**14-1)
      complex csf(MI,MJ)
      complex cex(0:MASK),cfact(0:1999)
      real ah_o(3),uin(3),vin(3),win(3)
      real xat(MAT,3),xm(3)
      double precision xarg0,xincu,xincv
      integer istl(MI,MJ)

      common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax
```

```
      common/tabels/cex,istl,cfact

c     Compute the origin of this reciprocal plane ...
      do 10, i=1,3
       xm(i)=ah_o(i)+(nw-1)*win(i)
10    continue

c     Zero the complex scattering factor array 'csf()' ...
      do 20, j=1,Numv
       do 20, i=1,Numu
        csf(i,j)=cmplx(0.d0,0.d0)
20    continue

      if(n.eq.0) RETURN

c     Loop over all of the atoms we are handling now ...
      do 100 k=1,n
c       Get initial argument to the exponent and increments along
c       the two axies 'u' and 'v'
        xarg0= xm(1)*xat(k,1) + xm(2)*xat(k,2) + xm(3)*xat(k,3)
        xincu= uin(1)*xat(k,1) + uin(2)*xat(k,2) + uin(3)*xat(k,3)
        xincv= vin(1)*xat(k,1) + vin(2)*xat(k,2) + vin(3)*xat(k,3)

c       Convert to high precision integers (64*i2pi=2^20) ...
        iarg0=nint( 64*i2pi*( xarg0-aint(xarg0)+1.d0 ) )
        iincu=nint( 64*i2pi*( xincu-aint(xincu)+1.d0 ) )
        iincv=nint( 64*i2pi*( xincv-aint(xincv)+1.d0 ) )
        iarg=iarg0

c       Loop over all image pixels.  'iadd' is the address of the
c       argument to the complex exponent (in the table 'cex()').
c       The ISHFT operation divides out the 64 and the IAND
c       is equivelent to a MOD and is used so that the argument to
c       the complex exponent is inside our table which has range
c       0=>2pi. 99.5% of the time the CPU will be busy with one of
c       the four statements inside loop 210.
        do 200, j=1,Numv
          do 210, i=1,Numu
           iadd=ISHFT(iarg,-6)
           iadd=IAND(iadd,mask)
           csf(i,j)=csf(i,j)+cex(iadd)
           iarg=iarg+iincu
210       continue
          iarg=iarg0 + j*iincv
200     continue
100   continue
c     We are through with all N atoms ...

c     Multiply the complex scattering factor by the atomic
c     scattering factor for this atom type and return ...
```

```
            do 40, j=1,Numv
               do 40, i=1,Numu
                  csf(i,j)=csf(i,j)*cfact(istl(i,j))
40          continue

      RETURN
      END
```

Appendix B

readat.f.

This appendix includes the code for `readat.f`, a Fortran 77 version of atom reading-in subroutine that is required by the diffuse scattering calculation program DIFFUSE [3] as shown in appendix A. This `readat` is suitable for the output file `output.txt` generated by the example in section 2.2.4.

```
      subroutine readat(LLL,isite,cat,inum,xxx,yyy,zzz)

       PARAMETER (mx = 256)
      real xxx,yyy,zzz
     CHARACTER*2  ty1, ty2, occc(mx,mx,1)
     INTEGER itest, LLL(3),isite,inum
     CHARACTER*4 cat

     save occc, xx,yy,zz,itest

     IF (itest.eq.1234) goto 657
     call getcoords(occc)
657     itest = 1234

     ia=LLL(1)
     ib=LLL(2)
     ic=LLL(3)

     xxx = 0.0
     yyy = 0.0
     zzz = 0.0

     inum = 0
     ty1 = occc(ia,ib,ic)
     ty2 = cat(1:2)
     if(ty1.eq.ty2) inum = 1
     return
     end subroutine readat
```

```
c-----------------------------------------------------------------------

      subroutine getcoords(occc)
         PARAMETER (mx = 256)
       CHARACTER*2    occc(mx,mx,mx)
        integer row(256)

        integer i,j,k,ii,jj,kk,itype,adim,typeflag,irec
c     read in the datafile...
  open(unit=1,file='output.txt')
  do i=1,256
     read(1,'(256i3)')(row(j),j=1,256)
           do j = 1,256
             if(row(j).eq.-1) then
                 occc(i,j,1) = 'Nb'
             else
                 occc(i,j,1) = 'O '
             end if
           end do
  end do
  close(unit=1)

      return

      end subroutine getcoords

c-----------------------------------------------------------------------
```

Appendix C

diffuse.in.

The input file for the program DIFFUSE, in this case for calculating the $hk0$ cut of the reciprocal space of the model crystal simulated by the code in section 2.2.4. In this case the two scatters are taken to be Nb and O, plainly an unphysical model; these were chosen simply because they have strong scattering contrast.

```
Binary Alloy Example                    !Run description
12645,9677                              !Random number seeds
2.1   2.1   2.1    0.0  0.0  0.0       !Cell
256 256 1                               !Simulation Size
Y                                       !Periodic Boundary?
-2.0    -2.0   -0.0                     !Bottom left corner
-2.0     2.0   -0.0    200             !Bottom right, pixels
 2.0    -2.0    0.0    200             !Top left, pixels
-2.0    -2.0   -0.0    1               !Out of plane, slices
2.315                                   !sin(theta)/lambda maximum
8,8,1                                   !Lot size
200                                     !Number of lots
1                                       !Number of atom sites per cell
2                                       !Number of atom types
E                                       !Subtract average lattice?
'Nb'                                    !Atom label
17.61, 1.1886,12.014,11.766             !Scattering function params
4.04183, 0.204785, 3.53346, 69.7957, 3.75591
0.0,0.0                                 !fprime, f-double-prime
'O '
3.0485,13.2771,2.2868,5.7011
1.5463,0.3239,0.8670,32.9089,0.2508
0.0,0.0
```

Appendix D

bin2gray.f.

This appendix includes the code for `bin2gray.f`, a Fortran 77 version of the program that converts the binary output from DIFFUSE into a `.pgm` file for viewing.

```
      PROGRAM BIN2GRAY
c       This program will take the file 'intensity.bin' which was
c       created by the program DIFFUSE and will convert the computed
c       diffuse intensities to a series of 8-bit gray-scale images
c       that can be viewed by a range of software packages.
The output
c       will be to a series of files named img###.pgm where ### is the
c       number of the reciprocal plane it contains.  The output format
c       is Jef Poskanzer's portable graymap (pgm) file format which can
c       be read by many programs and converted to just about any other
c       image format.

        parameter (MI=400,MJ=400)
        parameter (MI2=MI*2)
        real dsi(MI,MJ),ah_o(3),ah_u(3),ah_v(3),ah_w(3)
  real csi(MI,MJ)
        real stl(MI,MJ),cell(6)
        character*1 inc(MI,MJ),chead(MI2)
        real uin(3),vin(3),win(3)
        double precision xmean,xstd
        integer csize(3),ls_xyz(3)
        character*1 qalat,qhorz,qvert,qnorm,qval
        character*70 descrip
        character*10 fname

        common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax

c         Open the input file and find out what record size it is ...
          open(unit=1,file='intensity.bin',status='old',
     *          form='unformatted',access='direct',recl=74)
          read(1,rec=1)irlen
```

```
c
  write(6,*) ' File opened and irlen=',irlen
c
        close (unit=1)

c       Re-open the file (record size now known) and read header ...
          open(unit=1,file='intensity.bin',status='old',
     *         form='unformatted',access='direct',recl=irlen)
            read(1,rec=1)irlen,descrip
            read(1,rec=2)ah_o,ah_u,ah_v,ah_w,numu,numv,numw
            read(1,rec=3)cell,stlmax
            read(1,rec=4)iseed,jseed,csize,qbnd,nsites,ntypes
            read(1,rec=5)ls_xyz,nlots,qalat
          close (unit=1)

c       Now echo to standard output what was in the header ...
          print *,' '
          print *,'-------------------------------------------------------'
          print *,descrip
          print *,'-------------------------------------------------------'
          print *,' '
          print *,'  The computation volume is defined by:'
          print 101,(ah_o(i), i=1,3),(ah_u(i), i=1,3)
          print 101,(ah_o(i), i=1,3),(ah_v(i), i=1,3)
          print 101,(ah_o(i), i=1,3),(ah_w(i), i=1,3)
          print *,'  Image size is ',numu,' X ',numv,' X ',numw
          print *,'  sin(theta)/lambda maximum = ',stlmax
          print *,'  Crystal size is: ',(csize(i), i=1,3)
          print *,'  Lot Size is: ',(ls_xyz(i), i=1,3)
          print *,'  Number of Lots Computed: ',nlots
          print *,'  Subtract Average Lattice? ',qalat
          print *,' '
101       format(5x,'(',2(f5.2,','),f5.2,') => (',2(f5.2,','),f5.2,')' )

c       Get the increments along each of the three axes ...
*vocl loop,nopreex
          do 10, i=1,3
            uin(i)=0.d0
            vin(i)=0.d0
            win(i)=0.d0
            if (numu.gt.1) uin(i)= (ah_u(i)-ah_o(i))/real(numu-1)
            if (numv.gt.1) vin(i)= (ah_v(i)-ah_o(i))/real(numv-1)
            if (numw.gt.1) win(i)= (ah_w(i)-ah_o(i))/real(numw-1)
10        continue

c       Ask some pertinent questions about how to do this ...
          print *,'  Put a bottom half on using a mirror?'
           read(*,'(a)')qhorz
          print *,'  Put a left half on using a mirror?'
           read(*,'(a)')qvert
```

```
          print *,' Use the default ave+3*std normalization?'
           read(*,'(a)')qnorm
          if ( (qnorm.ne.'Y').and.(qnorm.ne.'y') ) then
             print *,' Normalize all planes by the same value?'
              read(*,'(a)')qval
             if ( (qval.eq.'Y').or.(qval.eq.'y') ) then
              print *,' O.K., what value should I normalize to?'
              read(*,*)anorm
             end if
          end if

c         Loop over all reciprocal planes that are in 'intensity.bin'
          do 100, nw=1,numw
            write(*,*)' Working on plane # ',nw,' ...'
c           Extract from file the values associated with plane nw ...
             open(unit=1,file='intensity.bin',status='old',
     *            form='unformatted',access='direct',recl=irlen)
               do 110 j=1,Numv
                 irec=numv*(nw-1)+j+5
                 read(1,rec=irec)(dsi(i,j), i=1,Numu)
110            continue
             close(unit=1)
c
c========================= HERE WE MAKE 2-FOLD ==================
c
  do i=1,numu
  do j=1,numw
  i1=numv-i+1
  j1=numw-j+1
  csi(i,j)=(dsi(i,j)+dsi(i1,j1))/2.
  end do
  end do
  do i=1,numu
  do j=1,numw
  dsi(i,j)=csi(i,j)
  end do
  end do
c
  write(6,*) '======== 2-FOLD IMPOSED ========'
c
c========================= HERE WE END MAKE 2-FOLD ==============
c
c         Find the Max, Min, Ave, and Std Dev of this image ...
          xmax=-1.e32
          xmin=1.e32
          xmean=0.d0
          xstd=0.d0
          do 120, j=1,numv
           do 120, i=1,numu
            if (dsi(i,j).gt.xmax) xmax=dsi(i,j)
```

```
            if (dsi(i,j).lt.xmin) xmin=dsi(i,j)
            xmean=xmean+dsi(i,j)
            xstd=xstd+dsi(i,j)**2
120         continue
            xmean=xmean/(real(numu)*real(numv))
            xstd=xstd/(real(numu)*real(numv)) - xmean**2
            xstd=sqrt(xstd)
            print *,'     Range = ',xmin,' => ',xmax
            print *,'     Mean , STD  = ', xmean,xstd

c           What do we normalize this image to? ...
            IF ( (qnorm.ne.'Y').and.(qnorm.ne.'y') ) THEN
               if ( (qval.eq.'Y').or.(qval.eq.'y') ) then
                 xnorm=anorm
                else
                 print *,'     Normalize to what? (0 gives ave + 4*std) '
                 read(*,*)xnorm
                 if (xnorm.lt.1.e-6) xnorm=xmean+4.*xstd
               end if
             ELSE
               xnorm=xmean+4.*xstd
            END IF
            print *,'     Normalizing to ',xnorm

c           Normalize all values in the image but set to white anything
c           outside of the maximum sin(theta)/lambda ...
            call stltab(stl,cell,nw)
            do 130, j=1,numv
             do 130, i=1,numu
c               dsi(i,j)=dsi(i,j)*255./xnorm   !altered to avoid zero's may-
               dsi(i,j)=(dsi(i,j)*254./xnorm)+1.
               if (dsi(i,j).gt.255.) dsi(i,j)=255.
               if (stl(i,j).gt.stlmax) dsi(i,j)=255.
               inc(i,j)=char( nint(dsi(i,j)) )
130         continue

c           Build a file name for this image ...
            call getname(nw,fname)

c           Build a pgm compatible header for this image ...
            nu=numu
            nv=numv
            if ( (qhorz.eq.'Y').or.(qhorz.eq.'y') ) nv=2*numv-1
            if ( (qvert.eq.'Y').or.(qvert.eq.'y') ) nu=2*numu-1
            call gethead(nu,nv,chead)

c           Open the output file and write the pgm header ...
            open(unit=1,file=fname,status='unknown',access='direct',
     *           form='unformatted',recl=nu)
              irec=1
```

```
            write(1,rec=irec)( chead(k), k=1,nu )

c         Write gray values appending left and bottom if asked ...
          IF ( (qhorz.eq.'Y').or.(qhorz.eq.'y') ) then
             if ( (qvert.eq.'Y').or.(qvert.eq.'y') ) then
               do 170, j=numv,1,-1
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=numu,1,-1),
     *                           (inc(i,j),i=2,numu)
170             continue
                do 175, j=2,numv
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=numu,1,-1),
     *                           (inc(i,j),i=2,numu)
175             continue
               else
                do 180, j=numv,1,-1
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=1,numu)
180             continue
                do 185, j=2,numv
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=1,numu)
185             continue
             end if
           ELSE
             if ( (qvert.eq.'Y').or.(qvert.eq.'y') ) then
               do 190, j=numv,1,-1
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=numu,1,-1),
     *                           (inc(i,j),i=2,numu)
190             continue
               else
                do 195, j=numv,1,-1
                irec=irec+1
                write(1,rec=irec)(inc(i,j),i=1,numu)
195             continue
             end if
          END IF

c         We need to add one more record to the bottom or else some
c         programs that read 'pgm' files will give an error.
I will
c         go ahead and write the run description for lack of anything
c         better to do.
           irec=irec+1
           write(1,rec=irec)descrip
           close(unit=1)

100     continue
```

```
c        We have finished with all of the reciprocal planes now.
      END

      SUBROUTINE STLTAB(stl,cell,nw)
c      Calculates the value of sin(theta)/lambda for each element
c      of the diffuse scattering array.  The cell parameters (a,b,c,
c      cos(bc),cos(ac),cos(ab)), computation spacings and dimmensions,
c      and the reciprocal section must be provided.  This table is
c      used simply to provide a white mask over values that have
c      exceeded the parameter stlmax.

       parameter(MI=400,MJ=400)
       real stl(MI,MJ)
       real ah_o(3),cell(6)
       real uin(3),vin(3),win(3)

       common/sizes/ah_o,uin,vin,win,numu,numv,numw,stlmax

c      The cell parameters ...
        a1=cell(1)
        a2=cell(2)
        a3=cell(3)
        c1=cell(4)
        c2=cell(5)
        c3=cell(6)
        s1=sin(acos(c1))
        s2=sin(acos(c2))
        s3=sin(acos(c3))

c      Some constants that I need (reference Cullity) ...
        V=a1*a2*a3*sqrt(1.-c1**2-c2**2-c3**2+2.*c1*c2*c3)
        S11=(a2**2)*(a3**2)*(s1**2)
        S22=(a1**2)*(a3**2)*(s2**2)
        S33=(a1**2)*(a2**2)*(s3**2)
        S12=(a1)*(a2)*(a3**2)*(c1*c2-c3)
        S23=(a2)*(a3)*(a1**2)*(c2*c3-c1)
        S13=(a1)*(a3)*(a2**2)*(c1*c3-c2)

c      Loop over all pixels in the image ...
        do 10, jj=1,Numv
         do 10, ii=1,Numu
           xh1=ah_o(1)+real(ii-1)*uin(1)+real(jj-1)*vin(1)+nw*win(1)
           xh2=ah_o(2)+real(ii-1)*uin(2)+real(jj-1)*vin(2)+nw*win(2)
           xh3=ah_o(3)+real(ii-1)*uin(3)+real(jj-1)*vin(3)+nw*win(3)
           stl(ii,jj)=sqrt(S11*xh1**2+S22*xh2**2+S33*xh3**2
     *                  +2.*S12*xh1*xh2+2.*S13*xh1*xh3+2.*S23*xh2*xh3)
           stl(ii,jj)=0.5*stl(ii,jj)/V
10       continue

      RETURN
```

```
      END

      SUBROUTINE GETNAME(nw,fname)
c       Builds the character constant 'img###.pgm' where ###=nw
        character*10 fname
         ihun=nw/100
         iten=nw/10 - ihun*10
         ione=nw - 100*ihun - 10*iten
         ihun=ihun+48
         iten=iten+48
         ione=ione+48
         fname='img'//char(ihun)//char(iten)//char(ione)//'.pgm'
      RETURN
      END

      SUBROUTINE GETHEAD(nu,nv,chead)
c       Builds a valid 'pgm' header. This is just a character string
c       with the "magic number" P5 followed by the width then the
c       height of the image.  The value 255 must be right justified in
c       this record for technical reasons.

        parameter (MI=400)
        parameter (MI2=MI*2)
        character*1 chead(MI2)

c       The magic number ...
         chead(1)='P'
         chead(2)='5'
         chead(3)=' '

c       The image width ...
         itho=nu/1000
         ihun=nu/100 - 10*itho
         iten=nu/10 - itho*100 - ihun*10
         ione=nu - itho*1000 - 100*ihun - 10*iten
         chead(4)=char(itho+48)
         chead(5)=char(ihun+48)
         chead(6)=char(iten+48)
         chead(7)=char(ione+48)
         chead(8)=' '
         if (nu.lt.1000) chead(4)=' '
         if (nu.lt.100)  chead(5)=' '
         if (nu.lt.10)   chead(6)=' '

c       The image height ...
         itho=nv/1000
         ihun=nv/100 - 10*itho
         iten=nv/10 - itho*100 - ihun*10
         ione=nv - itho*1000 - 100*ihun - 10*iten
         chead(9)= char(itho+48)
```

```
          chead(10)=char(ihun+48)
          chead(11)=char(iten+48)
          chead(12)=char(ione+48)
          if (nv.lt.1000) chead(9 )=' '
          if (nv.lt.100)  chead(10)=' '
          if (nv.lt.10)   chead(11)=' '

c         A bunch of "white space" ...
          do 10, i=13,nu-3
           chead(i)=' '
10        continue

c         A right justified '255' ...
          chead(nu-2)='2'
          chead(nu-1)='5'
          chead(nu)='5'
        RETURN
        END
```

Appendix E

ZMC −−help.

This does not at present mention the (experimental) −−discus option for outputting a DISCUS-compatible file.

```
|-----------------------------------------------------------------------|
| Usage:                                                                |
|                                                                       |
| zmc [--option_1] [--option_2] ... [--option_n] infile [outfile]       |
|                                                                       |
| infile contains the parameters and additional filenames to run        |
| the MC simulation.                                                    |
|                                                                       |
| outfile is the root name for most output; if not given the root       |
| name will be infile (i.e., outfile = infile)                          |
|                                                                       |
| Options always begin with two dashes, and if the option can be        |
| passed a value, the value must be indicated with an equals sign       |
| and there must be no spaces.  E.g.: --summary=inline is right,        |
| "--summary inline" is wrong, "--summary = inline" is wrong.           |
|                                                                       |
| Items enclosed in square brackets are optional.                       |
|                                                                       |
| Options are:                                                          |
|                                                                       |
| --crystal[=filename]                                                  |
|           Causes program to output a text file referred to as a       |
|           "ZMC crystal file" which contains the variables for each    |
|           molecule.  Can be analysed and/or read back in via          |
|           --reread[=filename] to resume simulation.  If filename      |
|           not given, will write to outfile.crystal.                   |
|                                                                       |
| --diffuse[=filename]                                                  |
|           Causes program to output a file designed to be read into    |
|           diffuse scattering calculation program DIFFUSE.             |
|           Butler and  Welberry J.Appl.Cryst.(1992)25 391-399          |
|           If filename not given, will write to outfile.diffuse.       |
```

```
|           If outfile not given, will write to infile.diffuse       |
|                                                                    |
| --summary[=inline]                                                 |
|           Prints out some information about the model crystal at   |
|           the end of the simulation.  If no argument passed, sends |
|           output to outfile.summary. inline writes output to stdout.|
|                                                                    |
| --pairs   Outputs variables for pairs of molecules connected by    |
|           each contact vector to files pairs_L_C_outfile.out       |
|           where L is the location number of the origin molecule    |
|           C is contact number.  Many of these files will be        |
|           duplicates.                                              |
|                                                                    |
| --fracs   Outputs fractional coordinates of every atom in the model |
|           crystal to outfile.fracs                                 |
|                                                                    |
| --cartsn  Outputs Cartesian coordinates of every atom in the model |
|           to outfile.cartsn. Does not add on unit cell translations.|
|                                                                    |
| --cartst  Outputs Cartesian coordinates of every atom in the model |
|           to outfile.cartst. Adds on unit cell translations.       |
|                                                                    |
| --reread=filename                                                  |
|           Reads in a crystal file and uses it as starting point for !
|           the simulation.  Reads in from filename.                 !
|                                                                    |
| --corro   Calculates correlation ("peanut") diagrams and writes    |
|           them to files r_C_o_XX_outname.out where C is the        |
|           contact vector and XX is the plane (xy, yz, zx).         !
|           Uses the position of the origin atom of each molecule    !
|           in the calculation.  Many of these files will be         |
|           duplicates.                                              |
|                                                                    |
| --corra   As for --corro except file names are r_C_a_XX_outname.out |
|           and it uses the average over all atoms in the molecule.  |
|                                                                    |
| --energy  Outputs the MC energy of each molecule to outfile.energy |
|                                                                    |
| --cif     Outputs a crude CIF file to outfile.cif.                 |
|                                                                    |
| --quiet   ZMC sends nothing to the screen except error messages and |
|           summary information if --summary=inline is set.         |
|                                                                    |
| --help    Prints this information and exits.                       |
|                                                                    |
| --help2   Prints more help and exits.                              |
|                                                                    |
| --version Prints version information and exits.                    |
|                                                                    |
| --plot    Runs interactive plotting of the simulation; crude but   |
```

```
|           sometimes useful. Exits after producing plots.            |
|                                                                     |
| --getcontacts                                                       |
|           Runs a simple routine to generate contact vectors         |
|           based on parameters specified in keyword file.            |
|           Then exits.                                               |
|                                                                     |
| --help, --help2 and --version can be run without infile             |
| or outfile.out being specified. --quiet does not work with --help,  |
| --help2 or --version. If --getcontacts is given, will not do MC     |
| or plot.  If --plot is given, will not proceed to MC.               |
|                                                                     |
|---------------------------------------------------------------------|
```

Appendix F

ZMC −−help2.

```
|-------------------------------------------------------------------|
|                                                                   |
| The main input file is a series of keywords and values.  Some     |
| are mandatory, some are not.  The order in the keyword file does  |
| not matter.  The details are outlined below.                      |
|                                                                   |
| It is best to begin all filenames called within the keyword file  |
| with alphaetical characters, not with numbers or other characters.|
|                                                                   |
| ZMC uses a z-matrix to describe a molecule.  If carefully         |
| constructed, this allows segmented motion of the molecules in a   |
| simple way.  The convention for naming these files is to give     |
| them the extension ".zmat".  While a z-matrix defines the         |
| molecular geometry, the molecule must be oriented and positioned  |
| within the unit cell.  This is done using a 3-vector (x, y, z)    |
| to specify the origin of the molecule (the position of the first  |
| atom) and a quaternion (a normalised 4-vector (q1,q2,q3,q4)) to   |
| give the orientation.  This information is in files with          |
| extension ".qxyz". If one uses a third vector to hold the values  |
| of the internal degrees of freedom for the molecule (this will    |
| be an n-vector if there are n internal d.f.) then the molecule    |
| is completely specified by the 3-vector, the 4-vector and the n-  |
| vector.  Given that a substantial molecule may have 50 atoms in   |
| it, requiring 150 coordinates, this is a great economy of         |
| variables.                                                        |
|                                                                   |
| A molecule is said to occupy a location rather than a site        |
| simply because site is a word commonly attached to atomic         |
| position, and there is a desire to avoid confusion by using a     |
| word (hopefully) without connotations.                            |
|                                                                   |
| A single Monte Carlo (MC) step consists of choosing a molecule    |
| at random, calculating its energy by summing over any             |
| interactions pertaining to it (whether internal to the molecule   |
| or between the molecule and its environment), then randomly       |
```

```
| altering the molecules configuration (putting random shifts on      |
| the three vectors noted above), calculating the new energy and      |
| then accepting or rejecting the new configuration based on a MC     |
| algorithm.                                                          |
|                                                                     |
| Hence the keywords allow the user to specify what the degrees of    |
| freedom are, what the force constants (spring constants) acting     |
| in the system are, and how wide the distribution of random          |
| shifts can be (referred to as various kinds of widths).             |
|                                                                     |
| It is my intention that a fuller manual for ZMC will become         |
| available with time, and that it will be distributed with           |
| example input files and simulations that work.  Also, some          |
| documentation exists in the reviewed literature, as noted in        |
| the messages on running the program.                                |
|                                                                     |
| ------------------------------------------------------------------  |
|                                                                     |
| CRYSTAL      Specifies the size of the model crystal in unit        |
|              cells. Must be followed by three integers.             |
|              e.g. CRYSTAL 32 16 5                                   |
|                                                                     |
| HEADER       Specifies (if given) a header (up to 100               |
|              characters long) for the keyword file and which        |
|              is then written into the output file.                  |
|              e.g. HEADER This is the header.                        |
|                                                                     |
| TEMPERATURE  Specifies the temperature of the Monte Carlo           |
|              simulation. This affects how likely it is that a       |
|              MC move which increases the system energy will be      |
|              accepted.  Must be followed by one real.               |
|              e.g. TEMPERATURE 1.0                                   |
|                                                                     |
| MCCYCLES     Specifies number of Monte Carlo cycles to do. One      |
|              cycle consists of a number of MC steps sufficient      |
|              to visit every location in the crystal once.           |
|              Must be followed by one integer.                       |
|              e.g. MCCYCLES 500                                      |
|                                                                     |
| INCUPDATE    Specifies how many MC cycles between adjusting         |
|              the sizes of the increments (widths) that govern       |
|              the shifts put on the variables in the                 |
|              simulation.  The initial randomness is governed        |
|              by XYZINITW, QINITW and ININITW, while the sizes       |
|              of the random shifts applied to the variables          |
|              during the MC are governed by XYZWIDTH, QWIDTH         |
|              and INWIDTH.  These latter parameters can be           |
|              dynamically adjusted to give a rejection ratio of      |
|              about 50% (half moves rejected) which is a rule        |
|              of thumb often applied to MC simulation.               |
```

	Further, the subroutine adjusts the shifts on the individual variables to ensure that all are having approximately equal influence on the acceptance/rejection ratio. If it turns out that one variable (for example the x position of the origin) has a very big width, this indicates that the simulation is only weakly sensitive to shifts in this direction. This may reflect reality, or may imply more or stronger constraints (interactions) are needed. Must be followed by a single integer. e.g. INCUPDATE 1
BADJUST	Specifies how many MC cycles between adjusting the strengths of the interactions in the system to achieve a specified average B-factor across ALL atoms. Interactions are scaled globally, so this is analogous to changing the simulation temperature. It is helpful because it gives a good indication of the scale of the interactions. Since it scales them all at once their relative values do not change. The B-factor specified (B = 8pi^2U) is an average across all atoms in the z-matrix, so is a very crude number, and is calculated as isotropic, which is also a crude approximation. Must be followed by one integer and one real, where the real is the desired B-factor. e.g. BADJUST 1 2.5
NUMSPRCON	Specifies the number of different spring constants (interaction constants) acting on contact vectors in the model. Will be deduced from other inputs if not given. Must be followed by one integer. e.g. NUMSPRCON 23
NUMINSPRCON	Specifies the number of different spring constants (interaction constants) acting on internal degrees of freedom in the model. Will be deduced from other inputs if not given. Must be followed by one integer. e.g. NUMINSPRCON 23
NUMLOCS	Specifies the number of locations in the unit cell. Location is used instead of site because site tends to be used to refer to atoms; a location is a position in the unit cell which is occupied by a molecule. The occupation of a given location may vary from cell to cell -- the type or orientation of z-matrix on the location may vary. Will be deduced from other inputs if not given. Must be followed by one integer.

```
|              e.g. NUMLOCS 23                                          |
|                                                                       |
| NUMZMATS     Specifies the number of different z-matrices             |
|              present in the simulation.  Will be deduced from         |
|              other inputs if not given.  Must be followed by          |
|              one integer.  e.g. NUMZMATS 2                            |
|                                                                       |
| NUMINTERNAL  Specifies the number of internal degrees of              |
|              freedom a given z-matrix is given.  Must be              |
|              followed by the sub-keyword ZMAT then two                |
|              integers, the ZMAT number (as given in the               |
|              ZMATRIX lines of the keyword file) and how many          |
|              internal degrees of freedom that z-matrix has            |
|              been given.  Will be deduced from other inputs if        |
|              not given.  To specify that z-matrix 2 has 3            |
|              internal degrees of freedom, write:                      |
|              NUMINTERNAL ZMAT 2 3                                     |
|                                                                       |
| NUMCROSS     Specifies the number of cross-terms a given              |
|              z-matrix is given. A cross-term is an interaction        |
|              between internal degrees of freedom.  Must be           |
|              followed by the sub-keyword ZMAT then two                |
|              integers, the ZMAT number (as given in the               |
|              ZMATFILE lines of the keyword file) and how many         |
|              cross terms that z-matrix has been given.  Will          |
|              be deduced from other inputs if not given.  To           |
|              specify that z-matrix 2 has 3 internal cross             |
|              terms, write: NUMCROSS ZMAT 2 3                          |
|                                                                       |
| OCCFILE      Specifies the name of the file the occupancies           |
|              are to be read from.  If not given it is assumed         |
|              that it is not needed (a model in which there is         |
|              no occupancy disorder, generally).                       |
|              e.g. OCCFILE occupancies.txt                             |
|                                                                       |
| ZMATFILE     Specifies the file containing a z-matrix.  Must          |
|              be followed by an integer, which is the z-matrix         |
|              number, and a filename.                                  |
|              e.g. ZMATFILE 1 paraterphenyl.zmat                       |
|                                                                       |
| QXYZFILE     Specifies the file containing the variables              |
|              describing the average origin of the z-matrix            |
|              (xyz) and its orientation (quaternion, q).  Must         |
|              be followed by an integer, which is the z-matrix         |
|              number, and a filename.                                  |
|              e.g.  QXYZFILE 1 paraterphenyl.qxyz                      |
|                                                                       |
| CELL         Specifies unit cell parameters (angles in                |
|              degrees). Must be followed by 6 reals.                   |
|              e.g.  CELL 12.34 5.126 8.902 90.0  109.35 90.0           |
```

CONTACTFILE	Specifies the file containing the contact vectors. e.g. CONTACTFILE Contacts.txt
SPRCON	Specifies value of spring constants (interaction constants). If given with a single real (e.g. SPRCON 12.0) sets a default value that applies to all springs. Subsequent invocations, with a real followed by integers, (e.g. SPRCON 50.2 1 2 3 4 5 6 7) sets the spring constants for those contact vectors specified. The two invocations here would set all springs to be 12.0, then set the spring constants on vectors 1 to 7 to be 50.2, leaving the others, if any, at 12.0.
SIZE	Specifies value of size-effects on contact vectors. Invocation rules are the same as for SPRCON.
INSPR	Specifies spring constants on internal degrees of freedom. An example would be: INSPR 50.0 ZMAT 1 INT 1 ZMAT 2 INT 2 3 This would set the spring constant (interaction constant) for internal degree of freedom 1 on z-matrix 1 to be 50.0 and internal spring constants 2 and 3 of z-matrix 2 would also be 50.0. These must all be set explicitly; there is no defaulting invocation as for SPRCON.
INTERNAL	Specifies the nature of an internal degree of freedom. To specify internal degree of freedom number 1 for z-matrix number 2 to be the dihedral angle of atom 17, type: INTERNAL ZMAT 2 INT 1 dihedral 17 The types of internal d.f. are bond lengths (use "length" instead of "dihedral"), bond angle (sub-keyword "angle") or dihedral angle as shown.
CROSSSPR	Specifies spring constants on cross terms. An example would be: CROSSSPR 50.0 ZMAT 1 CROSS 1 ZMAT 2 CROSS 2 3 Implemented much like INSPR
CROSSDEF	Specifies cross-terms. For example: CROSSDEF ZMAT 1 CROSS 1 INT 1 2 would define the first cross-term of z-matrix 1 is an interaction between internal d.f. 1 and 2.
XYZWIDTH	Sets the sizes of the random increments on x, y

```
|              and z coordinates of z-matrices.  Can be invoked      |
|              in several ways.  "XYZWIDTH 0.2" would set widths     |
|              to be 0.2 for all x,y and z on all z-matrices.        |
|              "XYZWIDTH X 0.1" would set the widths on the          |
|              x-shifts on all z-matrices to be 0.1.                 |
|              "XYZWIDTH 1 0.15" would set widths on x, y and z      |
|              for all of x, y and z on z-matrix 1 to be 0.15        |
|              and (lastly) "XYZWIDTH 1 Y 0.4" would set the         |
|              width of the y coordinate on z-matrix 1  to          |
|              be 0.4.                                              |
|                                                                   |
| XYZINITW     Sets the initial randomness on the variables;        |
|              invocation as for XYZWIDTH.                          |
|                                                                   |
| QWIDTH       Sets the sizes of the random increments on           |
|              quaternion components.  As for XYZWIDTH but refer    |
|              to Q1, Q2, Q3 and Q4 rather than X, Y and Z.         |
|                                                                   |
| QINITW       See QWIDTH and XYZINITW                              |
|                                                                   |
| INWIDTH      Sets the sizes of the random increments on           |
|              internal degrees of freedom.  As for XYZWIDTH but    |
|              refer to I1, I2, ..., IN rather than X, Y and Z.     |
|                                                                   |
| ININITW      See previous couple of entries.                      |
|                                                                   |
| VMIN         When creating some contact vectors, this is the      |
|              minimum length (Angstrom). e.g. VMIN 1.9             |
|                                                                   |
| VMAX         When creating some contact vectors, this is the      |
|              maximum length.                                      |
|                                                                   |
| NEWCONTACTS  Specifies the new contact vector file name.          |
|              e.g. NEWCONTACTS newfilename.out                     |
|                                                                   |
| CONTATOMS    Specified which atoms to look at when generating     |
|              contact vectors.  "CONTATOMS ZMAT 1 17 25" would     |
|              look for contacts between atoms 17 and 25 on         |
|              ZMAT 1.  To look for contacts involving more than    |
|              one z-matrix type, use multiple instances of         |
|              CONTATOMS.                                           |
|                                                                   |
|-------------------------------------------------------------------|
```