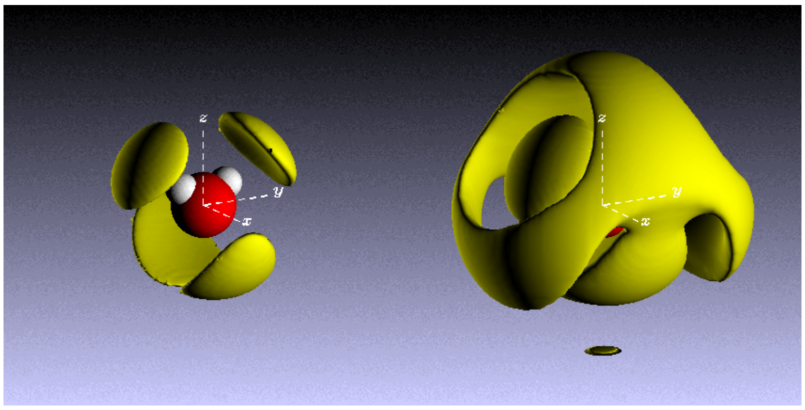
Empirical Potential Structure Refinement



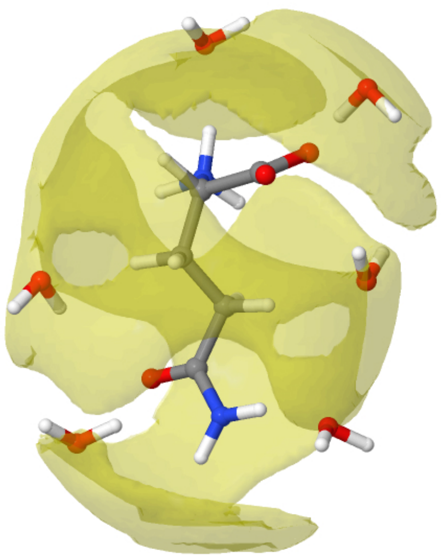
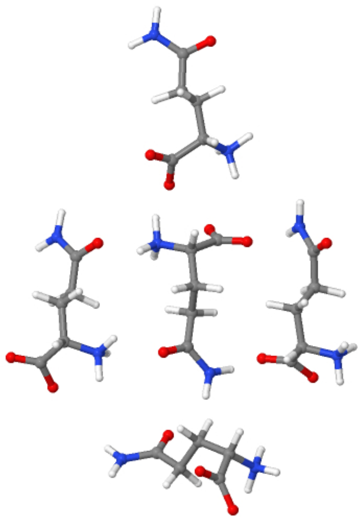
A User's Guide

Version 25: February 2017

by

Alan K Soper

with contributions from Daniel Bowron, Helen Thompson, Fabio Bruni, Maria Antonietta Ricci, Sylvia McLain, Stefan Klotz, Thierry Straessle, Silvia Imberti, Rowan Hargreaves, Tristan Youngs, Sam Callear, Christoph Salzmann, plus many others. Without the continued support and advice from these colleagues much of what follows would never have happened.

Contents

1. Overview: modelling the structure of a liquid or glass **1**

2. Empirical Potential Structure Refinement **6**

2.1 Fundamentals

2.2 Defining the reference interatomic potential.

2.3 Defining the empirical potential.

2.4 The uniform atom distribution.

2.5 Running the simulation.

2.6 Refining the empirical potential – introducing the data.

2.6.1 Defining the amplitude of the Empirical Potential.

2.7 Additional terms to the reference potential.

2.7.1 Atom overlap forces

2.7.2 Container forces

2.7.3 Additional terms

2.7.4 Three-body forces

3. EPSRshell **19**

3.1 Introduction – EPSRshell menus.

3.2 File naming conventions.

3.3 The main menu.

3.4 Script operation.

3.5 The setup menus.

3.6 The plot menu.

3.7 Plotting the box of atoms – **plotato**.

4. Preparing for an EPSR simulation **35**

4.1 Building the **.ato** file – single atom molecules – **makeato.**

4.2 Creating a molecule and building **.mol** and **.ato** files using **Jmol – runjmol**, **readjmol**, and **makemole**.

4.3 Making the simulation box – **mixato** and **addato**.

4.4 Randomising and modifying the **.ato** file – **randomise** and **fmole**.

4.5 Building long chain molecules – **dockato**.

4.6 Building periodic structures – **makelattice** and **makelatticeato**.

4.7 Editing the .ato file – **changeato**.

5. Operating the EPSR simulation program **51**

5.1 Setting up the neutron **.wts** files – **nwts**.

5.2 Setting up the x-ray **.wts** files – **xwts**.

5.3 Setting up the **.inp** and **.pcof** files.

5.4 Running the simulation.

5.5 Automatic control of **ereq.**

5.6 Output files and data format

5.7 Reviewing the output.

6. Auxiliary routines **68**

6.1 Input and output data formats and running.

6.2 **partials** – calculate site-site radial distribution functions.

6.3 **coord** – calculate coordination number distributions about specific sites.

6.4 **triangles** – calculate bond angle distribution functions.

6.5 **torangles** – calculate torsional angle distribution functions.

6.6 **clusters** – calculate cluster size distribution functions.

6.7 **chains** – calculate chain length distribution functions.

6.8 **rings** – calculate ring length distribution functions.

6.9 **voids** – calculate void distributions and the void radial distribution function.

6.10 **fluctuations** – calculate distributions of number fluctuations.

6.11 **writexyz** – writes a file of atomic coordinates in xyz format.

6.12 Building your own analysis routine.

7. Spherical harmonic expansion of many-body correlation functions **86**

7.1 Introduction – the spatial density function and orientational correlation function.

7.2 **sharm** – calculates the spherical harmonic coefficients for the spatial density function and orientational pair correlation function.

7.3 **sdf** – spatial density functions for non-molecular systems.

7.4 **plot2d** and **plot3djmol** – plotting the results from 7.2 and 7.3.

8. Some examples and exercises **97**

8.1 Single component Lennard-Jonesium.

8.2 Two component charged Lennard-Jonesium – NaCl.

8.3 Amorphous silica

8.4 Water.

8.5 Other examples

9. Appendices. **103**

9.1 Files you need to run EPSR

1. Overview: modelling the structure of a liquid or glass.

Modelling is at the core of all human activity. Sometimes a model is called a “theory”, which makes it sound more important and fundamental, but in fact all theories are really nothing more than sophisticated (or not so sophisticated) models. When it comes to interpreting experimental data whether or not you setup a model to understand it is not really an option. Even the simplest interpretation of a set of data involves a degree of modelling of some form or the other, even if it is only in the mind of the person performing the analysis.

In fact modelling is something we do subconsciously all the time. The image on the back of our retinas is upside down and must be inverted. Our brain continuously interprets the light, sound, smell, taste, and feel of the world around us to produce a three-dimensional snapshot of our surroundings. It uses that perceptual model to determine what action, if any, we need to take. The snapshot is updated on a timescale of order 0.1 seconds so if things happen much slower than this we will not see them moving or else perceive them to be moving very slowly – like a snail for example – whereas if they happen much faster the motion is blurred or may not be perceptible at all.

Our perceptual model of the world around us can be very accurate in the elements which it is testing, but inaccurate about elements it cannot see or hear or feel or smell. When the roof collapsed at Charles de Gaulle airport in 2004 the people inside the building were taken by surprise – they had no inkling the building would collapse because the internal state of the roof supports was beyond the scope of their perceptual model. Yet the elements of structural failure which led to the collapse must have been present long before the collapse actually happened. You are not aware of the bullet coming towards you because it is travelling too fast to be incorporated into your perceptual model, yet the dandelion seed which floats by in the breeze is travelling sufficiently slowly for your perceptual model to be continually updated so you are fully aware of its current position relative to yourself and the other objects around you. Therefore be quite clear that modelling is not some sort of “added value” that is used to generate pretty pictures. On the contrary, it is something that goes on all the time we are alive and is essential to our very existence.

In a similar vein quantum mechanics can be regarded as a mathematical model that is used to describe the behaviour of atomic particles. It turns out to be a very accurate model of course, but it is still only a model. It is very unlikely that an electron is actually solving Schrödinger’s equation as it moves around an atom!

When we try to visualise the structure of a liquid or glass we are faced with a conundrum: there is little in our macroscopic environment which is really quite like a liquid at the atomic scale. Our brain is desperate for a “picture” which it can recognise, but whereas a crystal structure has atoms placed in specific positions within a well-defined unit cell (quite analogous to the furniture placed in a room for example), the liquid evades such a simple visualisation. The room in a liquid is infinitely large, there is nowhere to hang your coat, and everything is continually on the move! Indeed it is hard to think of any analogous situation in the world around us which is quite like the structure of a liquid or disordered solid at the atomic level.

Perhaps one example of a macroscopic (2-dimensional) liquid is a crowded shopping centre or street – people milling about, going in all directions, at different speeds, occasionally stopping to talk to acquaintances.

What do we mean by “structure” in such circumstances? Notice that in general people do not actually collide with one another in a crowd. Even on a crowded subway train it is rare for people to be in physical contact. Our senses prevent us approaching another person very closely, unless of course the other person allows us to do so! It is as if an invisible force – in this case the instinct not to collide with other people (strangers especially) – which prevents most collisions. Instead if people attempt to approach one another too closely, this invisible force comes into play, and one or other or both deflect from their original collision path. Thus even though you are free to go anywhere in the street, you cannot go where another person is standing or walking!

Liquids and glasses involve analogous but unimaginably small, atom-scale, movements on unbelievably small timescales, perhaps 10^{-9} – 10^{-15} seconds. There is no structure as such in the crystallographic sense because the atoms are continually moving from one place to another. (The glass is distinct from the liquid in that the atoms have no overall movement – they are essentially a liquid in which the atoms have stopped diffusing, but can still perform some local oscillatory movements. The principle in a glass is the same however – as we move around the glass we will find no structures which repeat themselves indefinitely into the distance. The macroscopic analogue would be the crowded shopping mall which has become so packed that everyone has stopped dead in their tracks, but can still shuffle, wave their arms, and talk to each other. This would be called a “jam”.)

There is however a residual local structure in these situations that arises from the fact that no two atoms can occupy the same space, as with people in the crowded shopping mall. Each atom is surrounded by its own space, creating local arrangements of atoms which are continually changing as the atoms and molecules diffuse around. For molecules this local arrangement is often dependent on the relative orientation of the two molecules as well.

However the ‘structure’ here does not refer to the kind of structure found in a crystal where all the atoms or molecules are laid out in a rather regular and specific manner. Instead it is described by a statistical quantity (technically the ‘pair correlation function’) that is determined from the probability per unit volume that a particular position around a central atom or molecule is occupied by another atom or molecule. Hence in a simple atomic system like liquid argon or liquid helium, the pair correlation function will consist of alternating shells of high and low density as a function of distance from any particular atom, arising from the short range forces between atoms.

In a liquid made from molecules, the pattern is more complex, since it will depend on the distance, the direction we look from our central molecule, and the orientation of the neighbouring molecules. Even in systems where there are no molecules, such as network glasses and molten salts, the local structure can be strongly directional, since it will depend on whether we look towards an atom of the same kind as the one we are on, or whether we look towards an atom of a different kind.

The diffraction experiment, whether it is x-ray, electron or neutron, gives us an experimental glimpse at this local structure.

The information supplied is however far from being in a form which we can immediately visualise. Typically the information consists of a Fourier transform of a weighted average of the density of different atom types about other atom types as a function of distance and direction. Only in the very simplest cases, such as the inert gases, can this information be interpreted directly in terms of the pair correlation function.

Even in the most simple cases our direct visualisation still runs into difficulties. We typically might measure a coordination number for one particular atom type around another, based on our measurements. A coordination number of 5 might be reported for example, but does this mean all atoms are surrounded by exactly 5 atoms, or does it mean that *on average* each atom is surrounded by 5 atoms? Some could be surrounded by 3 or 4 atoms, while others could be surrounded by 6 or 7, while still others might be surrounded by even more perhaps. We like to think the former because it gives us a simpler visual picture, but the truth is more likely to be the latter. Unfortunately we have no way of telling from the diffraction experiment on its own which view is correct, unless we have additional information to incorporate into our structural model.

Empirical Potential Structure Refinement (EPSR) was developed as a tool to do precisely that, namely to incorporate additional or prior information into a structural model of the system being investigated. The process of taking even simple ideas on atomic forces – consider the hard sphere fluid for example – and calculating what they would be like when imposed on a three dimensional arrangement of atoms is a problem of extreme complexity which can only be solved approximately at best – see J.-P. Hansen and I. R. McDonald, *Theory of Simple Liquids*, (Academic Press). Computer simulation is also an approximate method, but it does largely overcome many of the weaknesses of the theory in being able to account to significant extent for the many-body interactions which take place in a condensed fluid. Hence computer simulation is by far the best way of generating models of liquids, especially as most of the systems which are studied are very far from "simple", and the corresponding theory is weak or non-existent in these cases.

EPSR is one method out of a few other methods that attempt to find distributions of atoms which are consistent with experiment. With EPSR you state your prejudice at the beginning via the reference potential, the assumed molecular shapes, the effective charges on atoms, the assumed minimum approach distances, and so on, and then let the computer sift through all the possible arrangements of atoms and molecules which are simultaneously consistent with both your starting prejudice and your scattering data, to the extent that this is possible. If it doesn't work it can only mean one of two things: either your starting assumptions are wrong, or the data are wrong, since we know there must be *some* arrangement of atoms which make up a particular material. The point is that if you can find physical arrangements of atoms which are consistent with your prior knowledge at the beginning of the experiment and with your measurements, there really isn't any more that can be done to extract structural information from an experiment on a glass or liquid, except perhaps try to find out if there are other distinct arrangements of atoms which meet all those requirements.

EPSR is a Monte Carlo method which evolved from Reverse Monte Carlo (RMC), which also attempts to build a structural model of a glass or liquid, and which in turn evolved from earlier attempts to use Monte Carlo methods to evaluate disordered materials structure on the basis of diffraction experiments. As a general rule, since they are based on the Boltzmann-like distributions of statistical mechanics, Monte Carlo methods tend to produce the most disordered solutions to a problem which are compatible with a set of specified constraints. They should therefore also produce the most probable solution.

There is no guarantee however they will produce the *correct* solution, but as is argued in statistical mechanics textbooks, for systems consisting of a large numbers of atoms the likelihood of any particular solution being vastly different from the most probable one is small. In this sense the differences between RMC and EPSR are technical rather than fundamental: ultimately I suspect it could be shown that the two methods are formally equivalent.

For the purposes of modelling molecular materials, these technical differences are however important and may explain why RMC is more appropriate in some instances and EPSR is more appropriate in others. As a general rule, RMC uses hard sphere constraints on atom distances to prevent atomic overlap, and square well constraints to define molecules. It is then hoped that the scattering data will overcome any deficiencies in these assumptions, and so give realistic near-neighbour distributions. It does this by moving atoms at random and then accepting or rejecting each move on the basis of the fit to the diffraction data, using the standard Metropolis Monte Carlo sampling procedure.

EPSR does things slightly differently: molecules in EPSR are defined by means of harmonic force constants between as many pairs of atoms as are required to define the molecule, as well as bond angle forces and dihedral angle forces. In addition *inter*molecular distances, and any *intra*molecular distances which do not have harmonic forces defined, are controlled by a Lennard-Jones potential, to represent the short range repulsive and longer range attractive (dispersion) forces, together with a pseudo-Coulomb potential where appropriate. The combination of intra-molecular forces and inter-molecular seed potentials is called the “reference potential” (RP) in EPSR. The diffraction data are introduced via yet another potential energy function, the “empirical potential” (EP), empirical because it is derived from the data. The contributions to this empirical potential are obtained from the difference, at each stage of the simulation, between the measured and simulated diffraction data. EPSR can therefore be thought of as a method for perturbing the reference potential model used in most other forms of computer simulation of liquids. (See the books by Allen and Tildesley, and Frenkel and Smit, references [3] and [4] in Chapter 2.)

The bonding potentials on molecules and between atoms are more restrictive in EPSR than in RMC. EPSR makes specific assumptions about particular bond lengths and widths in molecules, and the repulsive potential at short distances is not infinitely hard as in RMC. If ions or effective charges are present then these are incorporated via a pseudo-Coulomb potential (“pseudo” because in the current version of EPSR it is truncated rather than treated correctly at large distances). In the exercise on amorphous silica you have the opportunity to experiment with this simple interatomic potential: if you get the parameters correct you will discover that a simple potential consisting of Lennard-Jones parameters plus charges can do a remarkably good job at generating the local structure of silica – even the intermediate range order appears to be captured to some extent. This could not be achieved by standard RMC, because the charge ordering that occurs in this model of silica is crucial to generating its local order.

One could always imagine dreaming up more sophisticated starting potential functions for EPSR to get the simulation to approach the data even closer at the outset. But the combination of Lennard-Jones plus charges is sufficiently simple that it captures neatly the main elements of the forces we expect to see between atoms in a functional form with only 3 or 4 adjustable parameters for each atom. Anything more complicated and you very quickly lose control of the number of adjustable parameters. (In fact with the current version of EPSR, EPSR25, there is now some scope to change the starting potential so as to avoid the use of Lennard-Jones if required.) Of course you are bound to be able to find cases where it doesn’t work, in the sense that it cannot find a solution, but practical experience has shown that these cases are very few and far between. If a fit to a set of diffraction data cannot be found, it almost invariably means there is either something wrong with the supplied data or there is something wrong with the reference potential with which EPSR starts.

Of course whether the solution that is found is the correct solution is a different question which is much harder to answer, and we do not even begin to tackle that question in this manual.

Many of the routines described in this manual have been developed over the past 21 years by a process of trial and error. EPSRshell was originally written as a response to our first EPSR workshop in 2002, when it became apparent that a broad range of people would be likely to use EPSR, ranging from the computer literate to those with only minimal working knowledge of programming and operating systems. It is weakly analogous to Bourne Again Shell (BASH) under UNIX, hence the name EPSRshell: basically it runs its own very limited operating system, from which all the main EPSR commands and operations can be performed. It is entirely Fortran coded, and makes use of some “standard” features of the gfortran compiler which may be slightly different in form in other compilers. It is however a command line interface and therefore may not be to everyone's taste!

In 2016, Sam Callear began work on a Graphical User Interface version of the shell, EPSRgui, which is written to help the user easily create and run EPSR simulations. EPSRgui is now available. Even more than EPSRshell it is designed to ensure that sensible default parameters are incorporated whenever practical, so that the number of decisions the user needs to make is kept to a reasonable limit, and to be compilable under a variety of operating systems. EPSRgui has to be downloaded separately and has its own user manual. It is strongly recommended for anyone new to the EPSR technique. At the time of writing most but not all EPSR features are implemented in EPSRgui, so EPSRshell will continue to be available for the foreseeable future for those who prefer a command line interface.

The main EPSR routines can be compiled using the gfortran Fortran compiler. Hence it runs perfectly satisfactorily on LINUX, as well as Windows (most flavours) and OS/X for example.

Other graphical plotting is done via calls to GNUplot, and for **.ato** files, the files that contain the atomic coordinates in EPSR, can be plotted with Jmol. Small or medium molecular structures can be refined with a call to MOPAC7. It is assumed these are available under the operating system in which EPSRshell is run (Jmol of course requires Java to be installed). There is of course no warranty that any of the routines work correctly, so they have to be used at your own risk! The main elements of the method are described in Sections 2 – 7, with some examples of things you could try to gain experience in Section 8. Section 9 gives some useful file information.

2. Introduction to EPSR.

Empirical Potential Structure Refinement (EPSR) [1] evolved early in 1996 when attempting to do Reverse Monte Carlo (RMC) [2] simulations on systems which contained molecules. It is not the purpose of this manual to go into the basics of computer simulation, since some excellent books on this exist. See for example Allen and Tildesley [3] or Frenkel and Smit[4]. Most of the EPSR computer simulation code was derived from what is written in Allen and Tildesley's book. Here we will simply outline those parts of EPSR which are different from what is said in these "official" guides. Equally we will not go into any of the details of the neutron and x-ray scattering equations since these are available in many places, such as the "Gudrun" manual.

2.1 Fundamentals

In essence EPSR is a standard Monte Carlo simulation of the system being studied. There are typically up to four types of atom move within EPSR, these being whole molecule translations, whole molecule rotations, rotation of specific molecular sidechains where appropriate, and individual atomic moves within molecules. Obviously if a "molecule" has only one atom only the first of these moves is needed. A move consists of a random (small) change in the (*x*,*y*,*z*) coordinates of the atom or molecule, or else a rotation of the molecule by a random amount about a randomly chosen axis. For internal sidechain rotations the axis of rotation is defined in the input files, but the amount of rotation is still random. Other types of atom move could be contemplated (such as molecular inversion through a plane of symmetry) but have so far not been implemented.

The acceptance of a move is based on the usual Metropolis condition, namely if the change in the potential energy of the system as a result of the move, °$U = U_{after}$-U_{before} is less than zero the move is always accepted, and if it is greater than zero, the move is accepted with probability $\exp\left[-\frac{\Delta U}{\mathrm{kT}}\right]$. This simple procedure ensures the system proceeds along a Markov chain and over a period of time visits a large volume of the available phase space.

The potential energy in EPSR consists of two primary terms, the reference potential energy, U_{Ref}, and the empirical potential (EP) energy, U_{Ep}. U_{Ref} takes on a standard form and the parameters may be available from the literature. This potential is used throughout the EPSR simulation, but is used on its own at the outset to form the molecules (if present) and to get the simulation box into a likely region of phase space for the system being studied (i.e. sensible molecular geometries, no atomic overlap, etc.). U_{Ep} on the other hand does *not* take any standard form and, once the simulation with the reference potential alone has come to equilibrium, is used to guide the atomic and molecular moves in directions that give the closest representation of the diffraction data.

The total potential of the system is represented as $U = U^{(Ref)}+U^{(Ep)}$. Each of these terms can be split into terms related to the separation of individual atoms and molecules, with atoms of different type having different interaction potentials. Thus for example the potential energy between atoms of type α and type β separated by distance r would be given by

$$U_{\alpha\beta}(r)=U_{\alpha\beta}^{(\mathrm{Ref})}(r)+U_{\alpha\beta}^{(\mathrm{Ep})}(r) \tag{2.1.1}$$

with the total potential energy of the system given by

$$U=\frac{1}{2}\sum_{i}\sum_{j\neq i}U_{\alpha(i)\alpha(j)}(r_{\mathrm{ij}}) \tag{2.1.2}$$

where r_{ij} is the separation of atoms (*i*,*j*), and *α(i)* represents the "type" (i.e. whether it is hydrogen, carbon, oxygen, etc.) of atom *i*. The summation in (2.1.2) proceeds over all atom pairs in the system, and the factor of ½ is needed to prevent double counting of atom pairs.

In EPSR there is no correction for long range effects on the potential energy. This is partly because the EP itself cannot be calculated for $r > \sim r_{max}$ (r_{max} is defined in section 2.4), and partly because making the necessary longer range corrections can be time consuming, without giving any real benefit in terms of modifying the local arrangement of atoms. Energy and pressure are calculated in EPSR as well as structural quantities and these can be a useful guide in some cases as to whether the results are sensible. If the pressure is extremely positive it usually means that there is significant atomic overlap somewhere in the system so is likely to imply one or more parameters have not been set correctly. If on the other hand it is strongly negative it would mean the system is trying to collapse on itself, so expect to see voids appearing in the structure and a rise in scattering at low *Q*.

Instead of an accurate long range correction the non-Coulomb part of the reference potentials are truncated smoothly by a suitable function, which at the time of writing is of the form

$$T(r) = \begin{matrix} 1 \quad r<r_{minpt} \\ 0.5\left[1+\cos\pi\left(\frac{r-r_{minpt}}{r_{maxpt}-r_{minpt}}\right)\right] \quad r_{minpt}<r<r_{maxpt} \\ 0 \quad r>r_{maxpt} \end{matrix} \qquad (2.1.3)$$

where r_{minpt} is the point where the truncation function drops below 1.0, and r_{maxpt} is the point where it drops to 0. Typically one might use r_{minpt} = 9Å and r_{maxpt} = 12Å.

Within the program the interatomic potentials are stored as a look-up table that is interpolated at specific *r* values. This speeds up the process of calculating the energy and pressure.

For the Coulomb potentials the "Local Molecular Field" method of truncating the Coulomb potential due to Chen and Weeks [5] is used:-

$$T_c(r)=\text{erfc}\left(\frac{r}{\sigma_c}\right) \qquad (2.1.4)$$

with σ_c a width parameter to be set by the User. Typically σ_c would be the diameter of the largest molecule in the simulation box, but, in EPSR, for convenience it is set to the largest of 3.5Å and $\frac{1}{5}r_{maxpt}$. The first value is deemed to be the smallest that can be used without introducing significant error [5]

2.2 Defining the reference potential.

The starting point of the EPSR simulation is to build an ensemble of molecules whose internal structure reproduces that which can be obtained from the diffraction experiment. Essentially each intra-molecular distance is characterised by an average distance, $d_{\alpha\beta}$, and a width, $w_{\alpha\beta}$, and the intramolecular structure is established by assuming the atoms in each molecule interact via a harmonic potential. The total intramolecular energy of the system is represented by

$$U_{intra}=C\sum_i\sum_{\alpha\beta>\alpha}\frac{\left(r_{\alpha(i)\beta(i)}-d_{\alpha\beta}\right)^2}{2w_{\alpha\beta}^2} \qquad (2.2.1)$$

where $r_{\alpha(i)\beta(i)}$ is the actual separation of the atoms *α*,*β* in molecule *i*, and

$$w_{\alpha\beta}^2 = d_{\alpha\beta} / \sqrt{\mu_{\alpha\beta}} \qquad (2.2.2),$$

with $\mu_{\alpha\beta} = {M_\alpha M_\beta} / {(M_\alpha + M_\beta)}$ the reduced mass of the atom pair *αβ*, and M_α the mass of atom *α* in atomic mass units. *C* is a constant determined by comparing the simulated structure factors with the measurements at large *Q*. A typical value of *C/*2 which seems to give good results is 65/(Åamu^{½}), assumed independent of temperature. The use of an effective broadening function, $w_{\alpha\beta}$, such as defined in (2.2.2) avoids the need to specify and refine individual Debye-Waller factors for each intramolecular distance. Although this is obviously an approximation, assigning individual Debye-Waller factors probably cannot be done unambiguously for a liquid if there are several intramolecular distances to refine. The chosen simplified form does however introduce a degree of realism into the broadening function, namely the width is related to the effective mass of the atom pair involved, as well as the bond strength (~1/$d_{\alpha\beta}$) as it would be in the real molecule.

As discussed in Section 1 the intermolecular reference potential is based on a Lennard-Jones12-6 potential plus effective Coulomb charges where appropriate:-

$$U_{\alpha\beta}(r_{ij}) = 4\epsilon_{\alpha\beta}\left[\left(\frac{\sigma_{\alpha\beta}}{r_{ij}}\right)^n - \left(\frac{\sigma_{\alpha\beta}}{r_{ij}}\right)^6\right] + \frac{q_\alpha q_\beta}{4\pi\epsilon_0 r_{ij}} \qquad (2.2.3).$$

with *n* set to 12. (In the past different values of *n>6* have been experimented with without much improvement over this basic function. Use of a purely exponential repulsive force can lead to problems in Monte Carlo unless an extra core potential parameter is introduced since the Coulomb attraction between atoms of unlike charge becomes infinitely attractive at *r* = 0, which would cause atomic overlap if it were not for the divergence of the Lennard-Jones potential at small *r*. At least the Lennard-Jones form of (2.2.3) includes the dispersion forces correctly.) Here *α, β* represent the types of atoms *i,j* respectively, and if there are *N* different types of atom in the system, there will be *N*(*N*+1)/2 pairs of such interactions. Where the types of the two atoms are different, the well depth parameter, $\varepsilon_{\alpha\beta}$, (in EPSR measured in kJ/mole) and the range parameter $\sigma_{\alpha\beta}$, (in EPSR measured in Å) are given by the usual Lorentz-Berthelot mixing rules [3] in terms of their values for the individual atoms:-

$$\varepsilon_{\alpha\beta} = \sqrt{\varepsilon_\alpha \varepsilon_\beta}; \sigma_{\alpha\beta} = \frac{1}{2}(\sigma_\alpha + \sigma_\beta) \quad . \qquad (2.2.4)$$

Note that further provision to restrict the approach of individual pairs of atoms to one another is provided when setting the reference potential - see later.

2.3 Defining the empirical potential.

A fundamental principle behind setting up the empirical potential is that it must represent only true differences between the simulation and diffraction data: it ideally should not reflect artefacts associated with the statistical noise, systematic errors and truncation effects of the diffraction data. If it did contain these artefacts then it is likely they would be carried over into the estimated distribution functions of the system.

In practice this is quite a difficult goal to achieve and a variety of methods for generating the empirical potential (EP) have been tried over the years. The one that appears to be most successful so far is in the form of a series of power exponential (Poisson) functions:-

$$U^{(EP)}(r)=kT\sum_i C_i p_{n_i}(r,\sigma_r) \qquad (2.3.1)$$

where

$$p_n(r,\sigma)=\frac{1}{4\pi\rho\sigma^3(n+2)!}\left(\frac{r}{\sigma}\right)^n \exp\left[-\frac{r}{\sigma}\right] \qquad (2.3.2)$$

the C_i are real, but can be positive or negative, and σ_r is a width function to be set by the user. The total atomic number density of the simulated system is ρ. You may note that $p_n(r,\sigma)$ has a first moment of $(n+3)\sigma$ and a second moment of $(n+3)\sigma^2$, which means it peaks near $r \sim n\sigma$ for large n with a width that gets gradually larger with increasing n. It is therefore a very natural function to represent an intermolecular potential, which tends to vary rapidly with r at shorter distances and becomes more slowly varying at longer distances. As n becomes very large this function approaches a Gaussian centred on $r= n\sigma$.

These facts are used to generate the values of i in (2.3.1). Essentially a set of radius values, r_i, are selected by the user to correspond with the likely range of the empirical potential and the values of n_i corresponding to these radii are given by

$$n_i=\frac{r_i}{\sigma_r}-3 \qquad (2.3.3)$$

The function $p_n(r,\sigma)$ has an exact 3-dimensional Fourier transform to Q-space:-

$$\begin{aligned} P_n(Q,\sigma) &= 4\pi\rho\int p_n(r)\exp(iQ\cdot r)\,dr \\ &= \frac{1}{(n+2)\left(\sqrt{1+Q^2\sigma^2}\right)^{(n+4)}}\left[2\cos(n\alpha)+\frac{(1-Q^2\sigma^2)}{Q\sigma}\sin(n\alpha)\right] \end{aligned} \qquad (2.3.4)$$

where $\alpha=\arctan(Q\sigma)$. Therefore in EPSR, the coefficients C_i are estimated directly from the diffraction data by fitting a series of the form

$$U_{\mathrm{EP}}(Q)=\sum_i C_i P_{n_i}(Q,\sigma_Q) \qquad (2.3.5)$$

to the data in Q-space (actually the difference between data and simulation – see Section 2.6), and then the coefficients so generated are used to produce the Empirical Potential via equation (2.3.1). The values of n_i in (2.3.5) are generated by an analogous formula to (2.3.3). This eliminates the need to perform a direct Fourier inversion of the diffraction data and largely eliminates many of the problems associated with noise in the data and truncation ripples.

In an exact world σ_Q in (2.3.5) would be identical with σ_r used in (2.3.1). In fact it is possible to induce further smoothing on the empirical potential by using $\sigma_r = f\sigma_Q$ in (2.3.1), with f=4 in the current EPSR program. This means the values of n_i in (2.3.5) will not be the same as those used in (2.3.1), so the program does not use an exact reconstruction of the data to generate the potential. In practice the best results are obtained with σ_Q typically an order of magnitude smaller than the spacing between the r_i values, so the primary effect of using $f > 1$ is to broaden the reconstructed function (2.3.1) compared to what it might otherwise have been and so make an even smoother empirical potential – the change in width produces very little discernible distortion of the resulting function. (Note that the width parameters (σ_r, σ_Q) being described here have nothing to do with the Lennard-Jones parameters $\sigma_{\alpha\beta}$ of the previous section. (σ_r, σ_Q) are simply width parameters which are the same for all atom pairs.) Typically use of σ_Q=0.02Å in (2.3.5) is found to give satisfactory results, with σ_r=0.08Å in (2.3.1) for the reconstruction. Note also that the functional form (2.3.1) is such that gradients (forces) and higher derivatives of the potential have an analytical expression.

With EPSR25 it is possible to also use a Gaussian representation of the EP, instead of the Poisson representation described here. A 3-dimensional Gaussian also has an exact representation in both *Q* and *r* space in the same way that the Poisson function does. In this case, however, the width of the Gaussian functions does not increase with distance, which is useful when refining the structure of a crystal, where the longer correlations do not decay exponentially as they do in a liquid or glass.

2.4 The uniform atom distribution.

For an infinite system the radial distribution function $g(r)$ is defined as the ratio of the local density of atoms at a distance *r* from an atom at the origin, $\rho(r)$, to the average density of atoms in the whole system, ρ, i.e. $g(r)=\frac{\rho(r)}{\rho}$. For the computer simulation box the atoms do not proceed to infinity, but are defined within a box (of dimension say *D* assuming it is a cubic box), although they can of course move freely through the sides of the box. The infinite extent of the system is then modelled by repeating this box indefinitely throughout space. Such an infinitely repeating lattice of boxes would give rise to Bragg peaks in the calculated diffraction pattern, but the "minimum image convention" [3] states that provided we restrict ourselves to looking at distances *less* than a distance $r_{max}=0.5\,D$, then we will only "see" the local environment of individual atoms and not be aware of the longer range periodicity. Thus effectively a sphere of this radius r_{max} is drawn around each atom when calculating $g(r)$ **.** This works quite well provided *D* is chosen large enough that the local density fluctuations induced by an atom or molecule in the box have effectively damped out for $r>r_{max}$.

In practice, in a computer simulation, the number of atoms at distance *r* from any given atom between *r* and $r+\Delta r$ is counted, *N(r).* If the atoms were uniformly distributed throughout the simulation box, this distribution, called the "uniform atom distribution", would simply be proportional to r^2, namely

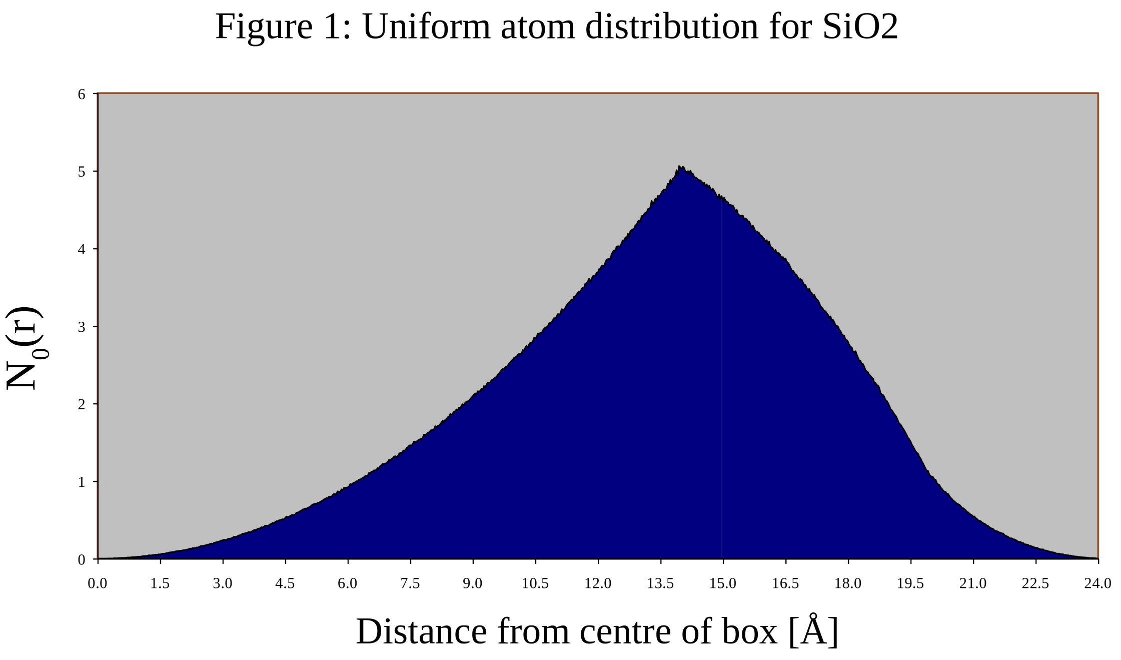
$$N_0(r)=4\pi\rho r^2\Delta r \qquad (2.4.1)$$

out to $r=r_{max}$. In that case, $g(r)=\frac{N(r)}{N_0(r)}$, which is the formula used to calculate $g(r)$ in EPSR.

In principle it is possible to look at the distribution of atoms beyond this sphere, $r>r_{max}$, effectively including all the atoms within the simulation box. However to calculate beyond r_{max} the uniform atom distribution becomes a different function of distance from the central atom, since only the region *inside* the simulation box can be included. The maximum distance that can be calculated is into the corner of the box, which for a cubic box is $\sqrt{3}r_{max}$. Provided one is careful only to count each atom in the box once when estimating there is nothing inconsistent with the minimum image convention by doing this.

What happens of course is that as you approach the corners of the box the numbers of atoms involved become small, so that the statistics at large *r* are poor. This can introduce large truncation oscillations in the calculated structure factors. Thus in the current version of EPSR, $g(r)$ is only calculated to half the box dimension (or the equivalent maximum distance in the case of a non-cubic box). If a larger distance is required due to the type of system being studied, the only way is to use a larger simulation box. (Note that r_{max} will not in general be the same as the range of the potential r_{maxpt} and will normally be significantly larger than r_{maxpt}.).

Figure 1 shows an example of this uniform atom distribution for a simulation of silica which used a cubic box of size of 27.9988Å, atomic number density 0.06834atoms/Å^3, and step size 0.03Å, as described in the Examples, Section 7.3. It can be seen how the distribution rises quadratically until the edge of the box is reached (13.9994Å), then decays abruptly beyond this as a smaller and smaller volume is sampled into the corners of the box, eventually disappearing at $r = \sqrt{3} \times 13.9994 = 24.2477$Å.



If it is really necessary to include all the atoms in the simulation box, then the only way to do this accurately is to use Bragg's law to calculate the structure factor at specific (hkl) values, then smear these “Bragg” peaks to create a continuous F(Q). Facilities to do this are now incorporated into EPSR.

2.5 Running the simulation

As described above there are four different kinds of moves within the EPSR simulation, namely individual atom moves where atoms are moved relative to one another within the molecule, whole molecule rotations and translations, plus other intramolecular moves, such as rotations of particular headgroups or sidechains. Currently there are no moves which involve change of symmetry.

For individual atom moves only the change in the intramolecular potential energy, $\Delta U_{\mathrm{intra}}$, is used to accept or reject a move. Thus the probability of acceptance of an intramolecular move within the Monte Carlo scheme is based on the value of $\exp[-\Delta U_{intra}]$. There is no thermal factor in this sampling, in order to simulate the zero-point disorder, which is temperature independent to first approximation. In the event that the assumed molecular geometry does not fully concur with what the diffraction data imply, it is possible to refine the assumed intramolecular distances to reduce any misfit to a minimum.

For whole molecule moves, and for other intramolecular moves, the usual Boltzmann thermal factor is used outside the intermolecular potential energies. In principle whole molecule moves should not involve the relative movement of atoms within the molecule. However round-off errors can accumulate over a number of such moves, so the intramolecular potential is included when calculating the total energy change. Thus the probability of acceptance of a whole molecule move, or an internal headgroup rotation move is based on the value of

$$\exp\left[-\left\{\Delta U_{\mathrm{Intra}}+\frac{1}{k_B T}\left(\Delta U_{\mathrm{Ref}}+\Delta U_{\mathrm{Ep}}\right)\right\}\right] \quad (2.5.1),$$

where for whole molecule moves ΔU_{intra} would normally make only a small contribution to the total energy difference before and after the move.

Individual atom moves amount to disordering the molecules without reference to the surrounding molecules so they result in a net disordering of the intermolecular order in the system. Therefore to prevent this disorder from accumulating there are typically 100 or 200 whole molecule moves for each individual atom move. In this way the zero point disorder of the molecules is simulated, while maintaining the intermolecular order in equilibrium. EPSR is therefore a rigid molecule simulation, with however every molecule slightly different from the next, and over the course of a simulation the precise geometry of these molecules will change slightly many times, although the average molecular geometry remains unchanged.

Once the simulation with the reference potential alone has equilibrated, i.e. the calculated distribution functions and structure factors become stationary and no longer change as the simulation proceeds, the empirical potential is introduced. This is calculated by taking the difference in Q space between diffraction data, *D(Q)*, and the simulated structure factor, *F(Q)*, and fitting a function of the form (2.3.5) to this difference to give the set of coefficients C_i. (The current EPSR code uses a second Monte Carlo loop to estimate these coefficients – this may not be the most efficient way to do it, but it is at least straightforward to program and does not involve any proprietary routines which would be difficult to distribute. In practice estimating the EP coefficients is not a time consuming part of the calculation.) These coefficients are then used in (2.3.1) to generate the empirical potential.

If isotope substitution has been performed then it is likely there will be several datasets to fit, so that an empirical potential will be need to be generated for each of the site-site distributions in the system. This is fine if, as in the case of say water or ammonia or a single component material such as liquid sodium, it is possible to obtain all the specific site-site distributions separately.

Very often however it is desired to fit the total diffraction pattern, without separating it into partial structure factors. Or more often, even with isotope substitution, there are not enough substitutions possible to give all the partial structure factors of the system. Many materials currently being investigated with neutrons and x-rays will fall into this category. Therefore a different procedure is needed to set up the empirical potential for different pairs of atom types in these cases. The present version (since 2006) uses a somewhat different procedure here compared to previous versions so it is described in detail below.

2.6 Refining the Empirical Potential – introducing the data.

If there are *J* distinct atomic components in the system being studied then there are $N = J(J+1)/2$ distinct site-site partial structure factors (PSF) and radial distribution functions (RDF) to be determined. We will assume that we have measured *M* diffraction datasets, $D_i(Q)$, each with a different isotopic composition, or one may be an x-ray diffraction dataset on the same material. The fit to the *i*th dataset of a particular experiment can be represented by a weighted sum over all the pairs of atom types of the relevant simulated partial structure factors $S_{\alpha\beta}(Q)$:

$$F_i(Q)=\sum_{j=1,N} w_{ij} S_j(Q) \qquad (2.6.1),$$

where *j* represents one of the atomic pairs of components (*αβ*) and the weights, w_{ij} are determined from the respective product of atomic fractions and neutron scattering lengths or x-ray form factors for that particular sample. For example for un-normalised neutron data $w_{ij}=(2-\delta_{\alpha\beta})c_\alpha c_\beta b_\alpha b_\beta$. The *b* values will in general be different for different atom types and datasets and for x-rays will have their own *Q* dependence[1]. In order to generate a perturbation to each of the site-site potential functions we need an inversion to the matrix w_{ij}. If there are enough

1 See Section 5.2 for description of how EPSR deals with the *Q* dependence of the X-ray form factor.

isotopes or x-ray datasets to completely separate all the partial structure factors, then this is no problem in principle, but, as explained above, such a luxury is rarely available in practice, and in any case, even if it were, there would remain questions about how reliable are particular datasets, especially when one or more of the PSFs make only a weak contribution to the total diffraction pattern. The question is therefore how to invert the matrix w_{ij} in general? For most systems of experimental interest the matrix of coefficients is ill-determined, making direct inversion unreliable.

In order to cover all of the situations likely to be encountered in real experiments, we assign a confidence factor (called the 'feedback' factor in the program), *f*, to the data, where $0 \le f < 1$, and form the modified weights:

$$w'_{ij} = f w_{ij}; \quad \text{for } 1 < i \le M \qquad (2.6.2).$$

In addition to these we form an additional, diagonal array of weights:

$$w'_{ij} = (1-f)\,\delta_{(i-M),j}; \quad \text{for } M < i \le (M+N) \qquad (2.6.3)$$

Effectively this additional set of weights is equivalent to saying we accept the data with confidence *f*, and the simulation with confidence (1-*f*). (In the program, *f* is called the feedback factor and is typically set to 0.9.) This leads to an overdetermined matrix in every case (*N* columns × (*M*+*N*) rows), provided $0 < f < 1$.

We then seek the inverse of this matrix, w_{ji}^{-1} ,such that the (*M*+*N*) × (*M*+*N*) matrix formed from $P_{ii'} = \left(\sum_{j=1,N} w'_{ij} w_{ji'}^{-1} - \delta_{ii'} \right)$ has a minimum norm, giving the least squares solution for w_{ji}^{-1} in the case of an overdetermined set of linear equations as here. This problem can be solved by standard techniques (EPSR again uses an iterative Monte Carlo loop to achieve the inversion to avoid the need to use proprietary software. Although slower than more refined techniques the Monte Carlo method is very robust at finding solutions.) Note that if the data are incomplete it is imperative that $f < 1$, otherwise the procedure will loop indefinitely without finding a solution. The solution can be checked from the requirement that the matrix $P_{j'j} = \left(\sum_{i=1,M+N} w_{j'i}^{-1} w'_{ij} \right)$ must be unitary.[2] Since the matrix w'_{ij} is always well determined from (2.6.1) and (2.6.2) (provided the feedback factor $f < 1.0$) there should never be a problem with singularities.

Thus the complete algorithm for calculating the EP can now be written down. At the beginning of the mth iteration of the algorithm each distinct pair of atom types, *j*, will have a set of coefficients, $C_{k,m}^{(j)}$, which are used to form the empirical potential for that atom pair. Following (2.3.1) the EP at the beginning of the mth iteration for any particular pair of atoms, *j*, is determined from

$$U_m^{(j)}(r) = kT \sum_k C_{k,m}^{(j)} p_{n_k}(r, \sigma_r) \qquad \mathbf{(}2.6.4\mathbf{)}$$

(At the outset, with *m* = 0, the coefficients $C_{k,m}^{(j)}$ are set to zero.) After the mth iteration the differences between data and fit, $(D_i(Q) - F_i(Q))$, are calculated, and represented by a sum of the form (2.3.5). This gives rise to a set of difference coefficients, $C_k^{(i)}$, one for each supplied dataset, which change as the simulation proceeds – ideally they should go to zero when the simulation approaches the data closely. These difference coefficients are then accumulated in the potential coefficients:-

2 This matrix is printed out at the end of the matrix inversion calculation to demonstrate that the inversion proceeded correctly.

$$C_{k,m+1}^{(j)} = C_{k,m}^{(j)} + \sum_{i=1,M} w_{ji}^{-1} C_k^{(i)} \qquad (2.6.5).$$

The revised values, $C_{k,m+1}^{(j)}$, are now used in (2.6.4) to form a new version of the EP and the simulation is run again. This whole cycle is repeated a large number of times until one of two conditions is reached. Either the difference coefficients, $C_k^{(i)}$, become insignificantly small so that the empirical potential does not change any more, or else the modulus of the empirical potential energy, (now defined as discussed in Section 2.6.1 below) reaches some predefined limit. The latter is normally the case and happens because there are systematic errors in the data which cannot be fit by any potential energy function. In that case the EP would increase in amplitude indefinitely if it were not capped, and might introduce artefacts into the calculated distribution functions. Thus in the EPSR approach there is still scope for subjectivity on the part of the experimenter: they normally will need to set a cap for the empirical potential energy modulus if systematic error is present.

Obtaining an inverse to our matrix of weight coefficients means we can in the spirit of the above also write down our best estimate of the partial structure factors based on our relative confidence in the data and the simulation:

$$T_j(Q) = \sum_{i=1,M} w_{ji}^{-1} D_i(Q) + \sum_{i=M=1,M+N} w_{ji}^{-1} S_{i-M}(Q) \qquad (2.6.6).$$

These estimated partial structure factors can then be compared with the simulated partial structure factors. The estimated partial structure factors can further be Fourier transformed to real space to give estimates of the site-site radial distribution functions.

It will be seen from (2.6.5) that the empirical potential is cumulative: unlike RMC where the object is to minimize chi-square, the empirical potential develops amplitude and structure as the simulation proceeds thereby bearing a memory of what shape it needs to satisfy in order to fit the data.

Once the EP has stopped changing, or the absolute energy of the EP has reached its specified limit (see sections 2.6.1 and 5.5 for descriptions of how this achieved in EPSR25), the simulation can be used to extract ensemble averages of required quantities. Some of the more common distributions that can be calculated are described in Sections 5 and 6. Our next task however is to look at how to run the simulation and associated routines from EPSRshell.

2.6.1 Defining the amplitude of the Empirical Potential

Successive versions of EPSR have used slightly different definitions of the absolute amplitude of the Empirical Potential (EP). In EPSR25 an attempt has been made to generate a definition that accurately reflects the ability of the EP to modify the atomic distribution functions, while at the same time guarding against the possibility of the EP becoming so large that the atoms and molecules become stuck and refuse to diffuse. At the same time, the definition has to be such that using the same potential in a system with a different temperature and density as the one for which it was generated gives the same amplitude. Finally, it should ideally not change from version to version, causing minor irritation to the user trying to understand what is going on!

None of these features were characteristic of previous versions. In EPSR25 an attempt has been made to use a definition for the amplitude of the EP which can be carried on from one version to the next, so that the values of **ereq**, the parameter in the EPSR input file that controls the amplitude of the EP, are consistent from one version to the next. The figure below shows one of the EP's from an EPSR simulation of methanol. The units, as always in EPSR, are kJ/mole.

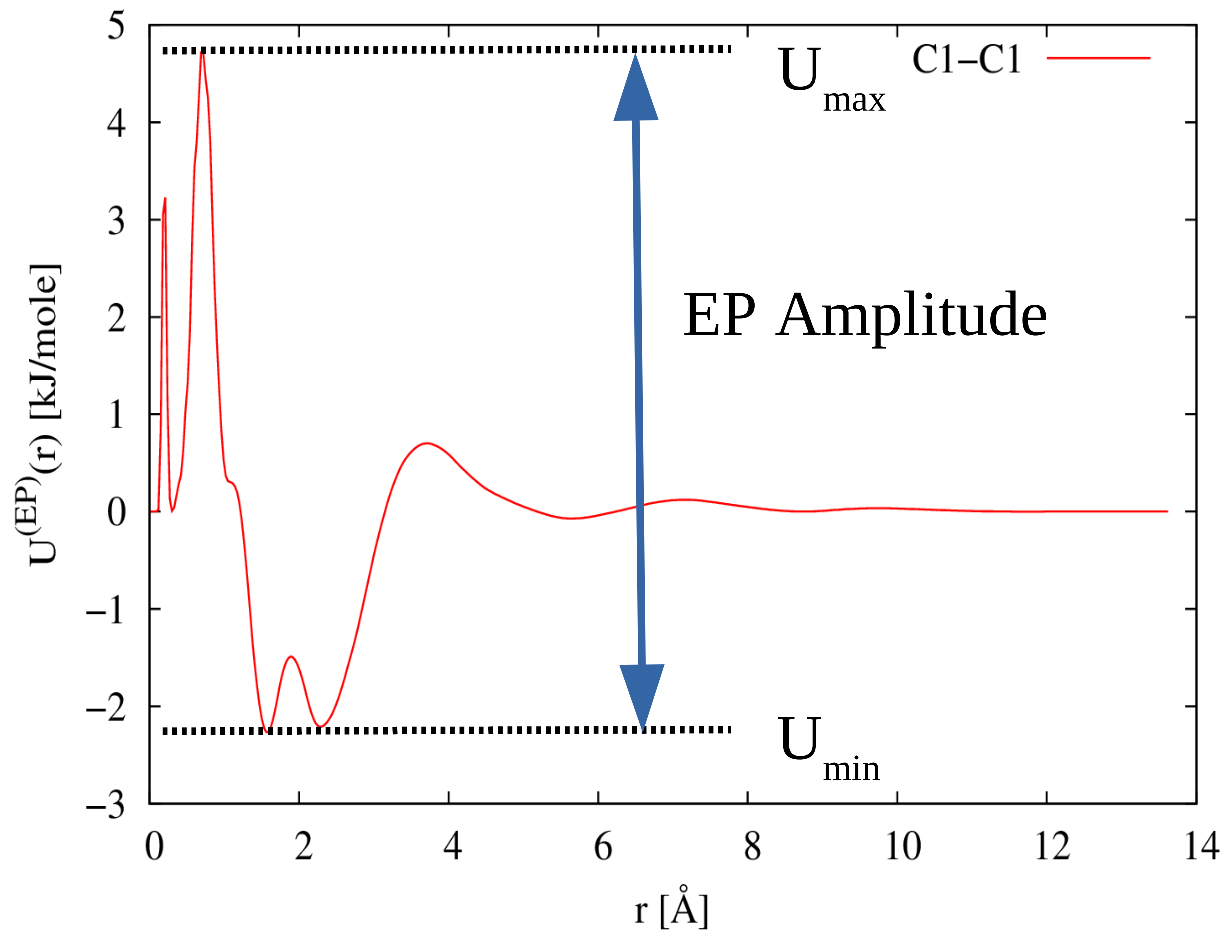


Figure 2.6.1 Definition of the amplitude of the Empirical Potential for the C1-C1 atom pair in liquid methanol.

The idea is to measure, for each atom type pair, the maximum and minimum value of the potential, U_{max}, and U_{min}, from which the amplitude of the EP for that particular atom type pair can be calculated, $A^{(EP)}(j)=U_{max}(j)-U_{min}(j)$. Hence in the example shown here, the amplitude of this EP would be ~7 kJ/mole. Comparing all such atom type pairs, the amplitude of the EP is then listed as the maximum of the all the individual amplitudes, $A^{(EP)}=\max(A^{(EP)}(j))$. This definition does not depend on the temperature, density or *g(r)*, unlike previous versions.

The significance of this definition is that, according to (2.5.1) changes of energy, Δ_{Ep} are divided by *kT*, when deciding whether to accept or reject a move. At 300 K this corresponds to ~2.5 kJ/mole. Hence if $A^{(EP)} \ll kT$ then we would expect the EP to be ineffective at changing the structure of the material. On the other hand, if $A^{(EP)} \gg kT$, then atomic motion may become inhibited by the large variations in the potential energy landscape, and so the simulation could become frozen. Hence an obvious goal has to be that $A^{(EP)} \sim kT$ - this way, the EP can both affect the structure, while at the same time allowing atom diffusion to continue. We can anticipate that $A^{(EP)}$ might vary quite a lot around this value, but it should be in the region of kT, unless we have good reason to allow otherwise.

2.7 Additional terms to the reference potential.

2.7.1 Atom overlap forces

There is also scope within the input files to provide an $n=0$ coefficient in (2.6.4) which will give rise to a short range repulsive force: this is useful for ensuring unphysical overlap of atoms not otherwise constrained by the reference potential does not occur. This repulsive force is not determined from the diffraction data, but is instead determined by setting a minimum distance, $r_{\min}^{(j)}$, for each atom pair (*j*) radial distribution function. Any penetration below this minimum distance causes the corresponding *n*=0 coefficient to increase in amplitude, which in turns increases the repulsive potential between this pair until the offending intensity in *g_j(r)* is removed. If no intensity is found below this minimum distance then the *n*=0 coefficient is gradually decreased towards zero so that it does not otherwise interfere with the rest of the RP. This repulsive potential is treated as part of the reference potential, so will appear even if the amplitude of the EP is set to zero. Its effect can be controlled by specifying the minimum allowed distance between individual atom pairs in the EPSR input file. A value of 0 for this minimum distance means that no repulsive potential is calculated for that pair of atoms.

The form of this short range repulsive force is either exponential or harmonic, and is controlled by a specific width, σ_{rep}, and limiting gradient, g_{lim}, set by the user:-

$$U_{rep}^{(j)}(r)=\begin{matrix} C_{\min}^{(j)}\exp\left(\Delta_{\min}^{(j)}(r)\right), & \Delta_{\min}^{(j)}(r)<\ln g_{\lim} \\ C_{\min}^{(j)} g_{\lim}\left(1+\Delta_{\min}^{(j)}(r)-\ln g_{\lim}\right), & \Delta_{\min}^{(j)}(r)\geq \ln g_{\lim} \end{matrix} \qquad (2.7.1),$$

where $\Delta_{min}^{(j)}(r)=\frac{\left(r_{min}^{(j)}-r\right)}{\sigma_{rep}}$ in the exponential representation, and

$$U_{rep}^{(j)}(r)=\begin{matrix} 0, & \Delta_{min}^{(j)}(r)<0 \\ C_{min}^{(j)}\frac{\left(\Delta_{min}^{(j)}(r)\right)^2}{2}, & 0\leq \Delta_{min}^{(j)}(r)<g_{\lim} \\ C_{\min}^{(j)}\frac{1}{2}g_{\lim}\left(2\Delta_{min}^{(j)}(r)-g_{\lim}\right), & \Delta_{min}^{(j)}(r)\geq g_{\lim} \end{matrix} \qquad (2.7.2),$$

with $\Delta_{min}^{(j)}(r)=\frac{\left(r_{min}^{(j)}-r\right)}{\sigma_{rep}}+1$ in the harmonic representation. The linear terms (bottom lines) in (2.7.1) and (2.7.2) are used at short distances to help prevent the simulation becoming jammed because of transient large values in the repulsive potential. In the program, if g_{lim} is set to 0 (the default value) or less, then the bottom line in 2.7.1 or 2.7.2 is not used – in other words the limiting gradient is set to infinity. The weighting coefficient, $C_{min}^{(j)}$ can either be set by the user or allowed to fluctuate, the value being set automatically by the program to the extent needed to prevent the specified atomic overlap.

2.7.2 Container forces

In addition to the minimum distance specification EPSR25 allows a maximum distance specification for any given atom pair – the atoms are regarded as being "contained" by each other. This is useful in heterogeneous systems when it is required to force a particular topology on the system – atoms of particular type must not penetrate or move too far away from a specified surface, for example. These container forces are specified in exactly the same way as (2.7.1) or (2.7.2), but with $C_{min}^{(j)}$ and $r_{\min}^{(j)}$ replaced with $C_{max}^{(j)}$ and $r_{max}^{(j)}$, and $\Delta_{\max}^{(j)}(r)$ replaced with $\Delta_{\max}^{(j)}(r)=\frac{\left(r-r_{\max}^{(j)}\right)}{\sigma_{\text{rep}}}$ for the exponential representation and i $\Delta_{\max}^{(j)}(r)=\frac{\left(r-r_{\max}^{(j)}\right)}{\sigma_{\text{rep}}}+1$ n the harmonic representation. If i $r_{max}^{(j)}$ s set to 0 (the default value) for any atom pair the container force is not calculated for that atom pair. As with the atomic overlap force, $C_{max}^{(j)}$ can either be set by the user or allowed to adjust automatically to the extent needed to force the specified containment.

Note that, unlike all the other forces in EPSR, container forces are, for obvious reasons, NOT subject to any form of truncation function – if they did they would not work! The container potential will increase steadily as the separation of any pair of atoms, *(j)*, increases above $r_{max}^{(j)}$.

This also means that they have to be calculated in a different way to other forces. All the other forces in EPSR are calculated for *every* instance of a pair of atoms of the specified type and the corresponding potential energies summed. However container forces are only applied if NONE of the instances of a particular atom pair lie within $r_{max}^{(j)}$. For example suppose it is required to make sure all atoms of type A lie within a specified distance of atoms of type Q. For any given instance of atom *i* of type A(*i*), all the pairs A(*i*)-Q have to be examined, and the container potential is only calculated if *none* of these pairs is inside the specified distance. This behaviour means the container force is transitory and can be on or off depending on the state of the molecules in the simulation box. As a result it is not deemed to contribute to the overall energy and pressure of the simulation even though it is used to accept and reject molecular moves.

2.7.3 Additional terms

It is sometimes necessary to introduce additional contributions to the reference potential to force particular features – for example a pronounced correlation peak at a particular position in a particular site-site distribution. This can be achieved by specifying the position, width and height (PWH) of up to 5 Gaussian terms in the reference potential. These have the form:-

$$U_{Gauss}^{(j)}(r)=\sum_{i=1}^{5} h_i^{(j)} \exp\left[-\frac{1}{2}\left(\frac{p_i^{(j)}-r}{w_i^{(j)}}\right)^2\right] \qquad (2.7.3)$$

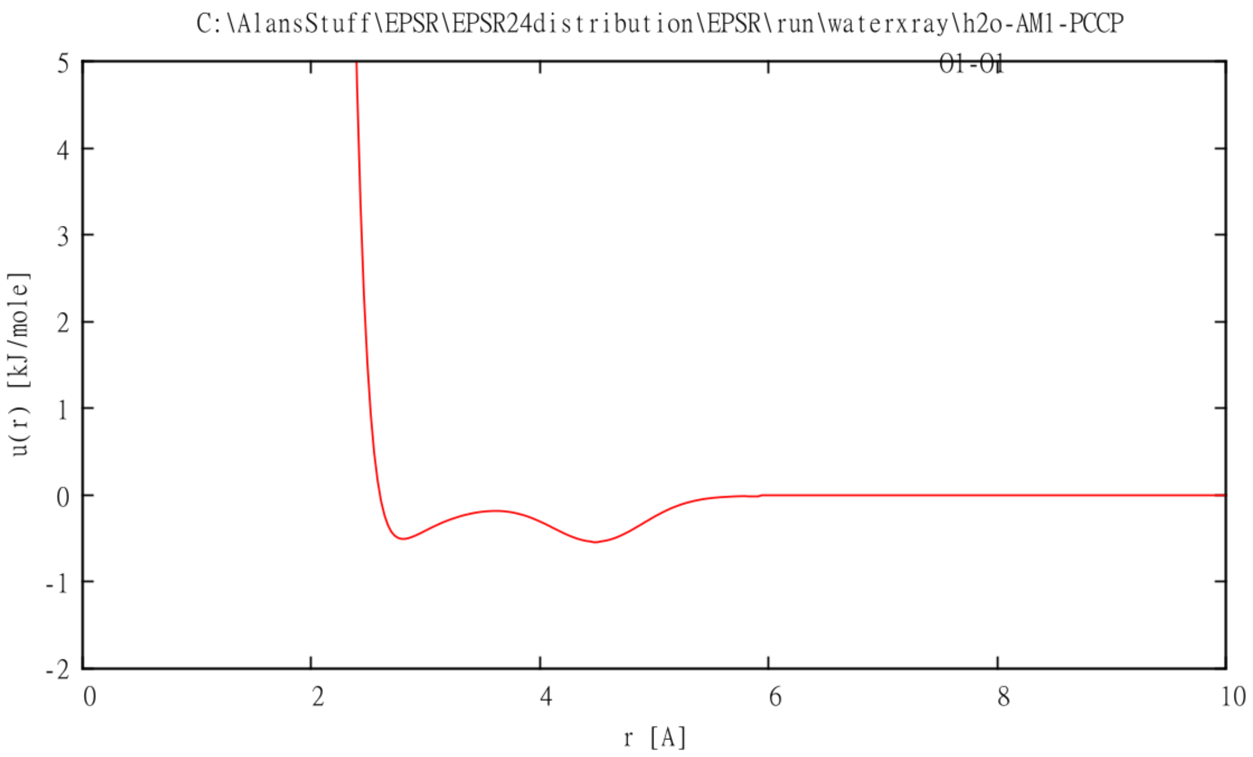
The amplitude of each term can be either positive or negative depending on whether a peak is to be enhanced or removed. If the specified position or width of any of these terms is < 0, then the corresponding site-site empirical potential is not refined.

In addition to these terms, the first term can be set to a modified version of the Buckingham exponential-6 potential [6]. The aim here is provide a realistic London dispersion force [7] at longer distances, with a Gaussian repulsive term at short distances. This term is only applied as the first term in the series (2.7.3) and then only if specified.

$$U_{Gauss6}^{(j)}(r)=\left|h_1^{(j)}\right|\left[\exp\left[\frac{-1}{2\left(w_1^{(j)}\right)^2}\left(\left(\frac{p_1^{(j)}}{r}\right)^2-1\right)\right]-\left(\frac{p_1^{(j)}}{r}\right)^6\right] \qquad (2.7.4)$$

The width parameter, $w_1^{(j)}$, here controls the steepness of the repulsive term in this potential. The form (2.7.4) ensures this Gauss6 potential goes to 0 at $r=p_1^{(j)}$. The dispersion term is rounded off at low *r* to prevent divergences near *r* = 0. Figure 2 gives an example of a combined Gauss6 potential at *r* = 2.6Å, amplitude 1.0kJ/mole, and width 0.2, plus a Gaussian potential at *r* = 4.5Å, width 0.4, and amplitude -0.5kJ/mole. Note that the amplitude of 1.0 kJ/mole for the Gauss6 potential does not give a well depth of exactly 1.0 kJ/mole, but approximately half this value.

Figure 2. Example of Gauss6 and Gaussian additional potentials



The dispersion force is truncated at low *r* to avoid any divergences there. The Gauss6 term to the reference potential can be used in place of the standard Lennard-Jones potential and has the advantage that the hardness of the repulsive potential can be controlled to some degree: Lennard-Jones is often too hard to give the best possible fits in many cases. A disadvantage is that the Lennard-Jones parameters are often well characterised for many atom types, whereas the wider use of the Gauss6 term will require setting up a table of appropriate values.

2.7.4 Three-body forces

Molinero and Moore have modelled water using a combined short-range two- and three-body potential [8], which is based on an original idea from Stillinger and Webber [9] developed for amorphous network elements. An analogous formulation has been introduced into the reference potential in EPSR, but at the time of writing the efficacy of this procedure has not been proven or fully tested, so for the time being this work should be regarded as unsupported pending further investigation. If it is felt the RP needs to be improved compared to what is currently available with the Lennard-Jones terms, it is suggested the user employs the additional Gaussian and/or Gauss6 potentials to emphasize particular interatomic distances.

3. EPSRshell

The introduction of EPSRgui means much of EPSRshell has been superseded, and the user wishing to use EPSRgui should follow the instructions associated with the GUI. Nonetheless EPSRshell will continue to be supported as it allows operation of EPSR from the command line, which is probably not as “user friendly as the GUI, but is more versatile in some regards. In any case it is worth understanding a little bit about EPSRshell since many of the naming conventions adopted here apply to EPSRgui as well.

3.1 Introduction – EPSRshell menus

As explained in the introduction EPSRshell is an attempt to provide an operating environment in which to setup, run, modify, run auxiliary routines, and display the results from an EPSR simulation. It works from a series of menus, and the list of commands and variables available within each menu is normally available by typing “help <CR>” or “? <CR>”. (In this manual pressing “Enter” will be represented by “<CR>”.) Hopefully the operation of this scheme, although it seems complicated when written down, will rapidly become familiar: there is some very limited help information provided in the program, but the aim is that the need to refer to this manual frequently for instructions should diminish with experience.

EPSRshell runs in the folder from where it is started. This will be referred to the “home folder”. The home folder must contain the file “system_commands.txt”, which contains the definitions of the system commands appropriate to the operating system in which the program is running. If this file does not exist then the program will start but not run correctly. See Section 9 for a full list and description of files needed to run EPSRshell.

EPSRshell can access and modify data stored in another folder. This will be referred to as the “working folder”. At the present time this has to be on the same disk as the home folder, but otherwise can be anywhere on that disk that is accessible. None of the files in either the working folder or the home folder that are to be accessed by EPSRshell should be set to “read-only”. The working folder will ideally contain a file, “plot_defaults.txt” which contains the current list of plotting types and their values appropriate to the data contained in this folder (see Section 3.6). This file does not need to exist at the start, and it will be copied from the distribution folder if it does not exist - creating one from scratch could be VERY tedious. The plot options can always be modified and added to as described in Section 3.6.

Each command line for EPSRshell is read in as a character string, which is normally parsed into 1 or more words. 1 to 5 spaces can be used to separate words or values in the command string. Only spaces are used to delineate words: any other symbol will be treated as part of the variable value. More than 5 spaces and the program will ignore the rest of the line. When commands are input from the input file of a particular program (not a script file) the maximum number of spaces between words is 11 instead of 5: this is so that a formatted write can be used when writing the file, making it easier for the user to read and edit the file directly if they want to.

There is no obvious distinction between commands and variables in the menus other than by what they do when typed. Commands will generally do something if typed, whereas variables simply set that particular variable if a value is specified. Most of the menus except the main menu use a common symbol processing routine so they all have a similar look and feel. Apart from the menu in which the program starts (the main menu) one can step through the list of commands or variables simply by pressing “Enter”. Or else if you want to skip forwards or backwards several commands or variables you can type the name of the command or variable, optionally followed by a value to change to if that command or variable is expecting a value. Useful commands for moving around the menu include:-

<CR>	Pressing “Enter” on its own signals that you accept the present value of the current variable or command. The program then lists the next variable or command in the sequence.
<value><CR>	Sets the new value of the current variable. Each variable or command is preceded by a brief description (except in the main menu, where you have to type “?” to get the list). Note that some variables may require two or more values – these have to be entered with spaces between. If any are missing then the program which uses that variable may use the default value if defined but possibly could develop a fault or potentially crash when running. If it does so, it will immediately revert back to the shell EPSRshell and you can try to find out why it happened. After changing a value the program lists the new value again, so that you can either accept it (by pressing <CR>) or rejecting it (by pressing “n”).
n<CR>	(No) Signals you do NOT accept the present or new value of the current variable or command and wish to revert to the old value. Obviously this will only work if a value has been changed.
u<CR>	(Up) enables you to proceed backwards line by line through the list of commands or variables.
<command or variable name> <optional value><CR>	jump to the specified command or variable, changing its value only if a new value is specified. (Remember at least 1 space is needed between the command or variable and its value.) An exception to this rule occurs within the plot menu. Here the command “p <CR>” on its own will re-plot the most recently plotted graph.
save<CR>	will save the current set of commands or variables to a file. If this involves overwriting an existing file, you will be prompted whether you want to do this or not, and if not, asked for the new name of the file to save to. Again the plot menu is the exception here: the list of plot commands and variables is always stored in a file called “plot_defaults.txt” in the working folder which you are currently accessing. The main menu has no “save” command.
e<CR>	(Exit) exits the current menu, saving anything that has changed since the menu was entered. If this involves overwriting an existing file, you will be prompted whether you want to do this or not, and if not, asked for the new name of the file to save to. Pressing “e” in the main menu will exit from EPSRshell and return you to whatever system you called EPSRshell from.
q<CR>	(Quit) quits the current menu WITHOUT saving any changes that are made. Quitting like this does not give you the option of saving, so use with caution. “q” in the main menu is the same as “e”.

IMPORTANT NOTE: EPSRshell input is case sensitive. All the commands and variable names are in lower case so if you use upper case (e.g. by accident) it is likely the shell will not recognise the command or variable. Some file extensions use mixed upper and lower case, but as these are set by the program and cannot be altered. The only time when the case is not important is when specifying filenames, since the Windows system ignores case when looking for files. On LINUX of course it is essential to use the correct upper and lower case sequence on filenames.

3.2 File naming conventions

As a general rule the input and output files for EPSR and its auxiliary routines will have a double extension of the form <filename>.<program>.<filetype>. The file is referred to within EPSRshell by its <filename>, with the <program> and <filetype> extensions set by the context in which the file is being used. There are some exceptions to this rule. The atom file is always referred to with extension **.ato**, with no program extension, as are the **.pcof** (potential coefficients file), and the diffraction data files. The latter can have any extension. Other files that do not have a program extension are associated with the makemole routine (see later), namely **.mol** and **.atm** files

Other than these four file types, the current program extensions are:-

.EPSR	EPSR input and output files
.NWTS	neutron weights input and output files
.XWTS	x-ray weights input and output files
.PARTIALS	partials input and output files
.TRI	triangles input and output files
.TOR	torangles input and output files
.COORD	coord input and output files
.CHAINS	chains input and output files
.RINGS	rings input and output files
.CLUSTERS	clusters input and output files
.VOIDS	voids input and output files
.SHARM	sharm input and output files
.SDF	sdf input and output files
.PLOT2D	plot2d input files
.PLOT3D	plot3d input files

The allowed filetypes are:-

.f01	EPSR – simulated PSF
.s01	EPSR – simulated intra-molecular PSF
.q01	EPSR – estimated partial structure factors “data”
.d01	EPSR – difference PSF “data” minus simulated PSF
.u01	EPSR – simulated diffraction data F(Q)
.t01	EPSR – supplied diffraction data D(Q)
.v01	EPSR – difference D(Q) - F(Q)
.g01	EPSR and PARTIALS – simulated site-site g(r)s
.r01	EPSR – Fourier transform of PSF “data”
.y01	EPSR – simulated intra-molecular site-site g(r)s
.x01	EPSR – simulated f(r) (FT of F(Q))
.w01	EPSR – f(r) (FT of D(Q))

.o01	EPSR – site-site reference potentials, including Morse and Gaussian terms
.p01	EPSR – site-site empirical potentials, including exponential repulsive terms
.z01	EPSR – running coordination number for each site-site RDF. This is calculated assuming the first atom of the pair is at the origin.
.inp,inpa	EPSR input files. The .inpa file is used only to store some accumulators and the current EP difference coefficients and does not need to be modified by the user.
.erg	EPSR – list energy, pressure, R-factor, and absolute energy at each iteration of the simulation, plus a number of other variables which assess the progress of the simulation.
.terg	EPSR – shows the mean energies associated with each atom type pair, separated into contributions from the reference potential, empirical potential, and total potential.
.uni	EPSR – uniform atom distribution.
.out	EPSR – list of diagnostic values from the simulation.
.dat	input files to all the auxiliary routines, except **plot2d** and **plot3d**
.txt	input files for **plot2d** and **plot3d**.
.h01, .h02, etc.,	output files for SHARM and SDF coefficients
.n01	output files for CLUSTERS, CHAINS, COORD, RINGS distribution functions
.c01	output files for TRIANGLES and TORANGLES angle distributions
.wts	output files for NWTS and XWTS

Note that normally up to 100 different distribution functions are allowed in each of these files (this gives 201 columns, assuming 1 column for the *x* values, and 2 or more columns for the distributions plus their standard deviations. Thus if more than 100 values are required, the extension number is incremented, 02, 03, etc. Currently this mainly occurs with the SHARM coefficients, but could happen with the PARTIALS (.g01) files if the number of distinct atom components goes above 13 (= 91 atom pairs).

IMPORTANT NOTE ON SPACES: There is no obvious problem in EPSR with having a space in a filename or folder name. Windows requires the entire filename which has spaces in it to be enclosed in double quotes “” for it to be able to recognise it. These quotes must include the file path name if present. However the quotes are only needed for calls to the Windows operating system – they are not needed for the Fortran OPEN statement which opens files for reading - OPEN will not recognise the quotes and give an error message. Whether or not these quotes are included in system calls is determined by the value of “system_quotes” in system_commands.txt (see Section 3.1). As far as is known all the routines work correctly with spaces in filenames and folder names, but not every routine has necessarily been tested explicitly. If it is found that a routine does not work correctly, delete the “” in system_commands.txt and start EPSRshell again. However it will no longer be able to process files and folders with spaces in their names. Generally speaking it is probably best to play safe and use names without spaces – instead use underscore (_) or hyphens (-) to delineate words in filenames.

Note that to be sure that the editor defined by “system_edit” works correctly, the executable should appear somewhere in the system path.[3] If it does not the “ed” command in EPSR may not work correctly.

3.3 The Main Menu

This is the menu that EPSRshell comes up in when first started. It is also the menu from which the simulation is run, and from which most of the commands to other menus are executed. The main menu is different from the other menus in that it does not prompt for commands but is expecting commands to be typed without prompting. It is always possible to get a list of commands by typing “help” or “?”The main menu commands can be divided into two kinds: those that will normally be used as one-offs, and those that might be used repeatedly as part of an EPSR cycle. The “setup” command is used to setup the EPSR and auxiliary routine input files. Note that any command that is run from a script file must be given the required arguments, otherwise the script will stop and wait for input from the console at each cycle. See Section 3.4 for details about script operation.

A list of the “one-offs” includes:-

system <system command> -	invokes the specified command for the system in which the program is operating, e.g. “system cmd” would invoke the command prompt under Microsoft Windows. Note that this will run in the home folder, NOT in the current working folder, which may be different from the home folder.
pwd	shows the current working folder.
cd <folder name>	sets the working folder to that specified. Note that no checking is done to test whether this folder actually exists, so the program will carry on as if the folder does exist, even if it doesn’t! You will soon find out that it does not exist when none of the EPSR commands work properly. Typing “cd” on its own shows the current working folder. Typing “cd home” or “cd .” will set the current working folder to the home folder. To change from the present working folder to another working folder directly, use the command “cd .\<new working folder>”. It is important to use the “.\” here to signify that the new working folder is a subfolder of the current **home** folder. Leaving out the “.\” is O.K. but it means the program will expect to find the new working folder as a subfolder of the current **working** folder. Typing “cd \<new working folder>” will set the working folder to be in the top folder of the current drive. Currently it is not possible to access working folders in drives different from that of the home folder. Note: The bulk of this manual is written from the point of view of the Windows user. In Linux for example, the path separator character

3If, under Windows, that editor is “WORDPAD” it may need to be copied from its current location to somewhere in the system PATH. For example it could be placed in the same folder as the EPSRshell executable. See Section 9.3 for how to do this for Windows.

is / instead of \ for Windows, so this needs to be remembered for Linux users.

Type “cd ..” to move one folder up the current folder tree. Thus to get the top of the folder tree type “cd ..” several times.

ls <file list>	lists the specified files in the current working folder.
md <folder name>	creates a new folder
ed <filename>,edit <filename>	invokes the editor specified by the “system_edit” variable in “system_commands.txt”. Note that if Wordpad is the default editor and the specified filename does not exist, then Wordpad will not start correctly. If a file does not exist simply type “ed” or “edit” on its own.
del <filename>	deletes the specified file in the working folder
ss <filename>	starts the specified script file (see Section 3.4). If a previous script file was paused (rather than ended) then typing “**ss**” on its own will restart that script file.
ps	pauses the currently operating script. This will have to be done in another window running EPSRshell. To restart the script simply type “**ss**” in the original window.
es	ends the current operating script so that it can only be restarted by typing “ss <file name>.
plot	reads the current “plot_defaults.txt” file, and then enters the plot menu, allowing to plot specified data sets.
runjmol	runs the Jmol program so that a molecule can be constructed
readjmol	reads in a .jmol file and converts it to a .mol and .ato file and gives options for running MOPAC on the molecule
makeato	makes a .ato file using a series of values input from the console. This is useful for single atom molecules or molecules with only a few atoms. See Sections 4.1-4.2.
makemole	reads a **.mol** template file and creates the corresponding **.ato** file and **.atm** files. See Section 4.3
fmole	sets up the atomic coordinates according to a set of bonds defined by makeato or makemole. See Section
bonds	calculates all the pair separations within a molecule – if more than one molecule of a given type is present in the .ato file, the average values of all these interatomic distances and there standard deviations are shown.
mixato	takes the first molecule out of each specified .ato file and forms a mixture containing the specified number of molecules of each type. This routine can be used to make a **.ato** file bigger or smaller, but note that since it uses only one molecule (the first) from each file it will be necessary to run **randomise** then **fmole** after the mixture is set up to generate distinct molecules.

addato	similar to **mixato** but molecules are added to an existing **.ato** file. The molecules are inserted at random into the simulation box. If any of the molecules in that box are "tethered" (see later), then the program will check for overlaps based on the specified Lennard-Jones parameters before inserting the molecule – if overlaps occur, then another location is tried.
changeato	allows all the main parameters in the .ato file to be altered – it works like a standard EPSRshell menu. Also allows the box shape to be changed to non-cubic.
randomise	performs a random translation and rotation of every molecule in the box.
nwts	sets up the .wts file which gives the relative or absolute neutron weight factors of individual PSFs in the input neutron diffraction patterns.
xwts	sets up the .wts file which gives the relative or absolute x-ray weight of individual PSFs in the input x-ray diffraction patterns.
setup <program> <filename>	invokes the setup menu for the specified program and reads in the input file if specified. If no program is specified a list of the possible options is given, from which one must be chosen. If no file name is specified you will be prompted for one, or the program will search the working folder for input files relevant to the requested program. This menu allows you to modify values to be used in either EPSR itself or one of the auxiliary programs, including the plot2d and plot3d routines. Note that at present the working folder cannot be changed from within the setup menu, so it is necessary to set the working folder BEFORE entering setup. See Section 3.5 for more details about the setup menus.
plot2d, **plot3djmol**	runs the plot2d or plot3d plotting programs. The input files for these will have previously needed to be set up using the "setup" command.
plotato	does a simple plot of a .ato file, using either GNUplot (option 1), PGPLOT (option 2) or Jmol (option 3). This is useful for checking that a molecule has been set up correctly. For PGPLOT the result is in pgplot.gif or pgplot.ps, depending whether the system command system_pgout in "system_commands.txt" has been set to /gif or /cps. For the other routines, the plot is shown graphically on the screen.
makelattice	Takes the input from a unit cell file and creates a lattice of atoms with the specified number of unit cells along the **a, b** and **c** axes. This program cannot currently cope with molecules of more than one atom.
makelatticeato	Takes an existing **.ato** file and expands it along the **a, b** and **c** axes using the requested number of unit cells in each direction.

Commands that can be run either as one-offs or repeatedly in a script file include:-

epsr	runs the EPSR simulation from a specified filename.
partials	calculate the site-site RDFs. Note that these are calculated in EPSR and so there is normally no need to run a separate calculation of the RDFs.

triangles	calculates the distribution of included angles for triplets of atoms which satisfy the specified distance constraints.
torangles	calculates the internal rotation angle along a specified bond in specified molecule in the .ato file.
chains	calculates the distribution of chains among molecules and atoms which satisfy specific distance constraints, using the “shortest path” criterion to estimate the chain length between two atoms.
rings	calculates the distribution of rings among molecules and atoms which satisfy specific distance constraints, using the “shortest path” criterion.
clusters	calculates the distribution of cluster sizes involving molecules which are at specified distances
voids	calculates the distribution of voids and the void radial distribution function for spaces between atoms defined by a specified distance from any particular atom.
fluctuations	calculates the distribution of number fluctuations in a series of boxes of size defined by the user.
coord	calculates the average coordination number and coordination number distribution for specified atom pairs over a specified distance range.
writexyz	writes an xyz file for a specified .ato file. Result can be appended to an existing file or separate, sequentially numbered files can be written for each dump.
sharm	runs the spherical harmonic reconstruction program for molecules.
sdf	a version of **sharm** in which individual groups of atoms can be used to define molecules. Hence it is useful for looking at local order in network glasses.
<anything else>	The shell will attempt to run the program <anything else> in the binaries folder. Hence the user can run their own program at this point. If it does not exist an appropriate message is printed, and the program is returned to the shell. In addition from 2010 it is now possible to write your own routines to process the ato file and incorporate them into the setup menu.

3.4 Script operation

A typical EPSR script file, here called “runepsr.txt”, might look something like the following:-

```
cd .\water
epsr h2o298tot
rings h2o298tot
#epsr h2o298tot_large
#sharm h2o298tot_newshm
#epsr h2o298tot_new_large
cd .\mw73
epsr mw73
clusters newcls
```

The first line changes the working folder away from the home folder to that specified. The subsequent commands perform the specified operations. The # indicates a comment line which will be ignored by EPSRshell. The script file is invoked with the command “ss runepsr.txt”. This starts it running and the list of commands in the script file are executed in turn, with the exception of lines containing a #. When it has executed all the commands, the script file is rewound and the sequence of commands is repeated again. If you wish to prevent this you simply insert a “ps” or “es” as the last line of the script file. Note that the script file always runs in the home directory. Note also that you can only run one script file at a time in a given home folder. The script can of course access several working directories as it runs.

A script can have several “states”. The current state of a script for the present home folder is stored in a file called “runflag.txt” in the home folder. The allowed states of a script are “**interactive**”, “**running**”, “**pausing**” and “**notrunning**”. There may be more than one instance of EPSRshell running in a particular home folder, but only one of these instances can run the script.

A script status of “**interactive**” means none of these instances of EPSRshell is running a script at this time.

A script status of “**running**” means that one of these instances of EPSRshell is running a script and the “runflag.txt” file will say which script file is executing.

A script status of “**pausing**” signals to the instance of EPSRshell which is running the script that it should pause at the earliest opportunity. Note that it does this as soon as it has finished executing the current command – it does not wait until all the commands in the script are finished.

A script status of “**notrunning**” signals that the script shown in “runflag.txt” is still active but is not actually running at present. If EPSRshell is started in this home folder it will start the script running again automatically if this status is set. Note that it starts from the top of the script file when this happens – it does not carry on from where it left off when it was paused.

If a script is running in one instance of EPSRshell all other instances will run automatically in “**interactive**” mode. However any one of these other instances can pause or end the script at any time. Note that if a script is already paused then pausing it in another instance of EPSRshell will have no effect, but it can still be ended by typing “es”.

When a script is paused (script status is “**notrunning**”) it can be restarted by typing “ss” or else it will start automatically if a new instance of EPSRshell is started in the same home directory where the script file is stored.

It is possible to include a comment in the script file, provided it occurs on a line which has a # somewhere in it, or on the same line as a command provided there are at least 6 or more spaces between the end of the command and the beginning of the comment.

Sometimes it is necessary to run EPSR for one or more iterations, then exit to the system to perform a separate operation, such as run some other program on the current .ato file, then run it again on a repetitive basis. To do this you need to have an “e” or “bye” as the last line of the script file. See the following example, called “runepsrlw.txt”:-

```
cd .\aminoacids\MyJunk
epsr myjunk#
epsr pureh2o
e
```

In this case, when the script file is started with the usual command “ss runepsrlw.txt” it will run the script in the usual way, i.e. change the working folder to “aminoacids\MyJunk”, ignore the next line because it has a # in it, then run EPSR on the file pureh2o. When it gets to the “e” it will do two things. Firstly, because this version of EPSR is actually running the script, it will pause the script and put it into a state of “notrunning”. Secondly it will exit the script and put the shell into interactive mode. Thus when EPSRshell is restarted in this folder it will automatically resume this script until reaches the “e” again, when will again pause the script, and exit.

Beware of writing a script file that does nothing, then has an “e” in it at the end. This will start and stop very quickly and be difficult to get rid of. If this happens delete the “runflag.txt” file, which will automatically put EPSRshell into “**interactive**” mode when it restarts.

3.5 Setup menus

Any of the commands that can be run iteratively, plus the plotting commands **plot2d** and **plot3djmol**, require a number of variables to be setup. For EPSR itself the number is at least 43 if there is only one dataset, and more than this if there is more than one dataset. Creating and editing these input files is achieved via the “**setup**” command. To avoid the need of having to worry about so many values, almost all the variables are given sensible default values on startup, so that only the crucial ones need to be specified. These are usually shown as “<undefined>” if they cannot be set from the default values. These are most of the time filenames that need to be specified for the particular case in question. If a variable is “<undefined>” and you don’t know what to type, a simple solution is to type “search” for the value: this enters the search menu where you have the opportunity to either pick a file of the relevant extension from a list of those in the working folder or else type the filename yourself.

To enter **setup**, you type “setup <program> <input filename>”, where the two arguments are optional. If the program name is not given you will be shown a list of the options and asked to choose one. If the filename is not typed then you will automatically enter the search menu to find a suitable file of the appropriate extension to setup. If the filename you specify does not exist, then setup continues assuming that filename and using the default values for all variables that need to be defined. On exiting it will save these values, or any new values that have been specified within setup, to the new filename. If this involves overwriting an existing file you will be prompted whether you want to do this or not, and, if not, be allowed to specify a new filename. Note that when exiting from the EPSR setup the current atom coordinates and EP coefficients are written to the specified .ato and .pcof files respectively. Hence if these filenames have changed in the course of editing, new .ato and .pcof files will be generated, but there is no check here about overwriting an existing file, so be cautious!

At the end of the EPSR setup there is list of atom pairs and their minimum separations. These are calculated on the basis of the supplied Lennard-Jones values, using “roverlap” and “rminfac”. However they can be altered by typing in new values if it is felt the values for particular pairs are not appropriate. Note that this must be AFTER the initial simulation with “ireset” set to 1, otherwise the minimum distances will revert to their default values. If ireset is set to 2, then potential coefficients are all set to zero, but the specified minimum distances and any extra terms which have been defined are not affected.

Currently the list of programs which require setup to be run to either create or edit the input file is:

epsr
partials
clusters
coord
chains
rings
triangles
torangles
sharm
plot2d
plot3djmol
voids

Note that it is always possible to create the required input files directly by editing, as was done under the old EPSR regime, but this can be dangerous unless you know exactly what you are doing as it is easy to introduce errors in this way. In general unless an error is catastrophic the program will plough on and not make you aware that something could potentially be wrong (e.g. the reference potential has been truncated at too short a distance). Of course this can happen even when editing using **setup**, but at least the shell reduces the chance of it happening.

Note that it is almost never necessary to specify file extensions when typing files within **setup** as these will almost invariably be set by the appropriate program. This is in spite of some comments in the program to the contrary. The only real exception to this rule is when using the **system** command (outside of **setup**). In that case it will necessary to specify the full filename, including the folder path if the working folder is different from the home folder (and any quotes “” that are needed if the file or folder names have spaces in them.

3.6 The plot menu

The **plot** menu is really only a minor variation on the **setup** menu. Here however one of the variables, “p”, is actually a command: it directs the program to take the existing or specified plot type and plot the data with the parameters defined for that type. Also the “l” variable is also a command: it produces a list of the current plot types.

Plot types are defined by means of the file “plot_defaults.txt”. If this file does not exist in the working, then only a single plot type will be set up with arbitrary values set for the variables. A list of the variables that are needed for each plot type can be obtained by typing “help” or “?” in the usual way. You can increase the number of plot types by increasing the value of npt. This will generate extra plot types but give them only the default values, so that you will need to enter each new plot type in turn and change any of the values that need to be changed. Currently within any plot type you can plot up to 2 different file extensions. These have to have the same filename, but can have different extensions as specified by the “ext” variable. The only variable that cannot be changed from within the plot menu is the “type” variable, which gives each plot type a brief description to remind you what it plots. This is so that the text cannot be inadvertently changed by a typing mistake when paging through the menu. To change the “type” variable you need to first save the current plot types by typing “e” and then editing the appropriate line(s) in the “plot_defaults.txt” directly using the “ed” command in the Main menu. Here is an example of the plot_defaults.txt file to show you what it looks like:

```
plot_defaults          Title of this file
l                      Lists available plot types
f       hda06kbar            File name to plot
b       1 - 10           Block numbers to plot (e.g. 1 2 - 5 9 - 6)
p       16           Plot using the current or specified plot type
npt      21           Number of types of plot

pt   1

type       1 - EPSR S(Q) fit            Type of plot
ext       .EPSR.f01            File extension(s) used to search for plot files
bspace     2            Spacing between blocks in plot file
boffset    2            Column number(s) for first block(s) in plot file(s)
xmin       0            Minimum value on x-axis
xmax       20             Maximum value on x-axis
ymin       -3            Minimum value on x-axis
ymax       3            Maximum value on y-axis
ydel       3            Spacing between plots
yfact      1.0 1.0            Factor(s) outside y values
xf         0.75            Fractional x coordinate of labels
yf         0.25            Fractional y coordinate of labels
title      <undefined>            Title of plot
```

```
xlabel     Q [1/Å]              x-axis label
ylabel     S(Q)              y-axis label
logx       n              log scale x (yes or no)
logy       n              log scale y (yes or no)
xcolumn     1              column number for x values [normally 1 or 0]

pt   2

type       2 - EPSR S(Q) fit and difference              Type of plot
ext        .EPSR.f01 .EPSR.d01              File extension(s) used to search for plot files
bspace      2              Spacing between blocks in plot file
boffset     2 2            Column number(s) for first block(s) in plot file(s)
xmin        0              Minimum value on x-axis
xmax        20              Maximum value on x-axis
ymin        -3              Minimum value on x-axis
ymax        3              Maximum value on y-axis
ydel       3              Spacing between plots
yfact      1.0 1.0              Factor(s) outside y values
xf         0.75              Fractional x coordinate of labels
yf         0.25              Fractional y coordinate of labels
title      <undefined>              Title of plot
xlabel     Q [1/Å]              x-axis label
ylabel     S(Q)              y-axis label
logx       n              log scale x (yes or no)
logy       n              log scale y (yes or no)
xcolumn     1              column number for x values [normally 1 or 0]
```

And so on.

Note that when setting the block numbers to plot, the blocks can be specified in any order (including backwards) and they will be plotted in that order. If a hyphen is used between two numbers (always ensuring there are spaces between the hyphen and the numbers, otherwise the routine will think you mean minus numbers) then all the block numbers between the two bounding values will be plotted. If any specified block numbers are outside the range contained in the specified data file, they will be ignored. If none of the specified blocks exist in the file being plotted, a message to that effect is put out on the console and nothing happens.

If the title of the plot is "<undefined>" as above, then the plot program will generate a title consisting of the working folder and filename.

Typing "p" invokes a plot of the current plot type, which can be set by typing "pt n" where n is the number of the plot type. Or else typing "p n" will automatically set the plottype to "n" and then plot that plot type.

Obviously there is some capability to modify the plot from within the plot menu (e.g. x and y ranges, spacing between plots, etc.). However typing "p" actually creates a GNUplot (.gnu) script file in the working folder, and then starts GNUplot with this script file. The filename for this file is generated from the word "plot", the current plot type and the name of the file being plotted, with extension .gnu, e.g. "plot04h2o298tot.gnu". If the plotting options currently available from within the shell are not satisfactory it is always possible to edit this file, making use of the full range of the GNUplot capabilities, and then re-run it directly from GNUplot.

Running GNUplot from within the EPSRshell plot menu makes use of the piping operator < to force GNUplot to read the data from the ".gnu" file instead of from the console. Once it has plotted the data however you can always click on the shell window and alter various options, such as line styles, and so on. With the current version of GNUplot (version 4) you have the further option of zooming in on particular regions of the plot, using the right click mouse button. This facility can be switched off and on by typing "m" (for mouse). Note that the way the program currently is set up you have to close the GNUplot window(s) before you can continue working in

EPSRshell. This is to prevent EPSRshell generating a large number of GNUplot windows. If you want to review or compare particular plots then open GNUplot (directly from the binary, outside of EPSRshell), go to the folder where the .gnu files are stored, then type "call '<filename>.gnu'" in the GNUplot window. This will bring that particular plot back to view.

Note the variable "boffset" in the above examples. This controls which actual columns of data will be plotted. In the cases where there are two datasets being plotted, boffset can have two different values so that for example column 2 of dataset 1 is plotted against column 3 of dataset 2. This can be useful when plotting the running coordination number for example.

3.7 Plotting the box of atoms – **plotato**

As the molecules being generated become more complicated it is becoming increasingly important to ensure the molecular structures that have been generated by EPSR are correct or at least fulfil known structural constraints, such as bond distances, bond angles, dihedral angles and stereo symmetry. **plotato** is used to plot the box of atoms. Once the name of the .ato file to plot has been identified, the user has three options, whether to use GNUplot, PGPLOT or Jmol to plot the .ato file, or part of it.

The GNUplot option uses the same routines and methods used by **plot,** but instead invokes the GNUplot "splot" command which is an attempt to do surface plots within GNUplot. It makes no use of the surface plotting capability however, but simply draws circles of different sizes and colours to represent atoms and lines to represent bonds. The image can be easily rotated to assist with viewing, and there is even an option within this option to plot two boxes at slightly different viewing angles, so that a stereo view can be generated. The result is not perfect – for example the different atoms are plotted as different plots, so that the later atoms plotted always overlay the earlier ones, irrespective whether they are closer or further away from the observer. However it gives a quick and easy representation of the box of atoms – one can see very quickly whether particular bond constraints have been satisfied.

The PGPLOT option uses the sphere and cylinder methods within the PGXtal set of routines to draw atoms and bonds as specified. There is also the option to draw the sides of the box, which can be helpful in some cases.

The Jmol option invokes the Java routine Jmol.jar to plot the .ato file.

The GNUplot and Jmol programs produce a graphical output, while the PGPLOT program produces an output determined by the system_pgout variable set by system_commands.txt. Note that due to the limitations of the PGPLOT library that was used to compile the Windows version of the programs, only the /GIF, /PS, and /CPS options are available at present with PGPLOT.

plotato is invoked by typing "plotato <filename>" within EPSRshell, where the filename is the name of a **.ato** file. If <filename> is not specified then the program enters the search menu to find suitable **.ato** files in the current working directory.

Initially you are asked for a number between 1 and 3. 1 corresponds to the GNUplot method, 2 to PGPLOT, and 3 to the Jmol option.

Next you are asked for the molecule number to appear at the centre of the box. If a '0' is given then you get all the atoms in the box.

If you have specified a particular molecule to be at the centre of the box, you will be asked to give the dimensions of the box you want plotted. These are expressed as 4 widths, one for each of the x and y axes and then a front and back distance along the z-axis, always starting from the molecule at the centre of the plot. If you want to plot only the central molecule, then give zero for all these distances. The front and back distances along the z-axis allow you to plot a section through the simulation box.

The next three values list the Euler angles, phi, theta, chi, that you want to rotate the box by BEFORE selecting the slab defined in the previous paragraph. In principle this should allow you to choose a section of the box around a given molecule with that central molecule in a particular orientation. Operation of this option can tricky until you become familiar with the idea of thinking of 3-dimensional rotations in terms of Euler angles. Note that the box rotation is performed BEFORE the selection of x, y and z values defined by the box ranges.

plotato will then list the different atom types (see Section 4.1 below for a discussion of the distinction between atom 'classes' and 'types') found in the .ato file and ask you list the numbers of types you wish to avoid being plotted. Type '0' to get all the atoms in the box.

Once these are specified with the Jmol option set (option 3) the program writes a file to the working folder called <.atofilename>.xyz, which is then used as input to **Jmol**. **Jmol** is called with the command defined by the system_jmol variable from system_commands.txt. **Jmol** makes its own decisions about atom colours and bonds, but no doubt these can be changed once the Graphical User Interface (GUI) is displayed.

For the GNUplot and PGPLOT options **plotato** will initially try to open a file called 'gnuatoms.txt' in the home folder where it will attempt to read information on the size and colour of each atom class. Below is an example of 'gnuatoms.txt':-

```
 1.  0.0500000007  20.
O      8    2.00000    1    12  1.000  0.000  0.000
H      6    1.40000    8    15  1.000  1.000  1.000
N      8    1.00000    3    15  0.000  0.000  1.000
C      8    1.95000    8     8  0.400  0.400  0.400
S      8    1.40000    5    14  1.000  1.000  0.000
Cl     8    1.20000    2    10  0.000  1.000  0.000
Si     8    1.00000    5    14  1.000  1.000  0.000
```

If this file does not exist, or an atom class in your .ato file is not found in this file, you will be prompted for the values needed, and these will then be saved to the same file, so that you don't have to keep typing the values again every time you run the program. If you want to change the size or colour of an atom which already exists in "gnuatoms.txt", you will need to edit this file directly.

In the first line of this file the first of the three numbers controls the rounding on the lengths of the three Cartesian axes. Thus "1.0" signals to round the length of each axis up to the nearest Å. The second number controls the size of the offset of the *xy* plane of the plot from the bottom of the *z* axis – this needed to help avoid axis labels clashing, but it is not perfect depending upon the orientation of a particular plot. It is expressed as a fraction of the length of each of the three axes, which are set equal in length. The third number is an overall scaling factor on the size of all the plotted atoms – it is useful to help make the plot look realistic. The actual scale factor used depends on the length of the axes as well as the value of this number.

For each of the subsequent lines the sequence is the same:-

The first symbol gives the atomic symbol which defines the class of each atom, e.g. hydrogen, carbon, oxygen, etc. using standard chemical symbols for each element. Note that this is distinct from the atom types which can vary depending on the location of the atom within a molecule. Thus there can be several different atomic types, e.g. versions of carbon, all belonging to the same atom class (carbon in this case).

The second number gives the GNUplot symbol to plot. This will depend on which terminal window is being plotted in but for the Windows terminal, 8 should correspond to a solid circle, while 7 corresponds to a hollow circle. This number is ignored by PGPLOT.

The third number is the character size, represented as a fraction of the current character size. On actual plotting the actual character size plotted will be modified by the overall scale factor given on line 1. This number is ignored by PGPLOT.

The fourth number signals the character colour used by GNUplot. These can be set from the GNUplot graphical window by right clicking in the window (type "m" first to inhibit the automatic rotation or scaling option) and going to "line styles". When finished these need to be saved by repeating the right mouse click in the graphical window, then click on "update wgnuplot.ini". The above values correspond to line 1 = red, line 2 = green, line 3 = blue, line 4 = lighter grey, line 5 = yellow, and line 6 = dark grey, but obviously you can set your own colours if you wish to do so. This number is ignored by PGPLOT.

The fifth number was used to specify colour for ATOMS input files and is no longer needed.

The last three numbers are the red, green, blue colour indices used to define the colour of the atom sphere used by PGPLOT.

Having got the atoms sorted out, **plotato** will then ask you to specify bond lengths between specified pairs of atom classes. Again these will initially be read in from a file, 'gnubonds.txt' in the home folder. Below is an example of this file:

```
 1.00000   0.00000   0.00000   0.10000   0.10000   0.10000   0.01000
 C  C    1.00000   2.00000   0.10000
 C  N    1.00000   2.00000   0.10000
 C  O    1.00000   2.00000   0.10000
 O  H    0.50000   2.40000   0.10000
 C  H    0.50000   1.50000   0.10000
 N  O    1.00000   2.00000   0.10000
 N  H    0.50000   1.50000   0.10000
```

The first line of numbers is used only by PGPLOT. The first three numbers give the RGB values for the bond colour (red in the above instance), the next three give the RGB values for the lines used to show the box edges, and the last number gives the radius of the line used to show the box edges.

Hopefully the subsequent numbers are reasonably obvious: the first two letters signal the atom classes to be used to form the bond (the order is not important), the next two numbers represent the smallest and largest separations allowed for these two atom types to appear bonded, and the last number, used only by PGPLOT, defines the radius of the bond in Å. GNUplot simply draws a line to represent bonds.

You have the option of changing or adding to any of these bonds by typing in the relevant numbers. If you type "0 0 0 0 0" the program ends bond input and writes out the new gnubonds.txt with the new (or revised) bonds.

plotato then asks you for the elevation (x) and rotation (y) of the desired plot. Note that these values have different effects in GNUplot compared to PGPLOT, so you will need to experiment a few times to get the desired effect.

plotato then plots the box of atoms, and returns to the shell (after you have closed the GNUplot graph window for the GNUplot option).

Alternatively, if with the GNUplot option, instead of 0 0 0 0 0, you type "ste 0 0 0 0", it will ask you one more question, to specify the viewing direction for each plot in terms of spherical polar coordinates, and plot two boxes of atoms next to each other. If the viewing angles are close to one another, these can be viewed stereoscopically to give you a quasi-3D view of the box. Note that the best stereo image is obtained by making the θ values close to (but not equal to) 90°, and then making the φ values 3 – 5° apart, depending on how good your eyes are focussing.

plotato allows you to type the whole command in a single line up to the point where you specify which atom numbers to be excluded. This means if Jmol is selected then the entire command can be written at once. For example:-

plotato test 3 20 10 10 -3 3 0 0 0 0

will plot the atoms in ‘test.ato’ using Jmol with molecule number 20 at the centre of the plot. It will attempt to plot all these atoms, but limit the displayed atoms to those within 10Å of this molecule along the screen x and y axes (horizontal and vertical respectively), and ±3Å along the screen z axis (normal out of the screen). It will not rotate the display and no atoms within the ranges specified will be excluded.

You can type part of this command, but note that the plotting box dimensions (arguments 4 – 7) must be specified as a set of four numbers together, the Euler angles for rotating the central molecules must be specified as a set of three numbers together, and the list of excluded atoms will terminate at the last number input. If the number of arguments given is less than that required to complete the command, then the user will be prompted for the additional values. The extra terms required for GNUplot and PGplot will always be requested at the console.

4. Preparing for an EPSR simulation.

To run the EPSR simulation program 4 main input files are required, plus of course files containing the diffraction data to be fitted. These have extensions .**inp,** which basically contains the information about what data is to be fitted, and how it relates to the simulation, **.pcof** which primarily contains information on the coefficients of the empirical potential plus some details of the simulation which the user will not normally need to modify unless there is a specific reason to do so, **.wts**, which tells EPSR the weightings on each partial structure factor for each dataset, and **.ato**, which contains the atomic coordinates of all the atoms and molecules in the box, plus their Lennard-Jones and Coulomb charge (where appropriate) parameters. There is also a **.inpa** file generated to store intermediate values of various parameters, but this should not be modified by the user.

Since the **.ato** file is the most complicated of these files to set up, the next Sections describe how it can be generated for any particular system.

It is important to note here that EPSR makes a clear distinction between atoms and molecules. The EPSR **.ato** file consists of a sequence of molecules, each of which contains a sequence of atoms. A single atom on its own, not in a molecule, is represented within EPSR as a molecule containing only one atom, and having no bonds. Molecules of two or more atoms must have at least one bond between pairs of atoms, so that all the atoms in the molecule are bonded, either directly or indirectly. Additional constraints on the molecular geometry are supplied by means of bond angles, dihedral angles, and, if the atoms are not joined by any of these forces, it is possible to specify a Lennard-Jones potential which prevents intra-molecular overlaps. An atom *can* be left unbonded within a molecule, but if so it is likely drift far away from the parent molecule and be very difficult to identify in the **.ato** file.

4.1 Building a single atom **.ato** file

Simplest way to make the initial **.ato** file is to run **makeato**. By asking some hopefully fairly obvious questions, such as the type of atom, the Lennard-Jones parameters, the atomic mass and the atomic charge, this program will allow you to build a box containing 1 atom with all the necessary parameters set. Subsequently **mixato** can be used to make the box larger (with more atoms and molecules) or to make mixtures of atoms or molecules (this will be discussed further, together with the format of a .ato file in section 4.3).

4.2 Creating a molecule and building **.mol** and **.ato** files using Jmol – **runjmol**, **readjmol,** and **makemole**

Type **runjmol** to start Jmol from EPSRshell. Along the top bar you will find a button called “Open the model kit”. Press this – it will give you a short blue vertical bar on the top left hand side which itself expands into three menus – atoms, bonds and other operations. A model of methane should appear at the centre of the screen. The model kit menu will also appear if you right click the screen. Using the different menus you can add atoms to this molecule or change the atoms that already there, change the bonds and bond types, and so on. If the molecule is a bit complicated you may need to do this a few times.

When you are happy with the molecule, right click and select save file – save the molecule as a .jmol file (i.e. NOT the default .mol file). This is so that **readjmol** will recognise it.

The next task is to create the **.mol** and **.ato** files associated with this molecule – these are the actual files that will be used to store the molecule geometry and atomic coordinates for this molecule, respectively. To do this type **readjmol** in EPSRshell, and press enter until you see the .**jmol** file you created in the previous paragraph. Press y to accept this file. You will be given the opportunity to change the atom labels to something meaningful to your system if you so wish. Use option 3 in order to get a change atom label list in the **.mol** file – in this way all of the atoms will be numbered in the .**mol** file, and then each number will be assigned an atom label towards the end of the file. The atom label will also correspond to the Lennard Jones parameters and charges assigned at the end of the file.

You will also be given a chance to run MOPAC on this molecule. The name of the MOPAC executable is given in the system_commands.txt file which is required when EPSRshell is started: it is listed against the system_mopac variable. Usually a MOPAC_7.exe is distributed with EPSR in the \bin folder, and equivalent executables can generated for Linux and OSX systems. Whether or not MOPAC is run, .**mol** and .**ato** files should be produced when **readjmol** has finished. You can plot the final molecule .**ato** file using **plotato** to see what it looks like.

To change any of the specified bond length, angles or other parameters you need to edit the **.mol** file. Check you are happy with the bond distances and angles specified in the .**mol** file. Also, edit the atom labels if you want to use a different labelling scheme or change any rotational groups or change any Lennard-Jones parameters and atom charges to values which have been obtained from other sources (e.g. MD simulation, OPLS). Once you have made all the changes, save the file and exit the editor. Now run **makemole** on this **.mol** file: this will set all the new values in the corresponding **.ato** file. Note however that **makemole** will not actually *move* the atoms in the corresponding **.ato** file to their new positions if bond lengths, bond angles or dihedral angles are changed – this has to be done by running **fmole** a large number of times. See 4.4 below.

Repeat these steps until you have a .**mol** and **.ato** file for each of the molecules in your system.

The format of the .**mol** file sections are as follows:-

First line	.gmol 0. Do NOT edit this line!!
atom	atom list number, atom label (will be atom number if a change label section is used), x, y, z coordinates, no. of bonds for this atom, and the atom numbers to which it makes *direct* bonds (i.e. not via bond angles or dihedral angles).
bond	atom type 1, atom type 2, expected bond distance
extra	optional command that can be added to include extra bond distances between specified atom numbers. This feature can be used to stabilise a molecule when the near neighbour bond distances and angles are not sufficient by themselves.
angle	atom label 1, atom label 2, atom label 3, angle. These define a bond angle group
dihedral	atom number 1, atom number 2, atom number 3, atom number 4, dihedral angle. Atoms 2 and 3 will form the axis of rotation. Atoms 1 and 4 are used to calculate and define the dihedral angle. Note that actual atom numbers are needed here, not atom labels as for the bond and bond angle definitions. It may be necessary to plot the molecule with **plotato** (Jmol option) to see the correct atom numbers to specify here.
rot	defines a rotational head group or side group. The two atom numbers specify the axis of rotation. The order of the two atoms is not important – **makemole** will make a list of all the atoms directly or indirectly bonded to each of these atoms and choose the rotational group which has the smaller number of atoms. This group will be bonded to the second atom of the pair, so the order of the pair may be reversed in the final output.

qradius	optional command that sets the charge radius of a particular atom type. Format is “qradius <atom label> <charge radius for this atom type>
changelabel	atom number, new atom label
potential	atom label, epsilon (KJ/mole), sigma, atomic weight (2 for hydrogen), charge, atom type
temperature	temperature of data collection
vibtemp	weights the bond length constraints (default 65, increase the number for more rigid, decrease the number for more floppy molecules[do not use 0]))
angtemp	weights the bond angle constraints relative to the bond length weighting (default 1, increase the number for more rigid, decrease the number for more floppy bond angles [do not use 0]).
dihtemp	EPSR24 calculates the dihedral angles in the simulation and compares these to the dihedral angle stated for a set of atoms in the .mol file and then tries the minimise the difference between these two values. **dihtemp** is the parameter for how close these two values need to be (default 10, increase this number for more rigid dihedral angles, decrease the number for more floppy dihedral angles [do not use 0]))
ecoredcore	defines Lennard-Jones well depth and core radius to be used to prevent intra-molecular atomic overlaps. Defaults to “0 0” if not specified.
density	not carried forward to box so not required

NOTE: Other formats for the first part of the **.mol** file can be used. These stem from earlier versions of EPSR and can still be used if necessary. See earlier versions of this manual to see how these are defined.

4.3 Making the simulation box – **mixato** and **addato.**

Once all the ato files have been created for each of the components in the system, they need to be placed in a box with the appropriate ratios of each component and the correct atomic number density. You can do this with **mixato**. This basically starts with your single atom or molecule files, and asks you how many **.ato** files you wish to mix and their filenames. It also asks you how many of each molecule type you wish to use, and the final atomic number density required. There is nothing to stop you specifying only one .**ato** file in **mixato**, so you can use this to generate a box containing as many molecules of a single type. If any of the **.ato** files specified contains more than one type of molecule, then an error message will be printed and the program will stop. In addition if there is more than one molecule in a specified **.ato** file **mixato** will only use the FIRST molecule in each **.ato** file and produce multiple carbon copies of that one molecule. Note that the order that each of the atoms/molecules are input into **mixato** will determine their ordering in the final .ato file.

The basic structure of the **.ato** file is as follows (cubic simulation box):-

Line 1: (free format) Number of molecules, box dimension (Å), temperature (K)

If the simulation box is not cubic, then this first line is replaced by 3 lines:-

Line 1a: (free format) Number of molecules, temperature (K)

Line 1b: (free format) Crystallographic **a**, **b**, and **c** axis lengths

Line 1c: (free format) Spherical polar angles φ_b (=angle **b** axis makes with **a**), θ_c, φ_c (spherical polar angles of **c** axis with respect to **a** assuming **a** lies along the Cartesian *x*-axis.) Note that these angles are NOT the same as the standard (α, β, γ) values used in most crystallography treatises.

Line 2: (free format) tol (used to control the displacement of tethered molecules), step size for intramolecular translations, step size for rotational and dihedral group rotations, step size for whole molecule rotations, step size for whole molecule translations, and vibrational weighting (i.e. coefficient *C*/2 in equation (2.2.1), typically *C*/2=65.), angtemp (1.0), dihtemp (10.0) plus two other values which are set by the program – changing these will have no effect. Setting any of the step sizes to zero will inhibit trial moves of that type. Note that the step sizes here are only notional since a separate record of suitable step sizes for each molecule type is kept at the end of the **.ato** file.

Line 3: (free format) Number of atoms in the first molecule, *x,y,z* coordinates of the centre of mass of this molecule, phix,phiy,phiz coordinates (used to specify origin if molecule is tethered, otherwise set to 0), “F” or “T” to specify whether this molecule is “free” or “tethered”, a number which counts the number of times an attempted move with this molecule is unsuccessful (set back to zero when a molecular move is accepted – this is occasionally needed to see if the simulation or a particular molecule has got “stuck”, i.e. only very tiny moves are being accepted, if at all) and the molecule number. These last three values are not required on input, in which case default values are used, so that .ato files generated under previous versions of EPSR will still run correctly.

Then for each atom in the molecule:-

Line 4: (format(1x,A3) (up to)three character label for this atom, preceded by a space, and followed by a number which signals the relative number of this atom within the molecule. As with the molecule number this number is not read on input, but it is helpful when checking whether the bonds have been setup correctly.

Line 5: (free format) *x,y,z* coordinates of this atom relative to the centre of mass.

Line 6: (free format) number of *other* atoms in the same molecule this atom is bound to (with a harmonic potential such as (2.2.1)), followed by their atom numbers and the corresponding bond length given in pairs. If the number of bonds is zero (unbonded atom) then this line should start with a 0 and contain nothing else. Normally this would only occur for a molecule containing only 1 atom. If an atom is left unbonded to any other atoms in the molecule it will drift around all over the simulation box, unrelated to its parent molecule. Bond angle bonds and non-direct bonds are signified by a negative value for the bond length.

Lines 4,5 and 6 are then repeated for each subsequent atom in the molecule.

Line 7: (free format) number of rotational groups in this molecule – if zero then you need simply a 0. This number should include any dihedral groups.

Line 8: (format(1x,a3) – only present if line 7 is not zero) 1 space and the word (in upper case) ROT or DIH. This is to specify an intramolecular headgroup or side chain rotational or dihedral group. Currently no other types of moves are recognized.

Line 9: (free format) Two atom numbers to be used to form the axis about which the headgroup will be rotated.

If the rotational group is a dihedral group there follows the number of dihedral angles in the group on one line, and then on subsequent lines the two atom numbers that will be used to calculate the dihedral angle, and the expected dihedral angle for these atoms subtend with using the two atom numbers specified in line 9. The first atom of this pair is expected to be bonded to the first atom of line 9, while the second atom of the pair is bonded to the second atom of line 9.

Line 10: (free format) The number of atoms in the headgroup to be rotated and their atom numbers. Obviously this list must not contain either of the atoms used to define the axis of rotation, otherwise the program may give unpredictable results.

Lines 8, 9 and 10 are then repeated for each subsequent rotational or dihedral group in this molecule.

This completes the input for the first molecule. Lines 3 – 10 are then repeated for each subsequent molecule in the box. It does not matter in what order they are entered although it is conventional to group all molecules or atoms of the same type together.

At the end of the molecule input, the Lennard-Jones and atomic mass and Coulomb charge parameters are specified. In reading the .ATO file the EPSR program will have used the atom labels to define a set of atomic “types”, 1 type for each different atomic label. Thus M and H might both refer to hydrogen atoms, but the reference potential parameters will still need to be defined separately for each type. The program will be expecting reference potential parameters for each atomic type – if one or more is missing it will print out an error message and may stop or crash later on.

Thus for each atom type there are two lines required.

Line 11: (format (2(1x,A3), 1x,I)) The atom label – this must appear exactly as it appeared in Line 4 when specifying the atom within the molecule, i.e. it is case sensitive. This is followed by the atomic symbol for this atom, exactly as it appears in the Periodic Table, and an integer, iexchange (0,1) which determines whether this atom exchanges with other atoms in the box. iexchange is automatically set to 0 if the atomic symbol is not H and will be checked and used by the **nwts** program. If either of the latter two values have not be set (e.g. from the earlier versions of the software) they will be automatically set to XXX and -1 respectively when read into EPSR to signal that they may need to be set.

Line 12: (free format) Lennard-Jones well depth, ε, (kJ/mol), core diameter, σ (Å), atomic mass (amu., note that hydrogen atoms are always given a mass of 2 amu, irrespective of whether they are isotopically substituted or not), Coulomb charge (e), and a charge radius (which should be set to 0 unless you intend to dock molecules – see Section 4.8).

Lines 11 and 12 are then repeated for each atom type present in the .ato file.

Line 13: (free format) Intra-molecular Lennard-Jones parameters. If non-zero these values are used to keep non-bonded atoms in the same molecule apart.

Line 14: (free format) Does not need to be given, but after a simulation will contain a series of integers which are used by the random number generator RAN1() so that the random number sequence starts from the place where it left off at the end of the previous simulation.

Line 15: (free format) There follows a list of the molecule types that have been found in this file, and, if they were generated from a template file, the name of the corresponding **.mol** file. This is so that if the corresponding **.mol** file is changed in some way the new bond lengths or rotational groups will be incorporated into this **.ato** file when **fmole** is run. Also on this line are the four step sizes for this molecule types, one for intramolecular vibrations, one for intramolecular rotations, one for whole molecule rotations and one for whole molecule translations. This is to ensure that large molecule moves are accepted with the same frequency as small molecule moves, even though the size of the move may be smaller in the former case. These values override those found at the beginning of the **.ato** file, unless the beginning values are 0, in which the latter values are applied. Hence the beginning values can be used to control whether particular molecule moves are performed, such as whole molecule rotations, for example.

IMPORTANT NOTE: When the **.ato** file is read each molecule is given a type number, as listed on line 15 above. This molecule type is determined from the atom type number of the first atom in each molecule. The atom type is in turn determined from the atom label that was set in the **.mol** file. Thus in mixtures of molecules it is important to ensure the first atom of each molecule has a distinct atom label, otherwise the molecule will not be recognized as a different type and an error will occur. For example suppose two distinct molecules both begin with a carbon atom. It is important that this carbon atom has a distinct label for each molecule, even if other atoms in the molecules have the same labels.

Finally at the end of the **.ato** file is the atomic number density of this simulation box and a list of the atom types, their atomic symbols, and their atomic fractions. These are not read by EPSR but can be useful for setting up the Gudrun input file.

addato works in a very similar way to **mixato.** The difference is that **addato** ADDS molecules to an existing **.ato** file, placing them at random in the output simulation box. However if any of the molecules in the existing box are “tethered”, then if the new molecule would overlap with the tethered molecle, as defined by the specified Lennard-Jones parameters, a new insertion point is sought. Hence if you wanted to add molecules to an existing simulation, but wished to avoid overlaps, you would first tether all the molecules in the simulation box, add the required number of new molecules, then set the previously tethered molecules free. This can be done with **changeato.** With **addato**, only a finite number of insertions are attempted for each added molecule so that if the system is already very dense it may not insert fully the requested number of molecules.

4.4 Randomising and modifying the **.ato** file – **randomise, fmole,** and **changeato**.

After performing **mixato**, all the molecules/atoms will be positioned on top of each other in the centre of the box. In order to distribute the molecule positions and orientations randomly throughout the box it is necessary to perform **randomise**. This randomises the positions and orientations of all the molecules, and randomises the orientation of any rotational groups. This is a vital step if you do not want to spend a huge amount of time bringing the box to equilibrium, and you want to ensure you have started from a truly random distribution. Note that if the internal rotation step is set to zero in the **.ato** file then random rotation of internal groups will not be performed, but there is no way to stop the randomisation of whole molecule rotations and positions. After **randomise** you need to run **fmole** for ~10000 cycles to disorder the molecules. IMPORTANT NOTE: After **mixato** you must run **randomise** BEFORE running **fmole**, otherwise the program will find all the molecules on top of each other and so likely overflow the neighbour list.

To change the molecule parameters for a **.ato** file, edit the appropriate **.mol** template files and then run **fmole** with 0 iterations to insert the new values in the **.ato** file. To change other parameters in the **.ato** file you can run **changeato** with the argument the filename of the **.ato** file that needs to be changed. The program **changeato** will allow you to modify some parameters, like density, temperature, potential parameters, simulation box shape, molecular tethering. It is also in principle possible to change bond lengths, and add and subtract bonds using **changeato**, but this is strongly discouraged compared to editing the appropriate **.mol** files and then running **fmole.** If bond lengths are changed or the vibrational, bond angle or dihedral angle temperatures are changed, you will need to run **fmole** a large number of times to get the atoms to actually move to their new bond lengths.

changeato also has a command to change the box shape from cubic. Programs like **makeato** and **makemole** generate a cubic box, but you can change this with the **boxshape** command within **changeato**.

4.5 Building long chain molecules – **dockato.**

The methods outlined so far will build boxes of atoms and molecules that are reasonably complex but well contained in the sense that individual molecules do not have a large spatial extent. To use these methods to build a long chain molecule or disordered network structure would be very inefficient and might not lead to the correct structure being produced. **dockato** is a first attempt to circumvent this problem by allowing you to dock two molecules together to form a new, single and enlarged molecule. This process can go on in principle indefinitely – until you run out of array space in practice. At the present time the **dockato** program is not particularly efficient so is not really suitable for building very large structures. But it can build something like a peptide chain in a matter of a few seconds, so it is potentially very useful.

Note (as of 2012-07-09): In the following text it refers to using “Q” atoms to perform the docking. This is not strictly necessary. Any pair of atoms can form docking sites in fact. However it is a good idea to use “Q” atoms to make sure the docking sites are indeed the ones you want them to be (e.g. they will be the ones that disappear on docking), since **dockato** will dock molecules at any suitable sites that fulfil the bonding criteria. Hence unless the labelling is done carefully you could end up having unwanted bonds being made.

To understand **dockato** you need to know something about so-called “Q” atoms in EPSR. Q-atoms were originally incorporated into EPSR to emulate charged sites on molecules which otherwise did not have any scattering properties. They are denoted by the first character of their atomic type, which has to be either “Q” or “q”. Hence atomic labels for real atoms in the system should not begin with either of these characters. They are not real atoms in the sense that although they may have mass (for defining the force constants between them and other atoms in the molecule), they do not contribute to the mass of the molecule, nor to the number density of the system in question. They can have potential parameters, like Lennard-Jones values and Coulomb charges, and so will contribute to the energy of the simulation, but they do not contribute to the scattering pattern, and so can have no neutron or x-ray weights associated with them. However they can and do have radial distribution functions associated with them which are listed alongside all the other RDFs from the simulation. Because they cannot contribute to the scattering pattern there can be no empirical potential associated with them.

The fact that Q-atoms are not real means that they can disappear without affecting the density of the simulation box (provided that in doing so does not create a charge imbalance). Hence the idea behind **dockato** is that wherever there is a docking site on a molecule it will have attached to it 1 or more Q-atoms. Then when two molecules are docked, 1 or more of these Q-atoms are removed when they overlap the atom onto which they have been docked. The two molecules are then merged to form a new single and enlarged molecule, with any overlapping sites removed.

To understand how this works consider a simple example – a chain of selenium atoms. The starting point is a **.mol** file called ‘se_base.mol’:

```
 5
|   |1   |2   |3   |4   |5   |6   |7   |8   |9   |10  |11  |12  |13  |14  |15
  1                    Se   *   2QSe
bond Se  QSe    1.50000
angle QSe Se  QSe  120.00000
potential Se   0.65000E+00  0.14000E+01  0.60000E+02 -0.20000E+01 Se
potential QSe  0.00000E+00  0.00000E+00  0.16000E+02  0.10000E+01 O
qradius Se   0.20000E+00
qradius QSe  0.20000E+00
temperature  0.300000E+03
density  0.100000E-01
ecoredcore    1.00000    3.00000
```

This (old **.mol** file format) defines a selenium atom with two ‘QSe’ atoms attached, forming an internal angle of 120°. Note that for **dockato** it is a good idea to set a finite **qradius** – this will be used by **dockato** to determine overlaps. If the charge radius is zero, the program may not recognise two atoms as overlapping when they are very close in practice.

This produces molecules which look like this:-

1

3

4

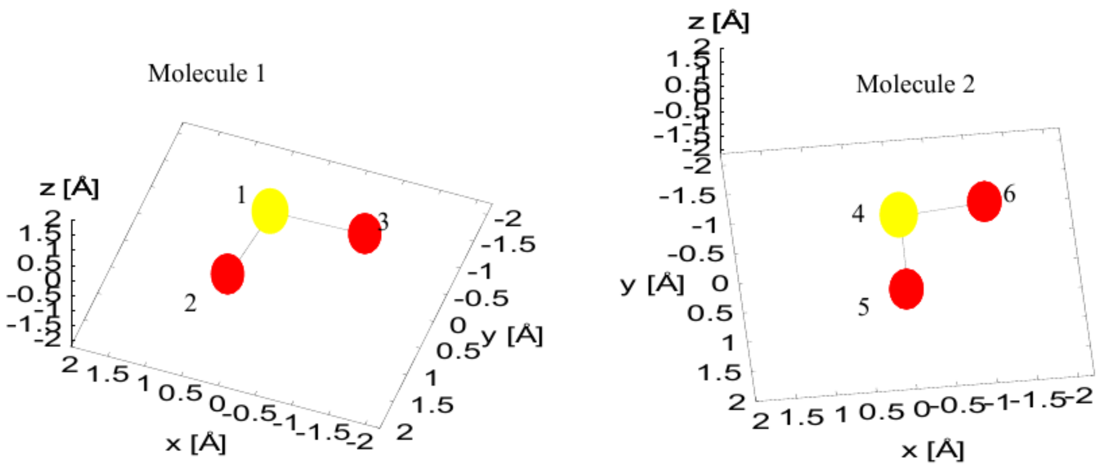


Figure 4.2

The yellow atoms are Se (atoms 1 and 4) and the red atoms are QSe (atoms 2, 3, 5 and 6). The object of the exercise is to bring atom 5 on top of atom 1 and atom 2 on top of atom 4. This involves a **translation** of molecule 2 so that atom 5 overlaps atom 1, followed by a rotation of molecule 2 about the overlapped atoms 1 and 5 so that atom 4 overlaps atom 2. At this point we have formed a new bond Se – Se between atoms 1 and 4, so that atoms 2 and 5 are no longer needed. This creates a new molecule with only 4 atoms in total, with atoms 2 and 5 removed:-

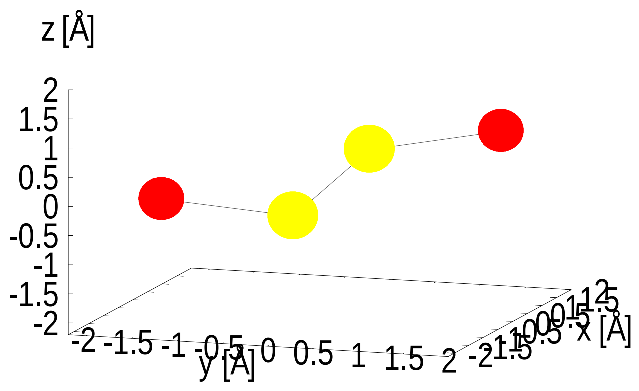


Figure 4.3

The docking is not quite finished however since there is a residual quantity undefined – the dihedral angle defined by atoms 3 and 6 about the bond between atoms 1 and 4. Note that atoms 3 and 6 are both QSe atoms. In this example the dihedral angle was set to a random value between 0 and 360° but it could have been specified more precisely. **dockato** has the ability to set this angle precisely or generate some random rotation over a specified range of angles, or to leave the angle undefined, so that a rotational 'ROT' group is generated about this bond.

If we now want to dock a third molecule onto this chain, we can do so in exactly the same way, e.g. atom 8 on the new molecule overlaps atom 4 on the old, while atom 6 on the old molecule overlaps atom 7 on the new, Fig. 4.4.

There is one subtlety that needs to be taken care of however. When we dock the third molecule we will again have to define the dihedral angle, but in this case the dihedral angle will be defined about the bond between atoms 4 and 7 by the atoms 8 and 1. These are QSe and Se atoms respectively, unlike the first docking where both dihedral atoms were QSe atoms. For any subsequent dockings the atoms needed to define the dihedral angle will always be QSe and Se.

dockato uses atom types to determine docking sites. Hence when setting up the docking molecule files it is worth bearing this in mind so that there is no possibility of confusion over which sites are to be docked.

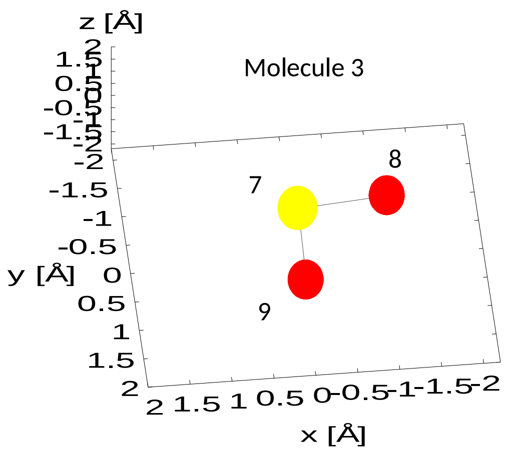


Figure 4.4

In the example given here the relevant command for the first docking would be:

dockato se_add QSe Se QSe 1 10 se_add Se QSe QSe 0 360 nooverlaps se_add_2

Here the name of the **.ato** file to be used for both **docking** and to be **docked** on to is ‘se_add.ato’

The first file name is the **docking** molecule, while the second is the **docked** molecule. We will overlap atom QSe on the **docking** molecule with atom Se on the **docked** molecule, at the same time rotating the **docking** molecule so that atom Se on the **docking** molecule overlaps atom QSe on the **docked** molecule. Having done that we choose a random dihedral rotation between 0 and 360° about the new Se – Se bond, with the dihedral angle defined by atoms QSe on both the **docking** and **docked** molecule. The output is to the file ‘se_add_2.ato’.

This defines all but 2 of the arguments to the command **dockato**. These are arguments 5 and 6. Argument 5 is the number of dockings that are to take place. In this case just 1. Argument 6 is the maximum distance these docked molecules can proceed from the starting molecule.

To summarise the arguments to the **dockato** command:

Argument 1:	Name of the **docking** molecule **.ato** file.
Argument 2:	Atom type in the **docking** molecule to be used for translating this molecule on to the **docked** molecule. This atom will disappear after docking operation.
Argument 3:	Atom type in the **docking** molecule to be used for rotating this molecule on to the **docked** molecule. This atom will be retained after the docking operation and the corresponding atom on the docked molecule will disappear
Argument 4	Atom type in the **docking** molecule to be used for defining the dihedral rotation of this molecule about the new bond. If this type is specified as ‘0’ a ‘ROT’ group will be generated about this bond, and no dihedral angle will be specified.
Argument 5	Number of dockings to be attempted. If any are unsuccessful a message saying this will be printed at the end.
Argument 6	Maximum distance in Å to which docking is to proceed.
Argument 7	Name of the **docked** molecule **.ato** file.
Argument 8:	Atom type on the **docked** molecule to be used for translating the **docking** molecule on to this molecule. This atom will be retained after the docking operation

Argument 9:	Atom type in the **docked** molecule to be used for rotating the **docking** molecule on to this molecule. This atom will be removed after the docking operation.
Argument 10	Atom type in the **docked** molecule to be used in combination with Argument 4 for defining the dihedral rotation of the **docking** molecule about the new bond. If either or both of these types is specified as ‘0’ a ‘ROT’ group will be generated about this bond, and no dihedral angle will be specified.
Argument 11	Smallest dihedral angle allowed.
Argument 12	Largest dihedral angle allowed. The actual angle chosen will be randomly between these two limits. Arguments 11 and 12 will be ignored if either or both of arguments 4 and 10 are ‘0’.
Argument 13	Overlap factor. Controls whether **dockato** will check if any atoms of the docking and docked molecules overlap (after docking is completed), based on the stated Lennard-Jones parameters. If set to ‘nooverlaps’ **dockato** will check for overlaps and, if atomic overlap is found (as defined by $\sigma_{\alpha\beta}$ in equation (2.2.4)), abandon that particular docking and attempt another one. Up to 10 docking attempts are tried. If this argument is set to ‘yes’ this means atomic overlaps are allowed and **dockato** will NOT check for overlaps. If set to any other character string, then this is treated the same as ‘yes’, i.e. overlaps are allowed. Alternatively a factor can be specified, in the range 0 - 1: this will be used to modify the minimum allowed distance between atoms, as set by the Lennard-Jones parameters. Thus a factor of 0.0 is the same as “yes”, while a factor of 1.0 is the same as “nooverlaps”. The total number of docking attempts is limited to prevent the docking sequence lasting indefinitely in the event that every docking attempt fails because of overlaps. In such cases you may not get the number of molecules docked that you asked for.
Argument 14	Name of **.ato** file in which to save the new molecule.

If defined the dihedral angle is constrained by a new dihedral group involving the two atoms specified by arguments 4 and 10.

The **dockato** command has probably not been as fully tested as possible, but it seems to work for building polymer and peptide chains. Here is the command that would be used if we added 10 ‘se_add.ato’ molecules to our new molecule ‘se_add_2.ato’, using a dihedral angle of 120° throughout:-

```
dockato se_add QSe Se QSe 10 20 se_add_2 Se QSe Se 120 120 nooverlaps se_add_12
```

And this is the result, ‘se_add_12.ato’:

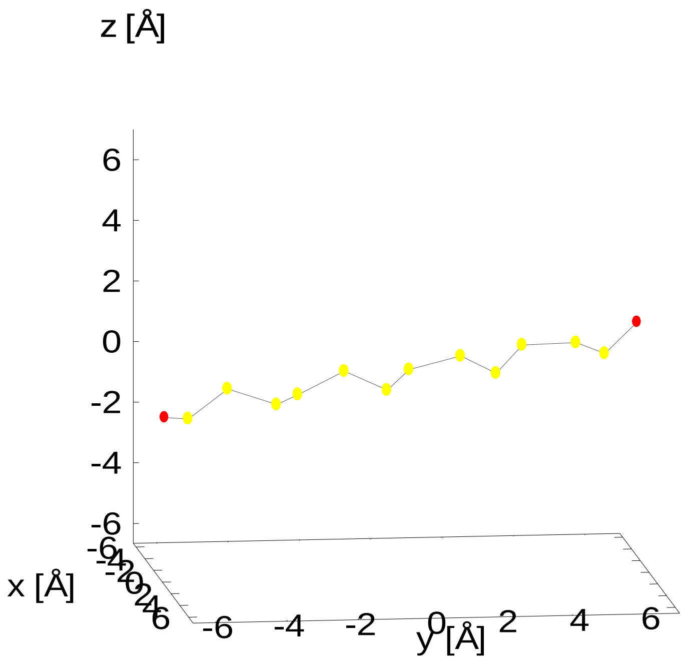


Figure 4.5

If instead we had used a dihedral angle of 20° throughout, we would have obtained the following result:

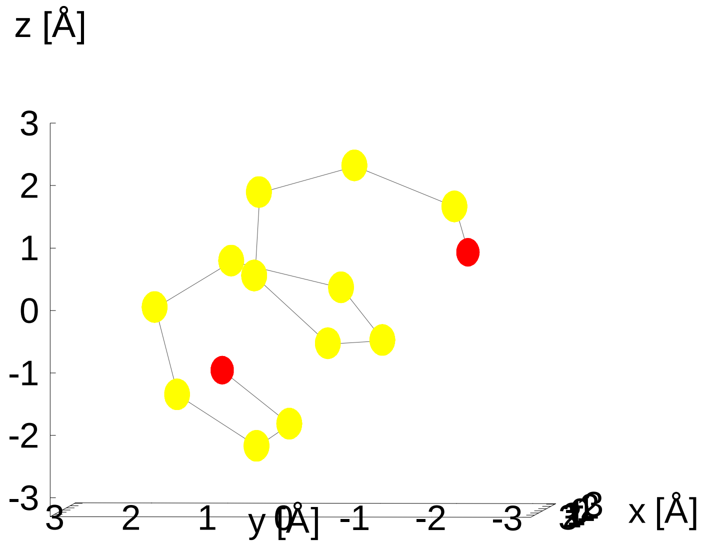


Figure 4.6

i.e. a helix.

If we had set the dihedral angle to zero, then we would have obtained simply a hexagon containing 6 Se atoms, since the chain would have wrapped around on itself after 5 dockings, and any further docking would have been impossible.

Although this is a very simple case, the principle is the same for more complex dockings.

When the molecules are merged to form the new molecule the atoms defined by arguments 2 and 9 of the **dockato** command are removed from the system. Hence it is important these are defined as ‘Q-atoms’ if you do not want real atoms to be removed in the docking process. In addition if as a result of the dihedral rotation the atom on the **docking** molecule defined by argument 4 overlaps any atom on the **docked** molecule, it too is removed in the merging process. Hence one should try to ensure this atom is also a ‘Q-atom’ if at all possible.

There is one other feature the user will need to be aware of. Once the docking has taken place, but before the **docking** and **docked** molecules are actually merged, a test is done, if requested as above, to see whether any of the other atoms on the two molecules are also overlapping. If any of these other atoms are overlapping, as determined by the mean value of the Lennard-Jones σ parameter, and they are not bonded to either of the atoms involved in the docking, namely atoms defined by arguments 3 and 8 of the **dockato** command, then the docking is abandoned. If the range of dihedral angles allows it, another dihedral angle will be chosen, up to 10 attempts before the docking is abandoned completely.

The same program can be used to produce networks of atoms, by having atoms with 3 or more ligands, but it currently is not very efficient and can take a while to form a network with say 50 atoms. In its current form **dockato** is best for producing chains of molecules, where there are relatively few options for docking sites.

Note that any template files that may be in the original **docking** molecule **.ato** file will be of no use in the **docked** molecule file, so the template for this **docked** molecule is set to the name of the final docked molecule file, i.e. the docked molecule becomes its own new template file.

A few hints to successful docking.

1) Any rotational groups in the original molecules are transferred and updated if docking is successful. Hence if you do not want to allow these rotations, for example in **randomise**, then you should delete them from the original **.ato** files before performing the docking. If rotational groups are defined and the internal rotation step is non-zero, then **randomise** will perform random rotations for each rotational group and could generate highly contorted structures. Therefore remove all **rot** groups from the parent molecules prior to docking, or set the internal rotational step to zero prior to running **randomise**. Sometimes it is desirable to leave such groups in, e.g. for the backbone of a flexible polymer, in which case the best option is to use **changeato** on the docked molecule **.ato** to set the internal rotation step to zero prior to running **randomise**. However remember to unset to a finite value (e.g.1.0) prior to running **epsr** or **fmole** otherwise the chain will remain rigid.

2) The docking program determines whether two atoms can be overlapped, i.e. docked, by measuring atom separations. Hence it is a good idea to run **fmole** on the molecules to be docked with a high vibrational temperature, of order 1.0E5. (This can be set with the parameter ‘**vibtemp**’ in **changeato**.) This should produce bond distances which are close to those specified. Once docking is complete, the vibrational temperature can be relaxed to its normal value (typically ~65) then run **fmole** again on the final file to relax the atoms again. Also using **qradius** > 0.0 will help **dockato** to find suitable docking sites.

3) If there are equivalent docking positions at each end of a molecule this can occasionally produce unexpected results because either end will be used for docking. Hence it is a good idea in such cases to start the docking sequence with a molecule which has a different termination to the docking molecule so that docking proceeds from only one end of the molecule.

4) If a molecule proceeds across the boundaries of the simulation box it is possible to create an “image” atoms so that the end of the molecule “sees” its periodic image. This can be achieved in **changeato.** However to implement this option you will need to be quite careful that the alignment of the molecule and the simulation box size are carefully defined to ensure these image atoms are indeed found in the correct position as images, otherwise you may end up distorting the molecule excessively when running **fmole** and **EPSR.**

5) If multiple dockings fail because of atomic overlaps, try reducing the overlap factor (Argument 13 above) below 1.0, to allow more overlaps.

6) It is strongly recommended you save the dockato commands in a text file for future reference where you can edit them if necessary. To run the command, simply copy it from the text file, and paste it into the EPSRshell window.

4.6 Building periodic structures – **makelattice** and **makelatticeato.**

These programs are designed to allow you to build a crystalline structure (**makelattice**) then reproduce that structure a specified number of times along each of the crystallographic **a**, **b**, and **c** axes. In both cases the output is in the form of a standard EPSR **.ato** file. **makelattice** requires an input file, which might have the extension .**unit**, but this is not mandatory as no particular extension needs to be specified. Here is an example of this file to create a sheet of graphene using a tetragonal unit cell:-

4.26 2.46 100	! This gives the lengths of the **a**, **b** and **c** crystallographic axes
90 90 90	! This gives the crystallographic (αβγ) angles between axes.
fractional	! This says subsequent coordinates are fractional. Could be “absolute”
4	! No. of atoms in unit cell
Cg 0.166667 0.5 0.0	! Atom label and fractional coordinates along (**a**,**b**,**c**)
Cg 0.333333 0 0	! Ditto
Cg 0.666667 0 0	! Ditto
Cg 0.833333 0.5 0	! Ditto
Cg C 0	! Atom label, atom symbol, and iexchange (0 normally)
0.8 3.7 12 0 0	! L-J parameters, mass, charge and charge radius
0.0 0.0	! **ecore** and **dcore**

The program is run from EPSRshell, and the user simply has to type the name of the unit cell file (including its extension). They are also asked for the number of unit cells they want along the (**a**,**b**,**c**) axes. This creates a .**ato** file which can be plotted with **plotato.** If absolute coordinates are specified then the coordinates are given as Cartesian (*xyz*) coordinates in units of Å. Note that with **makelattice** only single atom molecules can be specified currently. The atoms inserted into this lattice are automatically given “tethered” status – you can switch this off using **changeato**.

makelatticeato is even simpler than **makelattice**. The user is simply asked the name of the **.ato** file to expand, and the number of unit cells along the (**a**,**b**,**c**) axes. You are also given the option to convert all molecules to single atom molecules, but generally this step is not advisable.

4.7 Editing the **.ato** file – **changeato.**

In principle the **.ato** file can be edited directly with a suitable editor, but this will be hard work, particularly if the density or the box shape is to be changed, image atoms are to be created, molecules are to be aligned along a particular axis, and so on. **changeato** gives the user the ability to change a number of factors in the **.ato** file and save it in the correct format. It can be saved to a different filename if required, giving the ability to create copies of **.ato** files. The features currently available in **changeato** are as follows:-

bond	Allows you to add new bonds. Not recommended as it could be overwritten by **fmole**. Better to edit the .**mol** file and run **fmole** once to apply new the values. For a molecule produced by **dockato,** it will be necessary to edit the component molecule files from which the docked molecule was produced, run **fmole** on each of these files, then re-dock the component molecules using the same command as previously.
label	Allows you change atom labels. You will be given a list of the existing atom type numbers and types, and a list of the molecule type numbers and molecule types. You will then be asked which atom type in which molecule type you wish to change, and what fraction of these atoms you want the label changed. The actual atoms to change are chosen at random, unless you want all atoms of that type changed. This is so that the new atom types are distributed at random through the simulation box. Note that it is important when setting up a **.mol** file to ensure atoms are labelled correctly, as it can be quite hard to change them later once the **.ato** file is built. As with **bond** a safer option is to edit the **.mol** file and rebuild the simulation box from scratch, however this may not be possible for molecules that do not have a corresponding template file. If you want to change the labels of all occurrences of a particular atom type, then go to the lines after **natomtypes** (below) to do this.
tether	This allows you to tether molecules of a particular type, or set them free if they are already tethered. When each molecule is tethered its current coordinates are stored and, from then on, a spring constant is applied to hold it close to its tethering location. The interaction of this molecule with other molecules via the reference and empirical potentials, unless the first atom in the molecule is Q atom, in which Q-Q interactions are ignored in the intermolecular potential. The strength of the spring is determined by the value of **tol**, which can also be set with **changeato.**
remove	Allows you to remove molecules from the simulation box – the reverse of **addato**. You can either remove all the molecules of a particular type, or you can remove a specified fraction of them. The molecules removed are chosen at random.
rho	Sets the atomic number density of the **.ato** file. This number density does not include any Q atoms that may be present. When the density is changed (by changing the dimensions of the simulation box) all the molecular coordinates are scaled by a similar fraction so that this process does not introduce any strong overlaps or tension in the revised box. Of course the simulation will need to be re-equilibrated after changing the density since rescaling the coordinates in this way does not fully guarantee there will be no atomic overlaps.
temp	Sets the temperature of the simulation (K).
tol	Sets the force constant for the tethering potential. If this is set to 1.0E9, this will hold the molecules close to their starting position. Adjusting this value for a crystalline lattice controls how much diffuse scattering is predicted.
stepmi	Determines if intra-molecular vibrations will occur. Set to 0 if you want to stop these completely. The actual step size is given at the end of the **.ato** file after the name of each molecule template file.
stepri	Determines whether intra-molecular rotations – rotational groups and dihedral groups – will occur. Set to 0 if you want to stop these completely. The actual step size is given at the end of the **.ato** file after the name of each molecule template file.

stepr	Determines whether whole molecule rotations will occur. Set to 0 if you want to stop these completely. The actual step size is given at the end of the **.ato** file after the name of each molecule template file.
stepm	Determines whether whole molecule translations will occur. Set to 0 if you want to stop these completely. The actual step size is given at the end of the **.ato** file after the name of each molecule template file.
vibtemp	Sets the vibrational temperature. A good general value is 65, but if you want your molecules to appear exactly as defined by the bond lengths and rigid, then use a much larger value like 6.5E5 or larger.
angtemp	Controls the sharpness of bond angles. Default value is 1.0. Use a larger value if you want bond angles to be better defined.
dihtemp	Controls the sharpness of dihedral angles. Default value is 10.0.
ecore	Sets the Lennard-Jones interaction energy for intra-molecular overlaps. Normally set to 0 unless you are trying to restrict such overlaps.
dcore	Sets the Lennard-Jones diameter for intra-molecular overlaps. Normally set to 0 unless you are trying to restrict such overlaps.
box	Allows you change the current box dimensions and shape. You are required to specified the relative lengths of the **a**, **b** and **c** vectors (use negative values to indicate absolute values), then the new crystallographic angles, (αβγ). Note that if you change the box shape you will need to re-equilibrate the simulation as the minimum image convention may move molecules to unexpected places. Type "change" to effect the **box** command. The program attempts to achieve this keeping the density constant unless you specify actual values (negative) for the dimensions, so you will need to re-equilibrate the box after a shape change.
addbond	Same comment as **bond**. However it may be useful for molecules that do not have a corresponding template file.
removebond	Ditto.
add_w_rot	Add a whole molecule rotation group. This is distinct from whole molecule rotations which occur about the laboratory *x*, *y*, or *z* axes chosen at random. With a whole molecule rotation group the whole molecule is rotated about an axis formed from two of the atoms within the molecule. Within EPSR whole molecule rotation groups are treated the same as internal ROT rotation groups.
axisshift	Shifts molecules of a specified type along a particular axis. This shift is done irrespective of possible atomic overlaps, so the simulation box will need to be re-equilibrated after such a shift.
alignmole	Aligns a specified molecule along specified axis. The alignment is performed by first selecting two atoms within the molecule to define a molecular axis. This axis is then aligned with the specified Cartesian laboratory axis. Currently it is only possible to align against a specified laboratory axis, but in the future this could be generalised to align molecules along an arbitrary axis, like (111) or (110), etc.
setimage	Allows atoms on long chain molecules which span the simulation box boundary to interact with their image atom in the neighbouring cells. Care must be taken that one end of the molecule does indeed overlap its mirror atom in the neighbouring boxes, otherwise this parameter has the ability to excessively distort the molecular structure as it tries to satisfy the bonding constraints of both its own neighbours as well as those of its image atom.

unsetimage	This unset the effect of **setimage.**
removerot	Removes a rotational group. This is better done by editing the **.mol** file then running **fmole.** However this command may be needed for molecules that don’t have a template file, such as those created by **dockato.**
natomtypes	Number of atom types in this file – set by the program. After this line there is the option to change the atom label (**atlabel**) for all atoms of this type, reference potential parameters (**epsilon, sigma**), atomic mass (**atmass**), charge (**charge**),), charge radius (**qradius**), and atomic symbol (**atsymbol**), for each atom. In addition an integer **iexchange** can be set to 1 if this atom exchanges with other atoms of the same type, otherwise it is set to 0.

5. Operating the EPSR simulation program

Once we have a box of atoms or molecules, we can transfer the simulation to the EPSR program, to see how our starting configuration is doing in terms of fitting to the diffraction data. In the past there was an intermediate step using **fcluster**, but this is no longer done: the initial equilibration of the simulation box is best done within EPSR itself.

The first step is to set up the **.wts** files – these tell EPSR how to compare the simulated distributions with the diffraction data.

5.1 Setting up the neutron **.wts** files - **nwts**

By far the safest and easiest way to do this is to run the program **nwts**. This delivers the **.wts** in the correct format appropriate to EPSR. Since you need a **.ato** file to run EPSR, **nwts** can read that **.ato** file and hence determine for itself the atomic fractions of all the components. Otherwise it will ask you for the relevant information. **nwts** recognizes the chemical symbols and converts these to a neutron scattering length using the Sears 1991 compilation of scattering lengths. Be careful not to confuse the number of components – the number of atom **types** – in the sample with the number of atom classes, as define by the atomic symbols. Thus the methanol molecule might have 4 atom types, namely C, O, M, and H, but only 3 atom classes, in the form of C, O, and H.

Hydrogen has to be treated differently from other atoms in that it can exchange with other hydrogen atoms in some cases and not others. Therefore if you specify “H” as a chemical symbol you will be asked an additional question about whether it exchanges or not.

For total diffraction data that has not been normalised, the formula used by **nwts** to calculate the weight for a particular correlation between atom type α and atom type β is given simply by

$$w_{\alpha\beta}=b_\alpha b_\beta \qquad (4.1.1)$$

where b_α is the scattering length for component α. There is a set of weights for each pair of scattering atoms in the system. (Therefore Q-atoms should not appear in the list of atoms to be given weights.)

If the data have been normalised to the square of the mean scattering length, then these weights also need to be normalised to the relevant weighted sum of coefficients:-

$$w_{\alpha\beta}=\frac{b_\alpha b_\beta}{\left(\sum_\alpha c_\alpha b_\alpha\right)^2} \qquad (4.1.2)$$

for normalising to the square of the mean scattering length, or

$$w_{\alpha\beta}=\frac{b_\alpha b_\beta}{\left(\sum_\alpha c_\alpha b_\alpha^2\right)} \qquad (4.1.3)$$

for normalising to the mean square of the scattering length. Here c_α is the atomic fraction of component α, and the sum is over all components.

In earlier versions of EPSR this weight also included a factor to convert from per atom to per molecule if the data were normalised per atom. However from EPSR16 onwards all the internal correlations within EPSR are calculated per atom, there is now a factor at the head of the **.wts** file to indicate whether the data have been normalised per atom or per molecule. Therefore **.wts** files produced by **nwts** cannot be used in the earlier versions of EPSR.

For first order difference data, the program asks you to specify which atom(s) were substituted, and then calculates and outputs the change in the coefficients for those coefficients that are affected by the substitution. These then need to be normalised to the corresponding weighted sum of coefficients if the corresponding difference diffraction data were also normalised.

For second order difference data **nwts** proceeds in a similar way, only writing out those coefficients which are relevant to the quantity calculated. However **nwts** always assumes the second order difference data have been normalised.

To setup the **nwts** input file from within **EPSRshell** type:

setup nwts <filename>

The filename is optional and will be prompted for if not specified. This is the name that the final **.NWTS.wts** files will be based on.

fnameato specify the **.ato** file that this input file relates to. This will determine the number of components and the atom symbol associated with each component. It will also assume (initially) that all isotopes are natural and that no isotope substitution has been performed. If that is the case, then file can be immediately saved and run. If not then individual values need to be changed via the following headings.

output This is normally 1 to signify the associated data file has been normalised per atom. If the data were normalised per molecule (very unusual nowadays) then this value should be set to 2.

nsamples Always use 1.

xfrac Ignore and leave the default value (0.0).

normtot 0 if data are not normalised to anything, 1 if normalised to the square of the mean neutron scattering length (equation (4.1.2)) or 2 if normalised to the mean of the squared neutron scattering length (equation (4.1.3)).

normdif Not used unless **nsamples** =2. If **nsamples** =2, then the first order difference may have been normalised to nothing (0) or to the overall weighting factor (1).

ncomponent Number of components – set by the **.ato** file.

Then for each component there is a list of:-

atom Name of atom – set by **.ato** file.

atsymbol The atomic symbol for this atom. This is also set by the **.ato** file, but note that if a recognised symbol has not been set for this atom (usually signified by XXX for the atom symbol) then **nwts** will ask for the correct symbol when it runs.

iexchange This can only be set if the **atsymbol** is H, since in some cases the H atom can exchange with other H atoms (such as the H in ammonia for example). **nwts** assumes there is complete exchange, although if in practice the exchange is only partial, then a revised version of **nwts** will be needed. 1 means the atom is exchangeable with other H atoms, 0 means not.

abundances Lists the mass numbers and their corresponding abundances for this atom. Note that the abundances do not have to add up to 1.0 (unlike earlier versions) since the input values will be normalised within the program. 0 for a mass number signifies the natural isotope – this is the default value.

The next two lines apply only if isotope substitution is performed (**nsamples** > 1).

isubs 1 signifies this atom is substituted, 0 signifies not.

abund2 Lists the mass numbers and their abundances for the substituted atom.

The lines from **atom** to **abund2** are then repeated for each component in the simulation. At the end this file is saved by typing **save** then **e** to exit.

To run **nwts** simply type

nwts <filename>

in the shell. If the filename is not specified then a suitable filename will be searched for.

5.2 Setting up the x-ray **.wts** files – **xwts**.

For x-rays the principle is the same as for neutrons, but the detail is different. The first and foremost difference is that the scattering lengths are actually ‘electron form factors’ and they are Q dependent. They are usually given the symbol $f_\alpha(Q)$ for atom α.

This presents an immediate problem in that EPSR will strictly need to invert the weights matrix for every Q value for which there are data (Section 2.6). Currently EPSR does invert the weights matrix at selected Q values (in steps of ~2Å^{-1}) – so it takes a bit longer initially to get set up, but in fact it only can use one of them to perform the inversion. The point is that according to Section 2.6 the difference $\left(D_i(Q)-F_i(Q)\right)$ is fit to the expression (2.3.5) in Q space. Then the coefficients that come from that fit are used directly in (2.6.4) to produce the EP in r space, using the inversion matrix in (2.6.5) – there is no numerical transform of the data as such – so with x-rays for which value of Q should one choose the inversion matrix?

The correct way to proceed would be to invert the difference data at each Q value to the corresponding partial structure factor using the correct inversion matrix, numerically Fourier transform these Q space differences to r space, and then fit the potential function to each r space function in turn. This would however defeat the whole point of Section 2.3, namely to try avoid all the spurious structure that comes from numerical transforms of data. Alternatively we could fit the coefficients to each PSF using (2.6.6), but this would make the procedure for generating the EP very inefficient if there were a large number of PSFs, which there often are.

To solve this dilemma, EPSR currently uses the Q = 0 inversion of the weights matrix to solve (2.6.5) for x-ray data. This is an approximation which is probably not too bad for most elements, except possibly hydrogen, which can make a non-zero contribution to the weights at Q = 0, but which becomes progressively less dominant as Q increases. Hence although EPSR calculates and stores the inversion matrix at several Q values initially, it actually only ever uses the Q = 0 inversion.

Of course this difficulty only arises when estimating the empirical potential (EP), i.e when going from Q space to r space. For calculating the estimated x-ray structure factor from the simulation, the correct Q-dependent form factors are used.

A second issue is how should x-ray data be normalised? The total x-ray differential cross section, without normalisation is given by:-

$$F(Q)=\sum_{\alpha} c_\alpha f_\alpha^2(Q)+\sum_{\alpha}\sum_{\beta\geq\alpha}\left(2-\delta_{\alpha\beta}\right)c_\alpha c_\beta f_\alpha(Q) f_\beta(Q) H_{\alpha\beta}(Q) \tag{5.2.1}$$

where c_α is the atomic fraction of atom type α, and $H_{\alpha\beta}(Q)$ is the site-site partial structure factor between sites α and β. $H_{\alpha\beta}(Q)$ is defined here in terms of the site-site radial distribution functions:-

$$H_{\alpha\beta}(Q)=4\pi\rho\int_0^\infty r^2\left(g_{\alpha\beta}(r)-1\right)\frac{\sin Qr}{Qr}\,dr \tag{5.2.2}$$

Virtually all x-ray analysis proceeds on the assumption that the normalisation to use is to generate the function:-

$$H_X(Q)=\frac{\left(F(Q)-\sum_\alpha c_\alpha f_\alpha^2(Q)\right)}{\sum_\alpha\sum_{\beta\geq\alpha}(2-\delta_{\alpha\beta})c_\alpha c_\beta f_\alpha(Q)f_\beta(Q)}=\frac{\left(F(Q)-\sum_\alpha c_\alpha f_\alpha^2(Q)\right)}{\left(\sum_\alpha c_\alpha f_\alpha(Q)\right)^2} \quad (5.2.3).$$

However this ignores a fundamental aspect of (5.2.1) in that the first term on the right hand side of (5.2.1), the single atom scattering, is the scattering level that would be seen *if there were no other atomic correlations in the system*, i.e. if all the $H_{\alpha\beta}(Q)$ functions were zero. It therefore represents the Q dependent baseline about which the effects of structure oscillate. The second term in (5.2.1) cannot therefore be more negative than this baseline term is positive, otherwise the scattering would become negative, i.e.

$$\sum_\alpha\sum_{\beta\geq\alpha}(2-\delta_{\alpha\beta})c_\alpha c_\beta f_\alpha(Q)f_\beta(Q)H_{\alpha\beta}(Q)\geq-\sum_\alpha c_\alpha f_\alpha^2(Q) \quad (5.2.4)$$

for all Q values. The single atom scattering, which is normally ignored in this context, therefore places a fundamental limit which any estimates of the partial structure factors have to satisfy.

Before Fourier transforms can be performed on diffraction data we would like to remove the Q dependence of the baseline, as much as this is possible in the situation. This is especially true for EPSR since we ideally do not want to bias the EP in r space by some Q dependence from the form factors. For neutrons this is not a problem, since the baseline is, at least in principle, already flat, so it doesn't really matter how you normalise the data, or even whether you normalise them at all – in the end it is only a Q independent constant or factor, which can easily be taken account of. For x-rays however the chosen normalisation is intrinsic to the process of putting the diffraction data onto an absolute scale.

If a material consists only of elements close to one another in atomic number, use of (5.2.3) will probably not be too serious as the Q dependence of the atomic form factors for the different elements will be similar for the different elements. As a general rule however it is much safer to normalise x-ray data to the single atom scattering:

$$H_X^{'}(Q)=\frac{\left(F(Q)-\sum_\alpha c_\alpha f_\alpha^2(Q)\right)}{\sum_\alpha c_\alpha f_\alpha^2(Q)} \quad (5.2.5).$$

This normalisation means the x-ray data are much more comparable to the neutron data, since we can state categorically that with this definition:

$H_X^{'}(Q)$ oscillates about zero (5.2.6),

and

$H_X^{'}(Q)\geq-1$, for all Q values (5.2.7).

The rule (5.2.7) is certainly not true with the normalisation (5.2.3), which will introduce a shape to the baseline, of the form $B_X(Q)=\frac{\sum_\alpha c_\alpha f_\alpha^2(Q)}{\left(\sum_\alpha c_\alpha f_\alpha(Q)\right)^2}$. The figure below shows an estimate of this function for SiO_2, based on the independent atom form factors:-

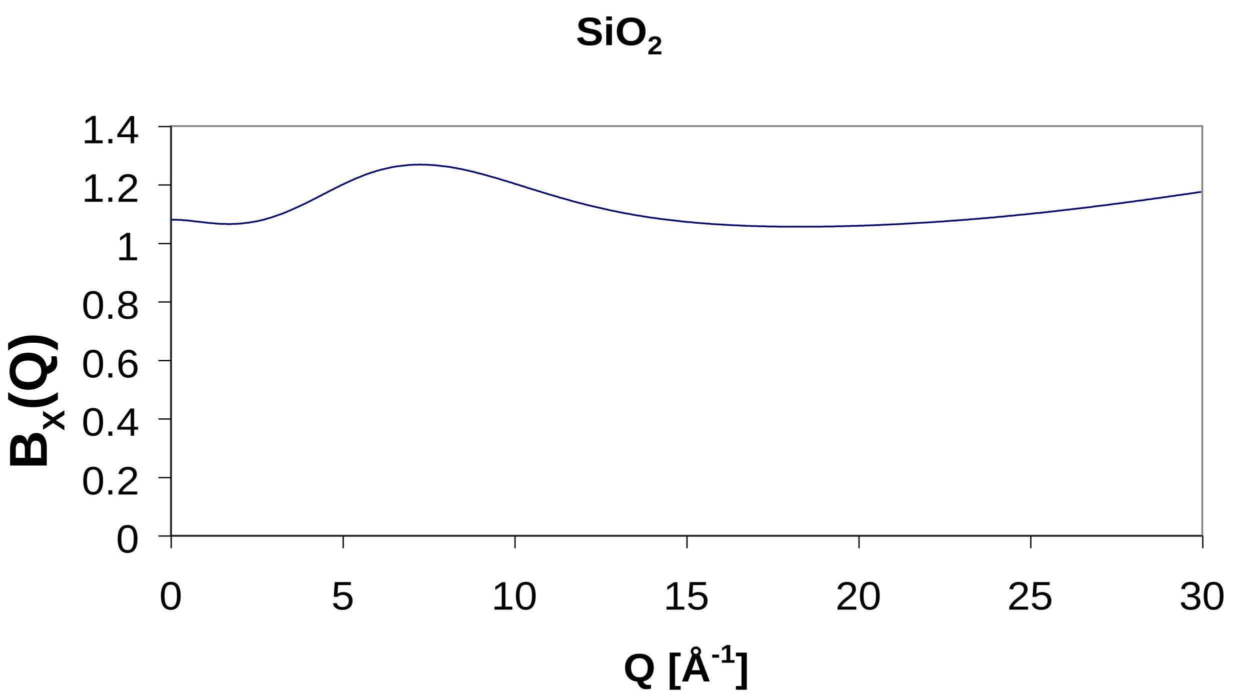


Figure 5.1

The next figure shows the same calculation for H_2O, using so-called 'modified' x-ray form factors (see later in this Section):

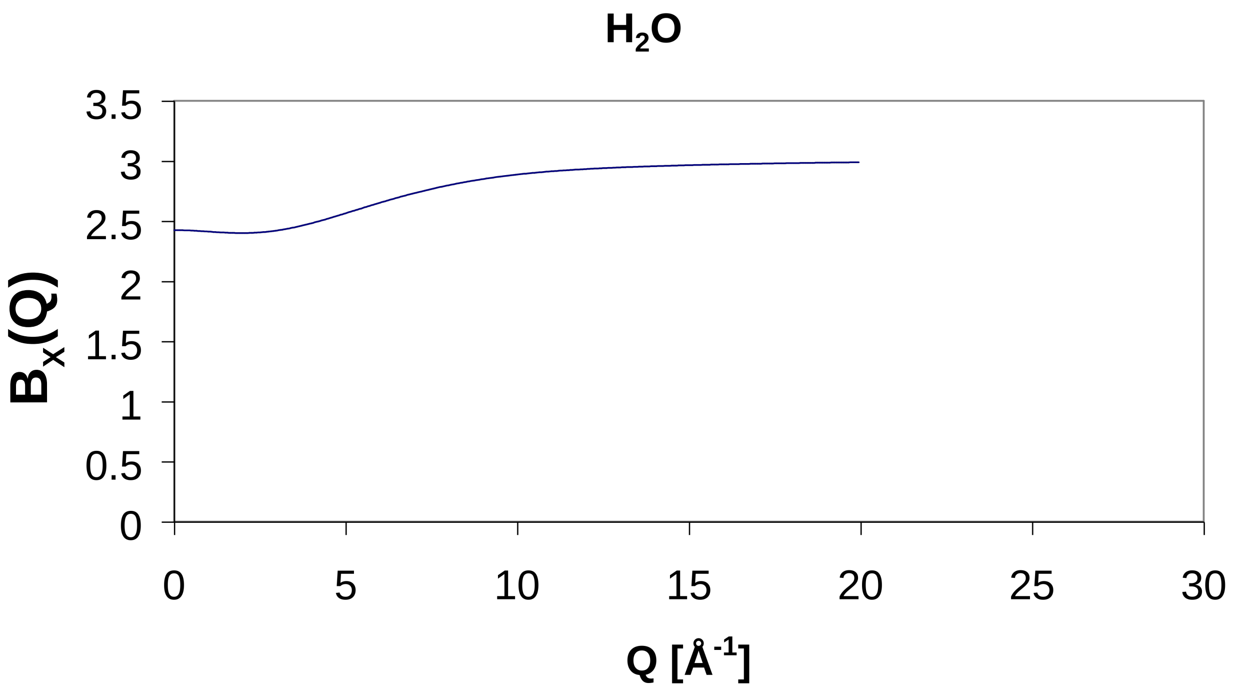


Figure 5.2

Note that the overall shift in these curves is not an issue – *Q* independent shifts do not affect the result in *r* space, and *Q* independent factors only give rise to corresponding factors in *r* space. It is the variation with *Q* that is important.

It is clear therefore that use of the normalisation (5.2.3) introduces a significant *Q* variability, by more than 20% to the baseline about which the structure factor oscillates. This extends over a wide range of *Q* values. Such *Q* variation will generate spurious structure in *r* space.

Fortunately EPSR can deal with either normalisation: when running **xwts** you will be asked to say how the data have been normalised, i.e. according to (5.2.3) or (5.2.5). If the former it is generally found however that fitting the data is more difficult, and the Fourier transform of the data will often behave unexpectedly at low *r*.

Another convenient feature of the normalisation (5.2.5) is that the outcome of the normalisation is the same irrespective of whether the differential cross sections have been calculated per atom or per molecule – the rules (5.2.6) and (5.2.7) will apply in either case. With the normalisation (5.2.3) the outcomes will differ in magnitude between the per-atom and per-molecule definitions by the ratio of the mean number of atoms per molecule.

Fortunately EPSR can deal with both cases, and the first question that is asked when running **xwts** is whether the data have been normalised per atom or per molecule.

xwts then steps through the atomic types and asks for a chemical symbol with which to identify this atom. This is purely for the purpose of generating the electronic form factors and has nothing to do with the atomic class as stored in the **.ato** file. This is so that each atom type can be assigned its own form factor if necessary. Thus for example an atom labelled ‘U’ could be given the form factor for a hydrogen atom, which would be a bit strange perhaps, but is perfectly logical within EPSR!

Another reason for this is to enable the user to specify whether they want to use so-called “modified” atom form factors, (MAFF). The idea here, originally espoused by T Head-Gordon and co-workers [10] and really only applicable when significant charge transfer occurs between two atoms, is that at high *Q* the form factor is dominated by the core electrons, and so behaves as listed in the independent atom form factor (IAFF) table, while at low *Q* the form factor is modified by the shift of valence charge onto the more electronegative atom. The proposal is to set up a modified form factor according to:

$$f'_\alpha(Q)=\left[1-\frac{q_\alpha}{f_\alpha(0)}\exp\left(-Q^2/2\delta_\alpha^2\right)\right]f_\alpha(Q) \tag{5.2.8}$$

where q_α is the amount of charge that is effectively shifted onto this atom from the others. It can have either a positive or negative sign. However note that in order to preserve overall charge neutrality for the system we must ensure that the sum

$$\sum_\alpha c_\alpha q_\alpha=0 \tag{5.2.9}$$

is rigorously satisfied.

IMPORTANT NOTE: currently **xwts** does NOT perform this check: it is left up to the user to ensure the values for q_α satisfy (5.2.9).

As an example, suppose in the case of water we allow 0.5*e* to shift off each hydrogen atom onto the neighbouring oxygen atom, then q_O = -1 and q_H = +0.5.

NOTE that within **xwts** the ratio i $\frac{q_\alpha}{f_\alpha(0)}$ s called ‘alpha’.

The value of the width variable, δ_α, is not well defined. For water it was determined by comparing the MAFFS with those obtained for a free molecule calculation, and was set at 2.2Å^{-1}.[5**,** 10]

An interesting possibility arises here that if charges are used in the EPSR reference potential, then strictly these same charges should appear in the MAFFs used to simulate the x-ray diffraction data. Thus one could imagine choosing the q_α's (and adjusting the Lennard-Jones parameters if appropriate) to give the best fit to all the data without invoking the EP at all. This has been done for water, and it gave surprisingly good results.

Finally **xwts** asks you whether the IAFFs or the MAFFs were used when the data were normalised. This is independent of whether (5.2.3) or (5.2.5) were used for the normalisation.

The output is a file with name '<you specify>**.XWTS.wts**'. The end of this filename is used to signal to EPSR that this is an x-ray dataset. Any other ending for the filename and the dataset and its associated **.wts** file will be treated as for a neutron dataset.

The independent atom form factors in EPSR are obtained from a file called 'f0_WaasKirf.dat' which MUST exist in the home directory for the program to run correctly. This list is based on the 5 coefficient compilation of the form factors due to D. Waasmaier and A. Kirfel, Acta. Cryst., **A51**, 416-413 (1995).

Setting up and running **xwts** is analogous to **nwts**. To setup **xwts** you type:-

setup xwts <filename>.

As before the first line is **fnameato** which signifies the **.ato** file this input file applies to. The next two lines, **output** and **normtot** have exactly the same function as for **nwts**. Next line is:

iafformaff signifies whether the IAFFs or MAFFs are to be used in the normalisation (see above).

ncomponent Determined from **.ato** file.

Then for each component, four values are required.

atom As before this is set by the **.ato** file and cannot be changed.

atsymbol This is initially set by the **.ato** file, but note that it can be changed by the user to any valid symbol that might occur in the list of x-ray atomic form factors (listed in 'f0_WaasKirf.dat' in the start up folder).

maffq This is the value of q_α associated with the modified atomic form factors (see equation 5.2.8 above).

maffdelta This is the value of δ_α associated with the modified atomic form factors (see equation 5.2.8 above).

Once all the relevant values have been checked or set, the input file is saved by typing 'save' then exiting. Finally the **xwts** program is run by typing

xwts <filename>

Again the <filename> will be prompted for if it is not specified on the command line. This produces a file called <filename>.XWTS.wts which is now recognised by EPSR as an x-ray weights file.

5.3 Setting up the **.inp** and **.pcof** files.

Once the **.ato** file has been built and the **.wts** file(s) have been made, you are in a position to set up the simulation. This is done from the EPSR setup menu by typing, within EPSRshell,:-

setup epsr <filename>

The filename doesn’t need to exist before hand, and you do not need to specify the extensions: **setup** will in any case strip off any extensions you supply and put on its own. If the filename exists it will read all the data from that file, otherwise it will set up the values from the default values. You can skip through these values by pressing “Enter” at each line, changing values as you go along. Below is a typical **.EPSR.inp** file showing all the variables that are currently in use.

```
meths.EPSR              Title of this file
feedback    0.9             Confidence factor - should be < 1. [0.8]
potfac      1.0             >0.0 to enable potential refinement, 0.0 to inhibit
ereq        6.98            EP amplitude[0.0].
ereqmin     0.00            Minimum value for ereq [0.0].
ereqmax     0.00            Maximum value for ereq [0.0]. Set maximum value to 0 to ignore.
Ereqstep    0.1   0.25          Set greater than 0 (e.g. between 0.02 and 1.0) to initiate control of ereq. [0.0]
thresh      0.2             Control baseline [0.5]
bias        2           Controls the bias on the steps in ereq. 0 means unbiassed steps. [2]
sfreq       50 1            Sampling frequency for trend line, and no. of times kT for automatic ereq to begin [50 1]
rspcrmin    1.00            Minimum distance for calculating the R-space coefficient [1.00]
rspcfrac    0.5             Fraction of R-space coefficient in control level [0.2]
num_threds  1               No. of parallel threads to be used, (0 to let program choose) [0]
nmolcell    10              Average number of molecules in a cell [10]
control-p   none 0 0            Control pressure [a, b and/or c axes control plus value plus step, none 0 0]
ref_intra   0.000    0.000          Weighting on EP and Coulomb terms for intra-molecular structure [0 0]
sizefactor  1.00000    0.90000  0.00000E+00         Multiplying factor for box dimension, decline rate, and threshold.[1.0 0.9 0.0]
nq          600             Number of Q values. [600]
qstep       0.05            Size of Q step [1/A]. [0.05]
ireset      0           1: complete reset; 2: sets the Empirical Potential to zero
iinit       0           Sets accumulators to zero. Recalculates r and Q. [1]
ntimes      5           Number of MC cycles between potential refinements. [5]
niter       1           Number of potential refinements before exitting. [1]
nsumt       -1              Number of iterations already accumulated. [-1 with reset]
intra       100             Number of iterations between molecule shakes. [100]
rotfreq     5           Number of iterations between internal rotation moves. [5]
inter       5           Number of iterations in running averages. [5]
rho         8.86000171E-02          Atomic number density - will be derived from .ato file
cellst      0.03            Size of r step [A]. [0.03]
rmaxgr      0.000000E+00            Range of g(r) and F.T. (0.0 will use half the cell box) [0.0]
ngrsamples  0           Requested no. of origin molecules to sample g(r). (0 will use 1000 molecules) [0]
fwhm        0.0             Resolution width - Q independent term. [0.0]
fwhmq       0.02            Resolution width - Q dependent term. [0.02 for SLS]
nsmoop      1           1 means background subtraction is ON, 0 means OFF
fnameato    methsbox.ato            Name of .ato file
fnamepcof   methsbox.pcof           Name of potential coefficients file.
revlorch    0.0 0.0             Broadening factor in Q space. [0.0 0.0]
qwidthqmax  -0.02 2             Broadening and maximum Q for Bragg peak calculation
mplicities  1 1 1           No. of unit cells along a, b and c for Bragg peak calculation
hklqmin     0.05 5 0.0          Minimum value of qhkl to be used, minimum radius for Bragg g(r), and Debye-Waller factor
diffuse     0           No. of unit cells along a, b and c for diffuse scattering calculation, maximum l, steps in l and m [0]
rejrate     0.75            Rejection rate [0.75]
qmin        0.05000    0.00000          Qmin for Fourier transforms and for potential fits. [0.05 0.0]
ndata       3           Number of data files to be fit by EPSR

data   1

datafile    cd3od.mdcs01            Name of data file to be fit
wtsfile     cd3od.NWTStot.wts           Name of weights file for this data set
nrtype      5           Data type - see User Manual for more details
rshmin      0.7             Minimum radius [A] - used for background subtraction
szeros      0.0             Zero limit - 0 means use first data point for Q=0
tweak       1.0             Scaling factor for this data set. [1.0]
efilereq    1.0             Requested energy amplitude for this data set [1.0]

data   2

datafile    cd3oh.mdcs01            Name of data file to be fit
wtsfile     cd3oh.NWTStot.wts           Name of weights file for this data set
```

```
nrtype     5            Data type - see User Manual for more details
rshmin     0.7            Minimum radius [A] - used for background subtraction
szeros     0.0           Zero limit - 0 means use first data point for Q=0
tweak      1.0            Scaling factor for this data set. [1.0]
efilereq   1.0           Requested energy amplitude for this data set [1.0]

data   3

datafile   ch3od.mdcs01          Name of data file to be fit
wtsfile    ch3od.NWTStot.wts             Name of weights file for this data set
nrtype     5            Data type - see User Manual for more details
rshmin     0.7            Minimum radius [A] - used for background subtraction
szeros     0.0           Zero limit - 0 means use first data point for Q=0
tweak      1.0            Scaling factor for this data set. [1.0]
efilereq   1.0           Requested energy amplitude for this data set [1.0]
q
```

Many of these do not need to be altered, but some values will need to be specified, normally **fnameato** (name of the **.ato** file), **fnamepcof** (name of the **.pcof** file), **datafile** and **wtsfile** before the simulation can be run. If you are unsure of values to type for these variables you can type “search” against the relevant variable and it will enter the search menu and list one by one all the files in the working folder that correspond to the specified type. If the **.pcof** file does not exist at the outset, then it will be created when the data are saved, but it is essential that an **.ato** file has been specified and read in for this to happen correctly.

Currently you are allowed up to 30 datasets. To increase the number of datasets above the default (1) you must alter the value of **ndata** first, and then go to each of the **data** entries in turn and set the appropriate values of **datafile** and **wtsfile**. You do this by typing “data n” in the EPSR setup menu, where n is the number of the data entry you want to go to – or else you can simply page through all the data sets in turn by pressing “Enter”.

The main list of values which are needed for running the simulation are

feedback This represents the confidence factor in the data. It is worth running several simulations with different values of this variable to see what the effect of different values of **feedback** is – occasionally a better fit can be achieved by using a *smaller* value of **feedback**. Do not be deceived into believing your data are very good and therefore set **feedback** as high as possible. Generally a value of ~0.8 - 0.9 gives the best results: higher than this and you may make the fit worse due to overemphasizing the systematic errors in your data.

potfac This controls whether the EP is refined or not and the size of the increment to the EP at each iteration. Normally set > 0. If this is set to 0 it will not refine the EP. In addition adjustable terms in the RP such as the reference potentials will not be altered either. Once you are convinced you have done all you can with the reference potential, you set **ereq** above 0 to start the EP refinement. Occasionally different values of **potfac** are needed. If the EP grows too quickly or energy is oscillating wildly use a smaller value. If on the other hand it grows too slowly then use a value > 1.

ereq This controls the amplitude of EP. This amplitude is defined in section 2.6.1 and is expected to be of order *kT* (~ 2.5 kJ/mole at 300 K). However wide variations around this value can occur, and in the past the best value to use had to be found by trial and error. With EPSR25 there is the option to search for the best value automatically, Section 5.5 below.

ref_intra Only use this as a last resort – it allows the empirical potential and coulombic potentials to refine the intra-molecular structure as well as the inter-molecular structure. For a simulation containing a very large flexible molecule, changing the ref_intra values might make it able to attain different conformations that would not have been achievable with just the reference potential. However, great care needs to be taken when using it to prevent unfeasible molecular conformations.

sizefactor Normally set to 1.0 0.9 0.0, this controls the shrinkage factor for molecules in the simulation box. If a simulation contains molecules with rings, such as benzene, then at the outset, with a random distribution of molecules, it is possible for some molecules to overlap and for their rings to become intertwined, which with the Lennard-Jones parameters will be impossible to break apart. This usually appears as a very large repulsive energy which never goes away no matter how long the simulation is run. To overcome this, the molecules can be shrunk by a factor, specified by 1/sizefactor, in the early stages of the simulation until any evidence of overlaps has disappeared. Therefore in such cases **sizefactor** is initially made large, like 20 or 30 or 40. Then whenever the energy drops below the third value (0.0 in the above case), **sizefactor** is automatically reduced by a factor (0.9 in the above example) until it returns to it normal value of 1.0. This reduction factor is applied every time the energy drops below the third value above, namely 0.0 in the above case.

nq Controls the number of *Q* values used in the simulation. It defaults to 400 and has maximum value of 2000.

qstep Controls the spacing between the simulated *Q* values. Defaults to 0.05Å^{-1}.

ireset Value either 0, 1, or 2. 0 is the normal value and does nothing. 1 resets the empirical potential to zero, resets the minimum distances between atom pairs depending on their Lennard-Jones parameters, plus sets various other accumulators to zero, sets **nsumt** to -1, and calculates the inverse of the weights matrix. 2 does the same as 1 EXCEPT that the minimum distances, and any other parameters such as maximum distances, Gaussian additional terms, and so on, between atom pairs are NOT reset. **ireset** must ALWAYS be set to 1 when starting a new simulation, and is set automatically (usually to 2 rather than 1) whenever parameters which define the potential are altered. It is a good idea to reset the potential (ireset = 2) after the simulation has been run for a while to make sure that the EP did not become overly biased by the starting configuration of atoms. If you get the same overall structure after resetting the EP and letting it grow again, it probably means the structure is reliable.

iinit Value either 1 or 0. This does a reset of all the accumulators, recalculates the uniform atom distribution, the *r* and *Q* scales, and recalculates the inversion of the weights matrix. It does not reset the empirical potential. It can be useful when you change one or more of the **.wts** files, but do not wish to modify the current simulation. It will happen automatically if you change the value of **feedback**, change the number of datasets to be fit, or number of *Q* or *r* values.

ntimes Number of Monte Carlo (MC) cycles between potential refinements. Default: 5

niter Number of potential refinements before exiting. This is normally 1, since on exiting all the current distributions are saved. Use of the script facility to perform repeated runs is better than increasing the value of **niter** since it means if you wish to view the current simulation, then all the distribution files will be up to date. The time spent starting and stopping EPSR is small compared to the time actually spent simulating.

nsumt This controls the accumulation of distribution functions. If **nsumt** is 0 or positive then it tells us how many configurations have been included in the current set of distribution functions and partial structure factors. If **nsumt** is -1, then a running average of the most recent configurations is maintained. The number of configurations in this running average is roughly twice the value of **inter.** The smaller the value of **inter**, then the smaller is the number of configurations included in the running average. This latter feature is used when a simulation has not reached equilibrium or a fit is still being searched for, so that there is some statistical averaging in the distribution functions that are output, but they will evolve as the configuration of atoms and molecules changes.

intra Number of whole molecule moves and internal molecule rotations between molecular shakes. Default 100.

inter Number of iterations in running averages. This is used when **nsumt** = -1.

rho The number density of the current .ato file. This is set by the program and cannot be changed.

cellst Spacing in *r* space (Å). Default is 0.03.

fwhm Resolution width in *Q* space – *Q* independent term.

fwhmq *Q* dependent resolution width. Hence the resolution width for a given *Q* value is

Δ(*Q*) = **fwhm** + *Q* * **fwhmq**

nsmoop 1 or 0. Controls whether or not background subtraction is performed prior to fitting the EP. The background is determined from the requirement that below a certain distance in *r* space, the Fourier transform of the data should adopt a specific value, determined from the weights files. This distance is set by the value of **rshmin** for the corresponding dataset. If **nsmoop** is set, as in the above example, the EPSR uses this minimum distance to determine, by direct Fourier transform of the data, a background function in *Q* space to be subtracted from the data prior to estimating the EP. Note that the output data is always the input data WITHOUT this background function subtracted (but interpolated onto the *Q* scale of the simulation), irrespective of whether **nsmoop** is set or not, so you can always see how the simulation is doing compared to the original data. Generally purists are not comfortable with the idea of subtracting a background from their data, thinking it might bias the overall outcome. In fact this background subtraction is something that is done in almost every data analysis scheme, and in the present instance it usually leads to a better fit rather than worse, since it helps to eliminate residual systematic error from the data. Therefore a recommendation is that **nsmoop** is left set (=1) which is the default setting.

fnameato The name of the .ato file corresponding to the current simulation. If one has not been set or a new one is required, the current working directory can be searched by typing “search<CR>”

fnamepcof The name of the file containing the Empirical Potential coefficients. The current directory can be searched for this file if one is not specified. However it is also perfectly normal to give this file a name, even ift it does not exist, so that when the current input file is written a new **.pcof** file is generated.

qmin The smallest value of *Q* to be used when setting up the Empirical Potential.

Bragg peak calculation in EPSR is possible provided certain values have been set:-

qwidthqmax Sets the width of the Bragg peak and the maximum Q to which Bragg peaks will be calculated. **qmax** must be set > 0 for any Bragg peaks to be calculated, but if it is too large the number of Bragg peaks becomes huge and the calculation is very slow. The assumed peak shape is Gaussian if **qwidth** is > 0 and Lorentzian if **qwidth** is < 0. Only the modulus of **qwidth** is used to give the actual width of the broadening function.

mplicities Controls the spacing between the Bragg peaks. (1,1,1) will calculate all the Bragg peaks compatible with size and shape of the simulation box. (3,2,1) would divide the simulation box into 3 primitive cells along the **a** axis, 2 primitive cells along the **b** axis, and 1 primitive cell along the **c** axis. This reduces the number of Bragg peaks to be calculated because where there are more than one unit cell along a crystallographic axis, the unit cells are averaged over the number of unit cells in that direction. But it also means loss of information about diffuse scattering that arises from the disorder from one primitive unit cell to the next. Currently the Bragg peak calculation in EPSR is not very efficient and needs attention.

hklqmin Specifies the minimum Q for which Bragg peaks will be calculated, the radius at which the Fourier transform of the data to real space is based on the Bragg peaks, and the Debye-Waller factor. The last value should be left at zero, otherwise it will interfere and possibly affect the disorder intrinsic to the simulation. A description of how these values work can be found in the reference [11]. If the minimum Q value is left at 0, a very large peak near Q = 0 will appear corresponding to the self correlation of atoms with themselves.

Following this there is a section where some of the values from the **.pcof** file are set. Here is a typical example of what this file looks like:-

```
roverlap    0.00000000            Minimum allowed intermolecular separation between two atom types.
rminfac     0.00000000            Factor to set the minimum separation between pairs. [0.0
rminpt      9.00000000            Radius at which potential truncation begins
rmaxpt      12.0000000            Radius at which potential truncation goes to 0.0
rbroad      0.00000000            Controls potential decay. [0.0]
expecf      0.300    0.000        Potential decay for short distance repulsive term and gradient limit. [0.3 0]
reppottype  exponential           Repulsive potential type: exponential or harmonic. [exponential]
addpottype  modmorse              Additional potential type: Gaussian or modmorse. [Gaussian]
rcharge     0.00000000            Calculates energy due to molecular polarisation. [0.0]
power       12.0000000            Repulsive power in Lennard-Jones potential. [12]
ncoeffp     120            Number of coefficients used to define the EP. Set by program.
pdmax       12.0000000            Maximum distance of Empirical Potential. Set by program.
pdstep      0.100000001           Spacing between coefficients in r. Set by program.
npitss      1000           Number of steps for refining the potential. [1000]
paccept     5.00000024E-04            Acceptance factor for potential refinement. [0.0005]
Gaussian    F            Select T for Gaussian representation of EP. Otherwise Poisson. [F]
psigma2     0.02000             Width for potential function. [0.02]
refpotfac   1.00000             Factor to apply to reference potential. [1.0]
q
```

It also carries information about how the EP is generated, such as the value of σ_Q in equation (2.3.5) (**psigma2)**, the hardness of the Lennard-Jones core (**power)**, the range of the potential truncation functions (**rminpt** and **rmaxpt**) (equations(2.1.3) and (2.1.4)), and the type and hardness of the interatomic repulsive potentials (**expecf**) used to prevent atomic overlap. The second value of **expecf** sets the gradient limit on the repulsive potential (g_{lim} in equations (2.7.1, 2.7.2)).

Normally the user will only need to modify the values of **rminpt**, **rmaxpt**, and **expecf**. The actual minimum distance for each atom pair is set to the largest of **roverlap** and $\mathrm{rminfac} \times \frac{(\sigma_\alpha + \sigma_\beta)}{2}$ **,** or the distance, if any, specified for that pair at the end of the input file.

After this section, the parameters associated with each diffraction dataset are assigned.

ndata Number of datafiles to be read in

For each dataset the following values need to be setup.

datafile The name of the data file required. Typing "search" at this point will initiate a search for datafiles of the specified extension. "*" can be used as a wildcard in searching for files.

wtsfile The name of the neutron (*.wts) or x-ray (*x-ray.wts) file associated with this dataset. The extension is always .wts for these files, so if "search" is typed, it will list only .wts files.

nrtype This specifies the type of the data file. Currently it can take one of five values. In all cases it is expected the data file will consist of a simple column format with one entry per line. A distinct file name is needed for each distinct data set.

1 Column format consisting of Q, S(Q) values in pairs, assuming point format

2 Column format consisting of Q, S(Q) and error bar values, assuming point format.

3 Old GENIE show data format, assuming the data are in histogram format.

4 Old GENIE show data format, assuming the data are in point format.

5 Gudrun output format, assuming data are in histogram format.

rshmin The minimum radius value for this dataset. This is used if background subtraction is being performed (**nsmoop** = 1).

szeros The *Q* = 0 limit for this dataset. If set to zero its value will be ignored. This in effect represents an extra datapoint (the compressibility limit) to be fit.

tweak Scaling factor for this dataset. Normally set to 1.0.

efilereq Weighting on this dataset when setting up the inversion of the weights matrix. Normally 1, but can be increased or decreased to give this particular dataset more or less emphasis. If set to 0, this dataset will not be included in the refinement, but the fit to this dataset is still calculated.

There follow the actual minimum separations for each atom pair in the **.ato** file. (These of course are inter-molecular atom-atom separations and for those atom pairs on the same molecule that are not otherwise connected by the specified molecular bond distances.

Thus if the lines

Si-Si 0.9

Si-O 1.335

O-O 2.214

appeared at the end of the **epsr setup** menu, this would mean a minimum distance of 0.9 Å for Si-Si pairs, 1.335Å for Si-O pairs , and 2.214Å for O-O pairs. They can be altered simply by typing in new values after the prompt.

These values are read from the **.pcof** file. Here is what the **.pcof** file looks like here:-

```
3
O1  O1
0.00000000   0.00000  2.6000  0.1900  0.5000E+00  4.5000  0.4000 -0.3600E+01
0.00000000 .......
O1  H1
0.00000000  0.00000000
0.00000000 .......
H1  H1
1.50000000  12.0130606
0.00000000 .......
```

The first value gives the number of atom pairs listed in the **.pcof** file. Then for each atom pair are the labels of the two atoms involved, followed by a list of the minimum distance and its weighting coefficient (the latter is not shown when using **setup epsr**), the maximum distance and its weighting coefficient (not shown in **setup**), if present, and then up to 5 sets of 3 values corresponding to the position, width and height of additional terms to the RP. These are normally Gaussian, but in this particular case, for the O1-O1 pair, because **refpottype** has been set to "modmorse" (see above **.pcof** file), the first of these will be a Gauss6 potential as defined by equation (2.7.4). The second and any subsequent terms are Gaussian as defined by equation (2.7.3). Figure 2 shows the resulting RP that results from these choices. Note that if one or more of the positions or widths of these additional terms is < 0, the EP for that term will not be further refined.

In the above case the O1-O1 minimum distance has been set to 0 and the corresponding weighting coefficient is also zero. For the O1-H1 pair no minimum distance is set, but for the H1-H1 pair the minimum distance has been set to 1.5Å with a weighting coefficient of 12.013, which, because it is > 0 will be adjusted automatically as the simulation proceeds. In order to fix this coefficient it is necessary to give an amplitude < 0, but since there is no access to the minimum or maximum distance weighting coefficients from the **setup** menu, setting this value can only be achieved by editing the .**pcof** file when EPSR is not running.

Minimum distances can play a powerful role in helping to avoid the empirical potential from generating spurious structure at low *r* from a dataset which is hard to fit. If atoms are found below the specified minimum distance, then the corresponding coefficient will grow in magnitude until the atoms are eliminated, thus overriding the EP. If no atoms are found below the minimum distance the coefficients are gradually diminished in amplitude until they reach equilibrium. Once set they can also replace to some extent the role played by the repulsive Lennard-Jones potential, since by adjusting the value of **expecf** one can readily control the hardness of this repulsive potential.

5.4 Running the simulation

Under EPSRshell running the simulation is straightforward, providing all the necessary input files have been setup. Simply typing "**epsr** <filename>" will put the simulation through one iteration. If the filename is not specified then it enters the search menu to find a suitable **.inp** file. Or else the simulation can be run repeatedly from a script file, as described above in Section 3.4.

5.5 Automatic control of **ereq.**

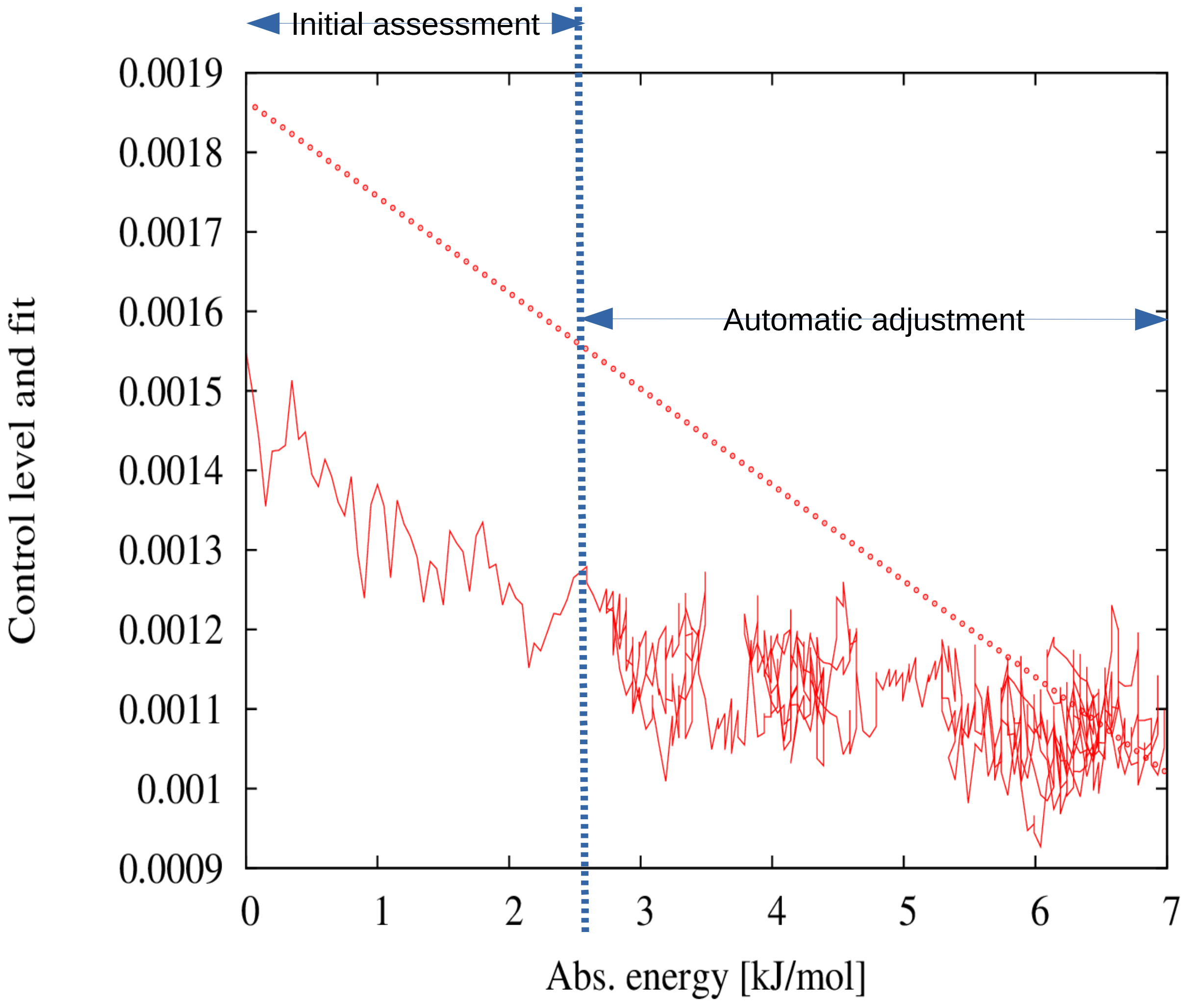
Section 2.6.1 has outlined the rationale for defining and setting the amplitude of the Empirical Potential in EPSR25. It is to be hoped that the same definition can now be used in future versions of EPSR. However it also allows us to consider whether there is some way we can control **ereq** automatically, and so avoid the arduous process of trial and error to find the best value for any given dataset. EPSR25 implements a scheme to do this, and you will notice from the above example there are a number of new lines in the EPSR.inp file. These are repeated below for clarity:

```
ereq        6.98            EP amplitude[0.0].
ereqmin     0.00              Minimum value for ereq [0.0].
ereqmax     0.00              Maximum value for ereq [0.0]. Set maximum value to 0 to ignore.
ereqstep    0.1   0.25            Set greater than 0 (e.g. between 0.02 and 1.0) to initiate control of ereq. [0.0]
thresh      0.2             Control baseline [0.5]
bias        2             Controls the bias on the steps in ereq. 0 means unbiassed steps. [2]
sfreq       50 1            Sampling frequency for trend line, and no. of times kT for automatic ereq to begin [50 1]
rspcrmin    1.00              Minimum distance for calculating the R-space coefficient [1.00]
rspcfrac    0.5             Fraction of R-space coefficient in control level [0.2]
```

A number of new ideas are introduced here. Starting from the bottom, **rspcfrac** and **rspcrmin** are used to define a new "Quality Factor" that represents the goodness of fit (χ^2)in both *Q* and *r* spaces. The two χ^2 are combined using the value of **rspcfrac** and the *r*-space fit is calculated from **rspcrmin** onwards. The resulting quality factor is now shown as the **7th column** of the **.EPSR.erg** file. It can also be plotted as a function of $A^{(EP)}$ (column 4 in the **.EPSR.erg** file) using plot type 30 in the **plot** menu (see section 2.6.1 for definitions).

ereqstep (4th line above) controls the size of the step between successive values of **ereq**, expressed as a fraction of kT. Thus at 300 K (kT~ 2.5 kJ/mole) and stepsize of 0.1 would correspond to changes in **ereq** of ~0.25 kJ/mole. The second value on this line gives the actual step used in kJ/mole and does not need to be set by the user as it will be automatically set by the program.

To estimate the way the quality factor is varying with $A^{(EP)}$ a straight trend line is fitted to the most recent **sfreq** lines in the **.erg** file. In the above example this number is 50. The slope of this line can be used to assess whether to increase or decrease **ereq** automatically**.** The figure below shows a typical example of plot type 30, showing the trend line as a series of dots.



To get the process started, you will notice that **sfreq** actually has 2 values, 50 and 1. The 50 relates to the number of steps over which to assess the current trend line. At the beginning, when there is no information on how large **ereq** should be, it is necessary to increase **ereq** uniformly, to build up an assessment on which to calculate the trend line. How far this monotonic increase in **ereq** proceeds before switching over to automatic is decided by the second value of **sfreq**. In the above example this value is 1, so the initial assessment proceeds for 50 steps, until $A^{(EP)}$ = 50x0.25 = 12.5 kJ/mole . If it had been 2, then it would have proceeded until $A^{(EP)}$ = 100x0.25 = 25 kJ/mole, and so on. This way the user controls how far the initial assessment proceeds before automatic control takes over. For glass structures, where the EP needs to develop a considerable amplitude before it takes effect, it might be necessary to make **sfreq** 50 10, or 100 10, or more.

The value of **bias** in the above input should be left at 2 – it is a left over from various tests with the automatic procedure and may eventually disappear.

The value of **thresh** controls whether the value of **ereq** is increased or decreased. If the gradient of the trend line is > **thresh** times the average modulus of the gradient of the trend line, then **ereq** is reduced, while if the gradient of the trend line is less than this value, **ereq** will be increased. Setting **thresh** to 0 can be done, but runs the risk of letting **ereq** increase indefinitely without improving the fit. A good value is ~0.2 in practice. Hence in the figure above **ereq** would be increased. Note however that this will ONLY happen if the quality factor IMPROVED as a result of the previous move. If it got worse, **ereq** is left unchanged, whatever the slope on the trend line.

Whether **ereq** is controlled automatically is set by the value of **ereqstep**: a value of 0 means no automatic control of **ereq**, while any positive value will cause it to start controlling. The second value on this line is written by the program, and does not need to be changed. The allowed minimum and maximum values of **ereq** are set by **ereqmin** and **ereqmax.** Zeros for these values means they will be ignored and **ereq** will take on any value that is allowed from the gradient of the trend line.

To initialise automatic control, it is a good idea to set **ereq** to zero, and set **iinit** to 1. This forces the **.erg** file to be rewound and all accumulators set to zero, so that a new trend line can be determined. All of these operations can be achieved simply by setting setting **ireset** to 2. Of course **ereqstep** has to be non-zero for this to work.

5.6 Output files and data formats

In the course of each loop of the run script, the program updates the **.EPSR.inp, .ato** and **.pcof** files, as well as producing a **.EPSR.out** file which lists a summary of some of the information produced by EPSR. It is a good idea to check this information from time to time. Typical questions are: are the energy and pressure sensible? Does the fitting factor (chi-square) look sensible and has it diminished since the beginning? Normally energies should be of the order of some kJ/mole and negative (which indicates a bound system). Thus for water you might get ~-40kJ/mole, while for silica you might get –4000kJ/mole, due to the very large electronic charges in that system. Pressures tend to jump around a lot due to the ever changing nature of the EP, but even if a pressure of say 1000kbar occurred that would indicate serious atomic overlap in the simulation.

The program also produces a **.EPSR.erg** file which lists the energy, pressure, chi-square, absolute energy (of the empirical potential only) and a number of other variables which change as the simulation proceeds. This is rewound whenever **ireset** = 1 or 2, otherwise new entries are added to the end of the file. It also produces a **.EPSR.uni** file which lists the uniform atom distribution being used in the simulations.

The remaining files have a simple column format, with *Q* or *r* values on the leftmost column, and then data and error in pairs of columns for each data file present. Section 2.3 listed the various types of files produced. This column format makes them easy to read into spreadsheets and plotting programs. Up to 201 columns are allowed in any one file, meaning that if each distribution has a set of values and standard deviations, it will occupy 2 columns in the output file, so up to 100 distributions are allowed in any given file. If more than 100 distributions are needed, then a new version number of the file is generated, 01, 02, 03...etc. We do not support any particular plotting format on the understanding that most users will want to select their own preferences, but these files can be readily plotted by GNUplot, and entering the **plot** menu allows you to generate suitable plotting files for displaying the data in GNUplot. Indeed the **plot** menu has a **p** command which allows you to plot data on line. Almost all the output files can be plotted using the **plot** menu. Equally the files can easily be read into Excel or Origin or other spreadsheet to view in whatever manner you would prefer.

5.7 Reviewing the output

This step is absolutely crucial! However you choose to look at the results, it is essential that you review them, before proceeding to calculate other quantities based on the simulation box. This is done within EPSRshell by means of the **plot** command described previously, or to view the box of molecules and atoms use **plotato**. Typing **plot** from within the shell causes the file **plot_defaults.txt** to be loaded from the current working folder. To save time this file should be copied from another folder if it does not exist since creating it from scratch is tedious. This file can be edited to generate different kinds of plots, or new plot types can be added from within the menu by increasing the value of **npt**. Typing “l” within the plot menu will produce a list of the currently available plot types. Section 3.6 gives more details about plot commands.

6. Auxiliary routines

6.1 Input and output data format and running.

Once the EPSR simulation is running satisfactorily, or even before that time, a number of other quantities related to the structure of the system being studied can be calculated. Typically one might calculate separately the site-site radial distribution functions (although with the current version of EPSR these are already output by the simulation), bond angle distributions, coordination numbers and their distribution, cluster sizes, ring and chain sizes, void distributions, spatial density functions, and so on. Therefore a series of auxiliary functions are provided within EPSRshell, and in principle new routines will be introduced into the shell to meet particular needs.

As explained in Section 5.4 above the output format has been set the same for all these routines so that the data are readily available for a number of common plotting platforms. If a particular platform cannot read these output files, then it would not be difficult to invent a routine to do the conversion. As already stated the **plot** menu can be used to generate simple GNUplot plots of any of the output files.

By the same token creating the input files for the auxiliary routines is entirely analogous to that used for the EPSR input files. Simply one types “**setup** <program name> <filename> to enter the setup menu for that program. And as with EPSR you are presented with a list variables which can be altered as necessary. There is no need for the specified file to exist, but if it does not exist it will be created with the default values when setup menu is exited.

Note that performance of the variable **nsumt** in all the routines in this section is identical to that described in Section 5.3, except that if **nsumt** = -1, one simply gets the relevant distribution function from the current box – there is no running average in these cases.

6.2 partials – calculate the site-site radial distribution functions.

This is invoked with the shell command “partials <filename>” where the filename can be set up with the command “setup partials <filename>” in the usual way. A typical **.PARTIALS.dat** file looks like this:

```
h2o-AM1.PARTIALS            Title of this file
fnameato    h2o-AM1.ato            Name of .ato file
nz          300             Number of r values [max: 1000]
rnormalise  1.0             Controls the r- or z-range of distribution [1.0]
nsumt       -1              Number of configurations already accumulated
atom-c      O1              Centre atom type [*]
atom-z      *               Centre atom type for rho-z [*]
atom-r      *               Centre atom type for rho-r [*]
ndist       2               Number of site-site distributions
q
    1    O1  O1
    2    O1  H1
```

It is probably the simplest since it really only requires the name of the **.ato** file to run. It will produce a site-site radial distribution function for each unique pair of atom types in the **.ato** file. The RDFs are calculated out to the largest distance for which a complete spherical shell can be contained in the simulation box. If one of **atom-c**, **atom-z**, or **atom-r** is specified then the density of the other atoms around these centres is calculated instead of the radial distribution function. For **atom-z** the density is calculated as a function of distance along the *z*-axis, and averaged over the *xy* coordinates at each *z* value. For **atom-r** the density calculated as a function of radial distance from the *z* axis and averaged over the *z* coordinate.

6.3 coord – calculates the coordination number and coordination number distribution between specified atom pairs over a specified distance range.

This is invoked with the EPSRshell call “coord <filename>” where the filename is setup with the command “setup coord <filename>. Here is an example of the **.COORD.dat** file that is generated by this process – this lists all the variables in the COORD menu:-

```
h2o298tot.COORD            Title of this file
fnameato    h2o298tot.ato            Name of .ato file
nsize       200            Maximum coordination number (max 200)
nsumt       -1            Number of configurations already accumulated
atom-c      *            Centre atom type [*]
atom-z      *            Centre atom type for rho-z [*]
atom-r      *            Centre atom type for rho-r [*]
ndist       3            Number of coordination number distributions

distribution   1

atom1       OW            Atom type 1 to define a coordination number
atom2       OW            Atom type 2 to define a coordination number
rmin        1            Minimum distance for this bond
rmax        3.4            Maximum distance for this bond
rminrmaxc   1 2            Minimum and maximum distance for atom 2 from atom c
distribution   2

atom1       OW            Atom type 1 to define a coordination number
atom2       HW            Atom type 2 to define a coordination number
rmin        1            Minimum distance for this bond
rmax        2.4            Maximum distance for this bond
rminrmaxc   1 2            Minimum and maximum distance for atom 2 from atom c

distribution   3

atom1       HW            Atom type 1 to define a coordination number
atom2       HW            Atom type 2 to define a coordination number
rmin        1            Minimum distance for this bond
rmax        3.0            Maximum distance for this bond
rminrmaxc   1 2            Minimum and maximum distance for atom 2 from atom c
q
 OW  OW
   1.00000    3.40000    4.76333    1.08760
 OW  HW
   1.00000    2.40000    1.78222    0.64920
 HW  HW
   1.00000    3.00000    5.42389    1.28486
```

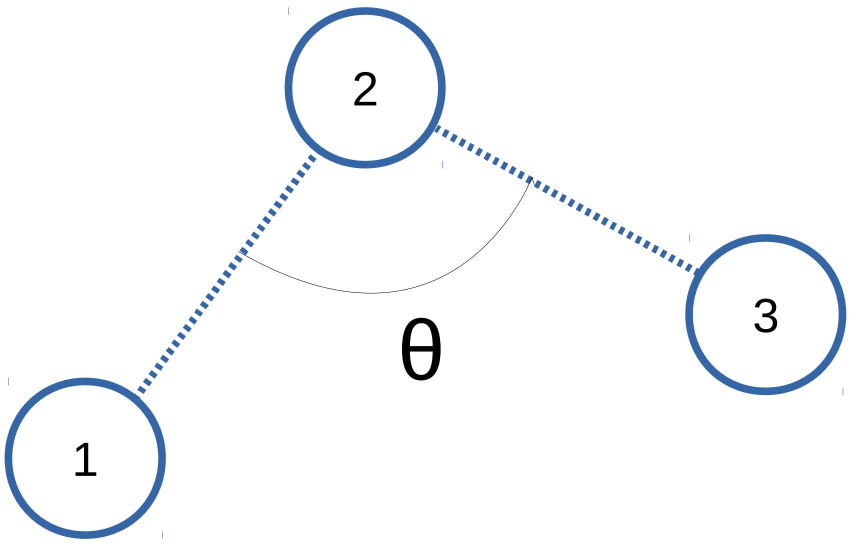
Note how at the end the procedure has written out the atom labels, the radius ranges, the coordination numbers and their standard deviations for each of the specified pairs of atoms. The corresponding **.COORD.n01** file will contain the distribution of these coordination numbers and also the dependence of the average atom separation on the coordination number, which can be plotted using **plot**. Hence each distribution of a **.COORD.n01** file has 4 columns instead of the usual two.

As with **partials** the coordination number can be calculated as a function of distance from a particular centre, defined by **atom-c**, **atom-z**, or **atom-r.** The distance range is set by the values of

rminrmaxc. For **atom-c** this would be the radial distance from that particular site, but for **atom-z** it would simply be the distance along the *z* axis from that site, and for **atom-r** it would the radial distance in the *xy* plane from the *z* axis. These last two cases therefore apply to systems with planar and cylindrical geometry respectively.

6.4 triangles – calculate the distribution of included angles between triplets of atoms

This program allows the calculation of the distribution of included angles of a triplet of atoms separated by specified distances. Atoms are specified in triplets and the angle calculated is the included angle formed by the middle atom of the triplet:-



In the case of this diagram the types of the three atoms would be given in the input file, and the allowed range of distances 1-2 and 2-3 would be specified. Whenever a triplet is found that satisfies the specified atom types and distances, cos θ is calculated via the cosine rule, and the results histogrammed against θ, after dividing by the sin θ dependence that would occur for a completely random distribution of angles.

Here is an example of the input file to the **triangles** program (called h2o298tot.TRI.dat):

```
h2o298tot.TRI            Title of this file
fnameato    h2o298tot.ato              Name of .ato file
nsize      100            Number of cos(theta) bins [100]
nsumt      -1            Number of configurations already accumulated
atom-c      *            Centre atom type [*]
atom-z      *            Centre atom type for rho-z [*]
atom-r      *            Centre atom type for rho-r [*]
ndist      1            Number of tri-angle distributions

triangles   1

atom1      OW            Atom type 1 to define a triangle
atom2      OW            Atom type 2 to define a triangle
atom3      OW            Atom type 3 to define a triangle
ltype      1            Single atom3 types [1], or multiple atom3 types [2]?
rmin12      1            Minimum distance for atoms 1 and 2
rmax12      3.4            Maximum distance for atoms 1 and 2
rmin23      1            Minimum distance for atoms 2 and 3
rmax23      3.4            Maximum distance for atoms 2 and 3
rminrmaxc   1 2            Minimum and maximum distance for atom 2 from atom c
q
```

If ltype is set to 2, then for atom3, instead of selecting just one of the atom3 atoms at the distance atom2 to atom3 it finds all of them and forms the centroid of those atoms to determine the vector going from 2 – 3. This can be useful when attempting the plot the distribution of angles of a dipole moment if this does not lie along one of the bonds.

As with **partials** and **coord,** these distributions can be calculated as a function of distance from a specified site as defined by **atom-c, atom-z,** or **atom-r,** with the relevant distance range defined by **rminrmaxc.**

6.5 torangles – calculate dihedral angle distributions.

A dihedral angle within a molecule (or ‘torsional angle’ – hence ‘torangle’) is defined in terms of 4 atoms, 2 to define the bond about which the angle will be measured, and 2 to define the two planes whose relative angle is to be measured. Dihedral angles have already been discussed with reference to the **dockato** program. Referring to Fig. 4.3, we see that the triangles formed by atoms 1 – 4 – 6 and atoms 1 – 4 – 3 form two different planes. The dihedral angle is the angle between these planes, with the sign of the angle determined by using the right hand rule. In **torangles** this angle is calculated by generating three vectors namely 1 – 4 (the rotation axis), 1 – 6, and 1 – 3. The vector product of 1 – 4 with 1 – 6 gives a vector perpendicular to the plane of triangle 1 – 4 – 6, while the vector product of 1 – 4 with 1 – 3 gives a vector perpendicular to the plane of triangle 1 – 4 – 3. The scalar product of these two perpendicular vectors, when expressed as unit vectors, gives the cosine of the angle between these vectors, which in turn defines the magnitude of the rotation in the range 0 and 180° between the two planes. The direction of the rotation is given by forming the vector product of the first perpendicular vector with the second perpendicular vector, which will give yet another vector either parallel or antiparallel to the rotation vector 1 – 4. If it is parallel the sign of the rotation is positive, while if it is antiparallel the sign of the rotation is negative. In the example given here it would be negative. However if we specified the order of the vectors differently, namely 1 – 4 (the rotation axis), 1 – 3, and 1 – 6, then the direction of the rotation would have been seen to be positive, since the rotation from the plane 1 – 4 – 3 to 1 – 4 – 6 involves a right handed rotation about the 1 – 4 axis.

A typical input file to **torangles** is given below:-

```
eth4wat6.TOR            Title of this file
fnameato    eth4wat6.ato            Name of .ato file
nsize      180            Number of cos(theta) bins [180]
nsumt       198            Number of configurations already accumulated
ndist      2            Number of tri-angle distributions

torangles   1

atom1      CM            Atom type to define which molecule type
ltyp1      5            First atom number to define the axis
ltyp2      8            Second atom number to define the axis
ltyp3      1 1            Two atom numbers for vector from atom 1
ltyp4      9 9            Two atom numbers for vector from atom 2

torangles   2

atom1      CM            Atom type to define which molecule type
ltyp1      1            First atom number to define the axis
ltyp2      5            Second atom number to define the axis
ltyp3      2 3            Two atom numbers for vector from atom 1
ltyp4      6 7            Two atom numbers for vector from atom 2
q
```

The first five lines are the same as for many other EPSR auxiliary routines. (Note that blank lines are written out to aid visualisation but they are ignored by the reading routines.) There are two distributions to be defined in this case, for the ethanol molecules in a simulation of ethanol and water.

atom1 is used to signal to the program which molecules in the box the following numbers apply to. Normally this would be the first atom in the molecule, which is the one used to identify the type of each molecule.

The first distribution is for the dihedral angle about the middle C-O bond (atoms 5 and 8, defined by **ltyp1** and **ltyp2**) formed by the carbon atom (**ltyp3**, atom 1) at one end of the molecule and the hydroxyl hydrogen atom (**ltyp4,** atom 9) at the other end of the molecule. The second distribution is for dihedral angle about the C-C bond (atoms 1 and 5, defined by **ltyp1** and **ltyp2**) using the midpoint of two hydrogen atoms on the end methyl group (**ltyp3**, atoms 2 and 3) and the midpoint of two hydrogen atoms on the middle methyl group (**ltyp4**, atoms 6 and 7) to define the angle between the corresponding planes. (The atom numbers to use will not be obvious until you inspect the relevant **.ato** or **.atm** files.). Thus the three vectors to be used to define the dihedral angle for the first distribution will be 5 – 8 (rotation axis), 5 – 1, and 5 – 9, in that order. If the order of the either the first two or last two sets of numbers was reversed, the sign of the dihedral angle calculated would be reversed. For the second distribution, the vectors will be 1 – 5 (rotation axis), 1 – 2/3, and 1 – 6/7.

6.6 clusters –calculate cluster size distributions.

For clusters command line to set up the input file **.CLUSTERS.dat** is as before: “**setup clusters** <filename>”. This produces a file which looks like this:-

```
mw73.CLUSTERS             Title of this file
fnameato    mw73.ato             Name of .ato file
nsize       1000             Maximum cluster size (max 1000)
nsumt       -1             Number of configurations already accumulated
ndist       3             Number of cluster distributions

cluster   1

atom1       C             Atom type 1 to define a bond
atom2       C             Atom type 2 to define a bond
rmin        2             Minimum distance for this bond
rmax        5.5             Maximum distance for this bond

cluster   2

atom1       O             Atom type 1 to define a bond
atom2       H             Atom type 2 to define a bond
rmin        1             Minimum distance for this bond
rmax        2.5             Maximum distance for this bond

cluster   3

atom1       OW             Atom type 1 to define a bond
atom2       HW             Atom type 2 to define a bond
rmin        1             Minimum distance for this bond
rmax        2.5             Maximum distance for this bond
q

Cluster   1: fraction containing 2 or more molecules   1.00000

Cluster   2: fraction containing 2 or more molecules   0.79762
Cluster   3: fraction containing 2 or more molecules   0.72222
```

This file sets up three cluster distributions for a mixture of methanol and water. The placing of a molecule in a cluster is determined from the separation of specified pairs of atoms. Thus for cluster 1 above the cluster is determined from the carbon atoms of the methanol molecules. Any two methanol molecules whose carbon atoms are determined to be between 2 and 5.5Å apart are said to be in the same cluster. All the molecules within the simulation bos which are also in the same cluster are tallied, and used to generate a distribution of cluster size, where the size is the number of molecules in the cluster. For cluster 2 it is still between methanol molecules, but now the decision on clustering is based on the separation of the hydrogen atom on one methanol molecule from the oxygen on another. Cluster 3 looks at the water cluster, using a criterion based on the separation of the water hydrogen atom from water oxygen atom to determine if two molecules are bonded.

The program can be run by typing "**clusters** <filename>". Because cluster size distributions have a power law dependence on size it is useful to use log-log scales when plotting these distributions.

As of 2010 it is possible to specify multiple atom types and distances to decide whether two molecules are bonded or not. This would be useful in the above for example if you wanted to include *either* C – C distances *or* O – H distances between neighbouring molecules to decide if they were bonded or not. Thus the input would look like:

```
cluster   1

atom1      C O           Atom type 1 to define a bond
atom2      C H           Atom type 2 to define a bond
rmin      2 1          Minimum distance for this bond
rmax       5.5 2.5          Maximum distance for this bond
```

Note that it is imperative that the number of entries on each line are the same, otherwise the program will give an error message and not run. This feature is useful if you have a complex molecule that has several distinct possible bonding sites with different names. If those possible sites are specified, then any one of the sets of values that are satisfied between two molecules, then the molecules are regarded as bonded.

6.7 chains – calculate chain length distributions.

Compared to **clusters** the **chains** program uses a more advanced criterion for deciding whether two molecules are bonded or not. This time 3 atom types have to be specified, the first 2 being on the same molecule, and there are 3 distance ranges (6 numbers) to be specified, as well as an included angle 2-1-3 range. In order to be classified as "bonded" the 1st and 2nd atoms, the 1st and 3rd atoms and the 2nd and 3rd atom each have to be within the specified distance ranges, and the included angle 2-1-3 has to be within the specified angle range.

The program calculates 2 distributions for each chain defnition. The first is the distribution of bond numbers, i.e. how many bonds are there per valid molecule (in mixtures some molecules for example may not fit the criteria for bonding so cannot be included in the calculation). The second is the chain length distribution. The chain length is calculated using the "shortest path" criterion [12], which basically sorts through all the possible linked paths between two molecules, and counts only the shortest path, i.e. the one with the least number of linkages between those two molecules. Each chain must begin and end on a molecule with only one link to another molecule.

To set up the input file we type "**setup chains** <filename>" in the usual way, and the program can be run with the command "**chains** <filename>. The **.CHAINS.dat** file looks like this:-

```
methanol.CHAINS             Title of this file
fnameato    methanol1.ato              Name of .ato file
nsize       200              Maximum chain length (max 200)
nsumt       -1              Number of configurations already accumulated
ndist       1              Number of chain length distributions

chain   1

atom1       O             Atom type 1 for first atom in first molecule
atom2       H             Atom type 2 for second atom in first molecule
atom3       O             Atom type 3 for atom in second molecule
rmin12      0.5             Minimum distance for atom1-atom2
rmax12      1.2             Maximum distance for atom1-atom2
rmin13      1              Minimum distance for atom1-atom3
rmax13      3.5             Maximum distance for atom1-atom3
rmin23      1              Minimum distance for atom2-atom3
rmax23      2.5             Maximum distance for atom2-atom3
ang213min   0.0             Minimum included angle for atom2-atom1-atom3
ang213max   30.0             Maximum included angle for atom2-atom1-atom3q
q
  1  0.21160E+01  0.10838E+01  0.20485E+02  0.54776E+01
```

When it has finished the program produces at the last line the average number of bonds per molecule and its standard deviation, and the average chain length and its standard deviation, as shown in this example for methanol. The corresponding distributions of these quantities are given in the **.n01** file produced when the program finishes, columns 2 and 4 respectively, for each set of chain criteria specified. Log scales may sometimes be useful when plotting these distributions because of the rapid decline of the chain distribution with chain length. Note that unlike **clusters**, **chains** does not normalise the distributions to the total number of chains in the distribution.

As for **clusters**, it is now possible to have several sets of atoms to define a bond between molecules. In the above example, if we wanted to include *either* O-H …O bonds *or* C-M…O bonds (for example) we would write

```
chain   1

atom1       O C           Atom type 1 for first atom in first molecule
atom2       H M           Atom type 2 for second atom in first molecule
atom3       O O           Atom type 3 for atom in second molecule
rmin12      0.5 0.6           Minimum distance for atom1-atom2
rmax12      1.2 1.3           Maximum distance for atom1-atom2
rmin13      1 2           Minimum distance for atom1-atom3
rmax13      3.4 4.5           Maximum distance for atom1-atom3
rmin23      1 1           Minimum distance for atom2-atom3
rmax23      2.5 3.5           Maximum distance for atom2-atom3
```
etc.

6.8 rings – calculate ring length distributions.

rings works in a manner entirely analogous to **chains**, and the input files are identical. Here is an example from an ethanol-water simulation:

```
eth4wat6.RINGS             Title of this file
fnameato    eth4wat6.ato             Name of .ato file
nsize       25             Maximum ring length (max 25)
nsumt       -1              Number of configurations already accumulated
ndist       2              Number of ring distributions

ring   1
```

```
atom1      O           Atom type 1 for first atom in first molecule
atom2      H           Atom type 2 for second atom in first molecule
atom3      O           Atom type 3 for atom in second molecule
rmin12     0.5            Minimum distance for atom1-atom2
rmax12      1.2            Maximum distance for atom1-atom2
rmin13     1            Minimum distance for atom1-atom3
rmax13      3.5            Maximum distance for atom1-atom3
rmin23     1            Minimum distance for atom2-atom3
rmax23      2.5            Maximum distance for atom2-atom3
ang213min   0.0            Minimum included angle for atom2-atom1-atom3
angw13max   30.0              Maximum included angle for atom2-atom1-atom3q

ring   2

atom1      OW           Atom type 1 for first atom in first molecule
atom2      HW           Atom type 2 for second atom in first molecule
atom3      OW           Atom type 3 for atom in second molecule
rmin13     0.5            Minimum distance for atom1-atom3
rmax13      1.2            Maximum distance for atom1-atom3
rmin13     1            Minimum distance for atom1-atom3
rmax13      3.5            Maximum distance for atom1-atom3
rmin23     1            Minimum distance for atom2-atom3
rmax23      2.5            Maximum distance for atom2-atom3
ang213min   0.0            Minimum included angle for atom2-atom1-atom3
angw13max   30.0              Maximum included angle for atom2-atom1-atom3q
q
   1  0.28182E+00  0.55811E+00  0.00000E+00  0.00000E+00
   2  0.11673E+01  0.12494E+01  0.64068E+01  0.36644E+01
```

The only difference between the two routines is that a ring can only begin and end on a molecule with two or more linkages. Again ring lengths are counted using the shortest path criterion – the number of rings associated with a particular molecule could be huge, but there will be only a few or one which satisfies the shortest path. At the end of running the program the input file lists the average number of linkages per molecule (called bonds in the program – to be distinguished from the intra-molecular distances also called bonds) for each distribution and the average length of shortest path rings found for those linkages. In the above example the average length of ethanol-ethanol rings found was zero, while the average length of water-water rings was 6.4 molecules per ring. The distribution of the number of neighbours and the distribution of rings as a function of number of neighbours and ring length will be given in the corresponding **.n01** file, columns 2 and 4 respectively, for each set of **rings** criteria specified. **rings** does not normalise these distributions to the total number of linkages or the total number of rings.

Once again as for **clusters** and **chains** it is possible to have more than one set of definitions for bonds for a given ring definition. This is helpful when you have (say) a network glass with network modifiers and you want to include both base atoms and dopant atoms in the definition of a bond.

6.9 voids – calculate void distributions and the void radial distribution function.

voids is a relatively recent addition to the EPSR suite and is still being understood and tested. Basically the idea is to estimate how much unoccupied space there is in a simulation box, and then calculate the radial distribution function of that unoccupied space. To do this it is necessary to divide the box into pixels – currently a maximum of 100 along each Cartesian axis is allowed, giving a total of 10^6 pixels in all. Each pixel is then assigned a status, occupied or empty, depending on whether there are ANY atoms within the specified distance (**radius)** of it. The ratio of number of unoccupied pixels to total number of pixels gives the fraction of unoccupied pixels listed in the example **.VOIDS.dat** input file below.

For each unoccupied pixel a search is made of the 26 surrounding pixels (assuming they are placed on a cubic grid). If any of these neighbouring pixels is also unoccupied, both pixels are assigned to the same void number. Once all the unoccupied pixels have been tested this way, the whole process is repeated, until the number of distinct voids detected does not change. This typically takes a few iterations. From here the size of each void is calculated by adding up the number of unoccupied pixels in each void.

For each void distribution, there are three outputs in the voids distribution file, <filename>.**VOIDS.n01**. The first is simply the void size distribution function as a function of void radius, similar to that given by **clusters**, **chains**, or **rings**.

The second is the total void-void radial distribution function, which is calculated by looking at each void pixel in turn and calculating the radial distribution of void pixels around it. The third is the same as the second, except that it only includes pixels that are not in the same void. This third distribution does not appear to be overly useful at this stage, since any given void can already have a complex structure with multiple length scales, so it doesn't make a lot of sense to separate the two void distribution functions in this way.

The input file for this calculation looks like the following:-

```
pccp.VOIDS              Title of this file
fnameato    pccp.ato              Name of .ato file
nsize       1000              No. of histogram points (max 1000)
npix        100              No. of pixels for void calculation (max 201)
voidmax     300              NOT USED BY CURRENT PROGRAM
nsumt       -1              Number of configurations already accumulated
ndist       3              Number of void distributions

void   1

radius      2.2              Average distance from atoms to define a void
exclude     HW              List of atom types to exclude (max 10)

void   2

radius      2.0              Average distance from atoms to define a void
exclude     HW              List of atom types to exclude (max 10)

void   3

radius      1.8              Average distance from atoms to define a void
exclude     HW              List of atom types to exclude (max 10)
q

Distribution   1: fraction of unoccupied pixels =    0.09477

Distribution   2: fraction of unoccupied pixels =    0.18309

Distribution   3: fraction of unoccupied pixels =    0.28555
```

npix is the number of pixels along half the side of the simulation box and is limited to 100. The program actually adjusts the precise number to give the optimum representation of the specified sphere, i.e. so that the representation of the sphere by a set of cubes gives the correct volume as close as is practical

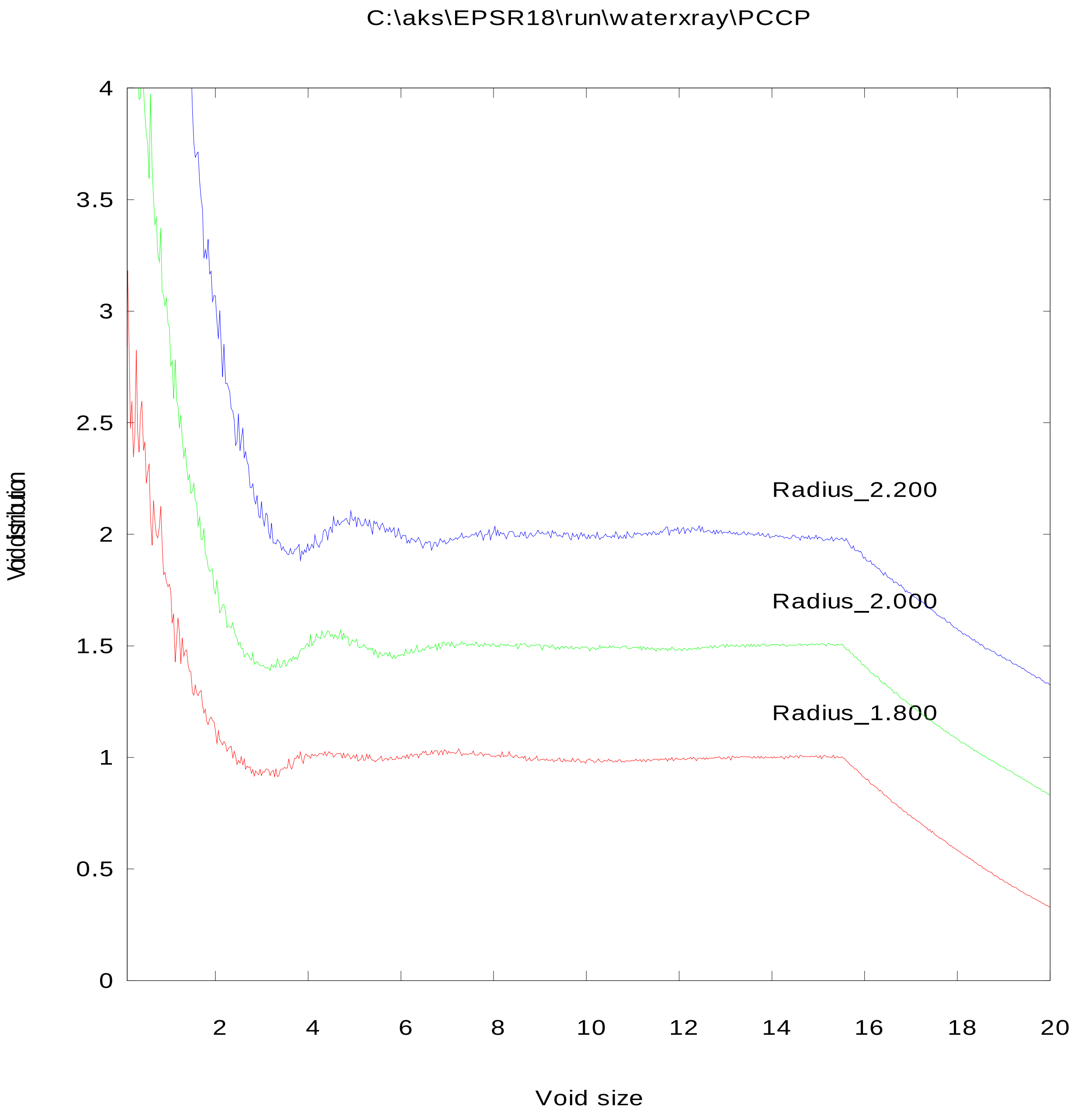
voidmax is not used in the current version of the program

nsumt works in the usual way, i.e. it determines how many distributions are accumulated. Set to -1 it only gives the current distribution.

radius sets the distance from any atom to test whether a pixel is occupied or not.

exclude allows you to exclude any atom types listed (separated by spaces if more than one). If you don’t want to exclude any atoms, leave it blank or type XX so that it will never find this atom (it doesn’t check to see whether the specified exists in the file or not).

The calculated total void radial distribution function which looks like this:-



The sharp down turn at high *r* arises because of the limit of half the box size being reached – there is no correction for the uniform atom distribution in this routine at present.

It is not totally clear what this distribution represents at present. The sharp rise at low *r* can be understood because at short distances any void pixel will likely see a much higher density of other void pixels in its immediate vicinity than at longer distances. The subsequent structure clearly indicates density fluctuations in the simulation box, but the fact that the position of these depends on the chosen void radius makes it difficult to assign any definite meaning to these fluctuations.

6.10 fluctuations – calculate the number fluctuations in boxes of specified size.

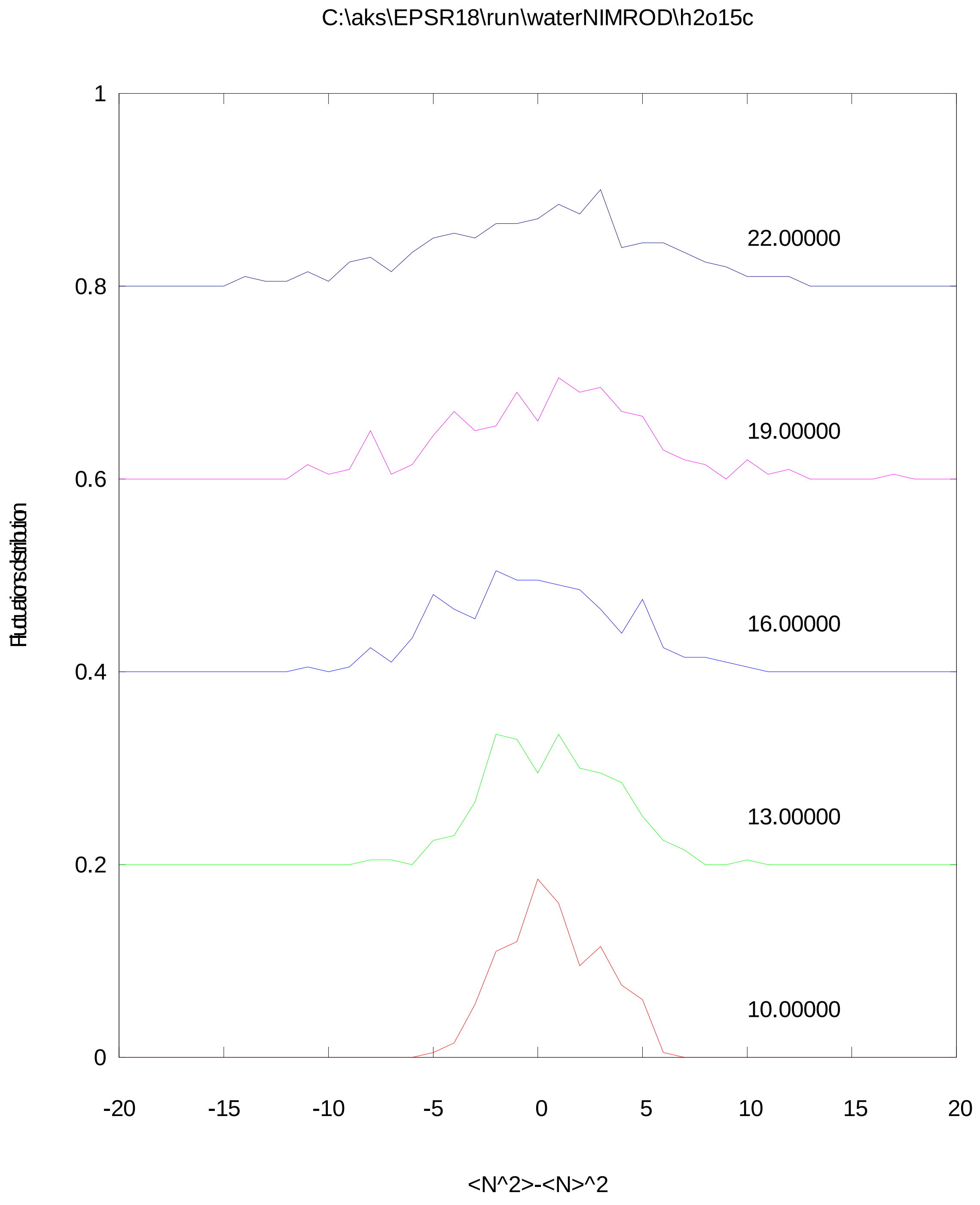
It is sometimes forgotten that disordered materials can have significant density fluctuations on the atomic length scale. In a monatomic material the density fluctuations give rise to the compressibility, which for condensed matter is quite small – this is the reason why the structure factor normally becomes small at low *Q*. However these are the density fluctuations over the macroscopic length scale, while on the scale of a few atoms the fluctuations can be quite large. **fluctuations** allows you to calculate these number fluctuations for a series of box sizes of specified dimensions. The input file looks like the following:

```
h2o22cnew.FLUCTUATIONS          Title of this file
fnameato    h2o22c.ato          Name of .ato file
nsumt       782000              Number of samples already accumulated
nsamples    1000                Number of samples to take from this configuration
ndist       5                   Number of volume sizes

Size   1

rmax        10.                 Radius of volume to probe
ntotal      33.3352013          Sum of N
nsqtotal    1117.47422          Sum of N^2

Size   2

rmax        12.                 Radius of volume to probe
ntotal      57.5960041          Sum of N
nsqtotal    3326.84795          Sum of N^2

Size   3

rmax        14.                 Radius of volume to probe
ntotal      91.4685127          Sum of N
nsqtotal    8380.01318          Sum of N^2

Size   4

rmax        16.                 Radius of volume to probe
ntotal      136.527722          Sum of N
nsqtotal    18658.0763          Sum of N^2

Size   5

rmax        18.                 Radius of volume to probe
ntotal      194.403794          Sum of N
nsqtotal    37816.1337          Sum of N^2
q
  10.  33.3352013  1117.47422  0.187146753
  12.  57.5960041  3326.84795  0.165779829
  14.  91.4685127  8380.01318  0.147858053
  16.  136.527722  18658.0763  0.13372609
  18.  194.403794  37816.1337  0.119846761
```

The only numbers you need to worry about (apart from the name of the **.ato** file of course) are the number of distributions, **ndist**, and the dimension of each box, **rmax**. Obviously set **nsumt** to zero at the beginning. Note that despite the name, **rmax**, corresponds to the dimension of a cubic box to be selected at random from within the simulation box. No check is made that this dimension is smaller than the simulation box, however it should be typically not more than about ¾ the size of the simulation box, otherwise the fluctuations will be meaningless, since obviously if you chose a dimension the same size as the simulation box, then the fluctuations would go to zero.

At the end of the file, the program lists for each specified box the value of $S(0)=\frac{\langle N^2\rangle-\langle N\rangle^2}{\langle N\rangle}$, which would go to the isothermal compressibility limit, $\rho k_B T \chi_T$, in the limit of an infinite box.

In addition the program will generate the distribution of these fluctuations for each specified box size (this example was just one configuration – run it for longer to get the distributions smoother):-



6.11 writexyz – writes a file of atomic coordinates in xyz format.

Sometimes it is useful to store the coordinates of individual atomic configurations. This can be done with the **writexyz** command. The command requires as arguments the name of the **.ato** file which is to be written out, and qualifier ‘append’ or ‘noappend’ which tells it whether to append the coordinates to the list of configurations in a single file, or to write each configuration to a separate file. The default is ‘append’ if this is not specified.

If ‘append’ is specified, then the name of the output file is ‘<ato-filename>append.xyz’. Each set of coordinates will appear as a separate block of data with a blank line between each block. The first entry in a block is the number of atoms in that block, and the second entry is the filename from which the coordinates were derived, followed by the block number. Subsequent entries are atomic symbol and (x,y,z) coordinates for each atom, one line per atom.

If ‘noappend’ is specified, then the files are numbered sequentially, ‘<ato-filename>00001.xyz’, ‘<ato-filename>00002.xyz’, etc., up to a maximum of 99999 files. After that an error will likely occur and the filenames will all be called ‘<ato-filename>*****.xyz’. The counter for these files is stored in a file called ‘<ato-filename>.nblock’. The format of these files is the same as the first block of the append file.

After the command ‘writexyz <ato-filename> append/noappend’ it is necessary to say what you actually want to write out for each block of data. The format here is the same as with plotato with the Jmol option, and all values can be entered in a single command line so that command can be entered as part of a shell script. After the first two arguments described above, you choose whether you want to write out all the atoms (0) or a subset of the box, starting from a specified atom (atom number). As with **plotato** you then specify the x, y, z-minus and z-plus dimensions of the box you want to output (4 arguments). Then you specify the orientation of the plot, using three Euler angles, (phi, theta, chi). Note that, as with **plotato**, this rotation is performed BEFORE outputting the x, y, and z values defined by the boundaries specified in the previous step, so you can orientate your set atoms to coincide with the axes of a particular molecule. This particular option is probably only useful when using **writexyz** interactively. Finally you have the option of selecting all the atom types that exist within the specified box (0) or listing those types (expressed as numbers) which you want to exclude from the box. Note however that for the molecule at the origin ALL the atoms in that molecule are shown, irrespective of whether they have been excluded or not.

If specifying all these values on a single command line the ‘append’ or ‘noappend’ argument MUST be given, otherwise an error will occur.

Some examples of how to use the command in a shell script are given below. The example **.ato** file, abcd.ato, is supposed to be a mixture of Ab (type 1) and Cd (type 2) atoms.

a) **writexyz** abcd append 0 10 10 -5 5 0 0 0 0

This will write all the atomic coordinates to a file called abcdappend.xyz. Each subsequent call to this command will append the coordinates to the same file. The atoms will be limited to a region of ±10Å along each of the x and y axes and ±5Å along the z axis. There will be no rotation of the box prior to plotting, and all the atoms within the plotting range will be included.

b) **writexyz** abcd noappend 0 10 10 -5 5 0 0 0 0

This is the same as a), but the configuration is written to files called abcd00001.xyz, abcd00002.xyz, etc., each time the call to writexyz is made.

c) **writexyz** abcd append 0 10 10 -5 5 0 0 0 1

This is the same as a), except that only the Cd atoms will be written out (type 1 = Ab are excluded).

d) **writexyz** abcd append 0 10 10 -5 5 0 0 0 1 2

This is the same as a), except that no atoms will be written out, since there are only two types of atom in this file, and both are excluded.

e) **writexyz** abcd noappend 300 10 10 -10 10 0 0 0 0

This will write a separate xyz file each time it is called. The atoms printed will be relative to the centre of mass of molecule 300 and within a range of ±10Å of this molecule along each coordinate axis (cubic plotting box). There will be no rotation and All the atoms that satisfy this constraint will be listed.

f) **writexyz** abcd append 250 8.0 8.0 -8.0 8.0 0 0 0 2

This will write an ‘append’ file containing only the Ab atoms which are within ±8Å along each coordinate axis with molecule 250 at the origin (Cd atoms, type 2, are excluded).

6.12 Building your own analysis routine.

It is now much easier to add your own routines to EPSRshell. Basically you need a file with the name <program_name>.**symbols** which will define the symbols you wish to access through **setup**. This file must exist in the area where you run EPSR from. In practice this could be quite simple, such as specifying an **.ato** file and another file which would contain the input to your program, or it could be more sophisticated as in the above examples. Using the **setup** menu means you can easily define, save and change variables that you wish to access in your program, but of course you may prefer to use your own input file. The symbols file might look like the following:-

```
extension .EXAMPLE.dat
prompt setup example>

symbol fnameato
symbol_default <undefined>
symbol_type  c
symbol_text  The ato file to process

symbol atom1
symbol_default XXX
symbol_type c
symbol_text Atom type 1

symbol atom2
symbol_default XXX
symbol_type c
symbol_text Atom type 2

symbol rmin
symbol_default 2.0
symbol_type r
symbol_text Minimum distance to be considered bound

symbol rmax
symbol_default 2.0
symbol_type r
symbol_text Maximum distance to be considered bound
```

This would apply to a program called **example**. The corresponding input file for this program might be:-

```
benzene.EXAMPLE              Title of this file
fnameato    benzpure.ato              The ato file to process
atom1       M              Atom type 1
atom2       C              Atom type 2
rmin        2.0            Minimum distance to be considered bound
rmax        3              Maximum distance to be considered bound
q
```

In order for this to work, the program you write will need to be linked with the EPSR link libraries (which of course will need to be compiled for your platform, but normally this is not difficult). Doing this means you have full access to the EPSR methods for reading and writing the **.ato** file and also accessing the atomic coordinates, and so on. Below is an extract of what this program might look like:-

```
      program standalone
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
c$$$  This is a toy example of an a program to analyse the ato file from
c$$$  EPSR. This example shows how to read and write an input file (the
c$$$  test.EXAMPLE.dat file), what variables are available when the ato
c$$$  file is read using the routines in EPSR.
c$$$
c$$$  This example calculates the fraction of "bound" bromide ions in an
c$$$  ato containing surfactant molecules in solution with bromide
c$$$  counterions. An ion is considered bound if it is within a certain
c$$$  distance of the headgroup of the surfactant molecule -- here the
c$$$  distance is calculated from the nitrogen atom in the headgroup.
c$$$
c$$$  Program structure:
c$$$
c$$$  1. parse the command line arguments
c$$$
c$$$  2. setup "symbols" for the example EXAMPLE.dat file
c$$$
c$$$  3. read that file and get variables from the .dat file
c$$$
c$$$  4. using the variables made available loop through the atoms in
c$$$  the ato file
c$$$
c$$$  5. upon finding the bromide ion, calculate the distance to the
c$$$  nitrogen atom (taking care of periodic boundary conditions), if
c$$$  there is at least one nitrogen within rmax of the bromide record
c$$$  it as bound
c$$$
c$$$  6. calculate the fraction of bound bromide ions and write this to
c$$$  the .dat file
c$$$
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

c     Get the command from the command line arguments
      call get_command_from_args()

c     Run the command

      call run_example()

      stop
      end
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      subroutine run_example()
      include 'epsr_dimension.inc'
      include 'commands.inc'
      include 'directory_name.inc'
      include 'system_commands.inc'
      include 'symbols.inc'
      include 'molecom.inc'
      include 'clustercom.inc'
          character*20 text

      integer*4 iBr,iN,ierrout,it,nin,nout,ia1,ia2,im1,im2,
     $     j, br_count, br_bound(matom), isum, ssg_ierr
      real*4 xdif,ydif,zdif,dsq12,da12,x1,y1,z1,x2,y2,z2,nbound,
     $     br_rmin,br_rmax
      character*3 atom1,atom2

      character*256 symbolsfname

      write(6,*) "Variables from get_command_from_args():"
      write(6,*) "home_directory:", home_directory
      write(6,*) "directory_name:", directory_name



      ssg_ierr=0
      symbolsfname="example"
      call setup_symbols_generic(symbolsfname, ssg_ierr)

      if (ssg_ierr.ne.0) then
         write(6, *) "Problem setting up symbols",
     $        " from .symbols file. Exiting example."
         return
      endif

      write(6, *) "Used example.symbols to set up symbols"

c     now setup the input file -- the .dat file...
      nin=11
      nout=6
      call setup_input_file(nin,nout,ierrout)

c     if an error was encountered on setup, exit now
      if(ierrout.ne.0) return

c     read the input file (the .dat file)
      call read_input_file(nin,nout,ierrout)

c     if an error was encountered on read, exit now
      if(ierrout.ne.0) return

c$$$  Get ato file name, the minimum and maximum cutoff distances to
c$$$  use from the symbols_value array which was read from the .dat
c$$$  file
c$$$
c$$$  the ato file name:
      iref = 1
      write(6,*) "atofname :", symbol_value(iref)
      iref = iref +1
100   format(a3)
      read(symbol_value(iref), 100) atom1
      iref = iref +1
      read(symbol_value(iref), 100) atom2
      iref = iref +1
      read(symbol_value(iref), *) br_rmin
      iref = iref +1
      read(symbol_value(iref), *) br_rmax
      write(6,*) "rmin :", br_rmin
      write(6,*) "rmax :", br_rmax
```

Your code goes here

Then finish up with:

```
c
c If any of the input variables have changed you need to assign them to their respective
c symbol_value before exitting, otherwise they will not be saved:

              icount=0
              text='atom1'
              call search_for_symbol(text,icount)
              iref=ireff(icount,1)
              write(symbol_value(iref),*) atom2
              iref=iref+1
              write(symbol_value(iref),*) atom1

      call save_files_others(11)

      close(11)

      return
      end
```

If you need advice on how to do this please do ask. The link libraries are distributed with the EPSR binaries. Note that it is possible to have multiple values on a single line: there is a parsing routine available for disentangling the separate values.

7 Spherical harmonic representation of many-body correlation functions

7.1 Introduction – the spatial density function and orientational correlation function

Once you get away from the cosy world of pair distribution functions, you run into two problems. Firstly defining any function in 2 or 3 or more dimensions requires increasingly large numbers of pixels, so there is a problem of how to store the functions. Secondly you need some method of visualizing the output so that it is more than simply a huge array of numbers. The spherical harmonic representation is an extremely compact method of storing many-body correlation functions, with the added bonus that having calculated the coefficients you can go back and interrogate the distribution in various ways without having to recalculate the atom positions all over again. Fortunately once you have made the mental leap and started to think in 3D or more, you discover a world rich in detail and subtlety that you would never have dreamed existed before! Nature reveals its secrets reluctantly, but when it does so be prepared for a shock!

The following account is intended to try to help you make that leap. It is not easy to get your head around these functions, but well worth the effort. The spatial density function (term originally due to Svishchev and Kusalik [13] I believe) is perhaps the easiest concept to think about. Basically you have an entity at the origin. This entity can be anything, e.g. a molecule, or group of atoms in a particular conformation. It could be a single atom but this would not be interesting as you can only have radial information around a single atom. The entity has to be sufficiently well defined that a set of coordinate axes can be defined relative to it, so that typically it will need at least 2 atoms, preferably at least 3. The question you want answered is how are other atoms or entities situated with respect to this central entity? Maybe Fig 7.1 can help you to see the problem:

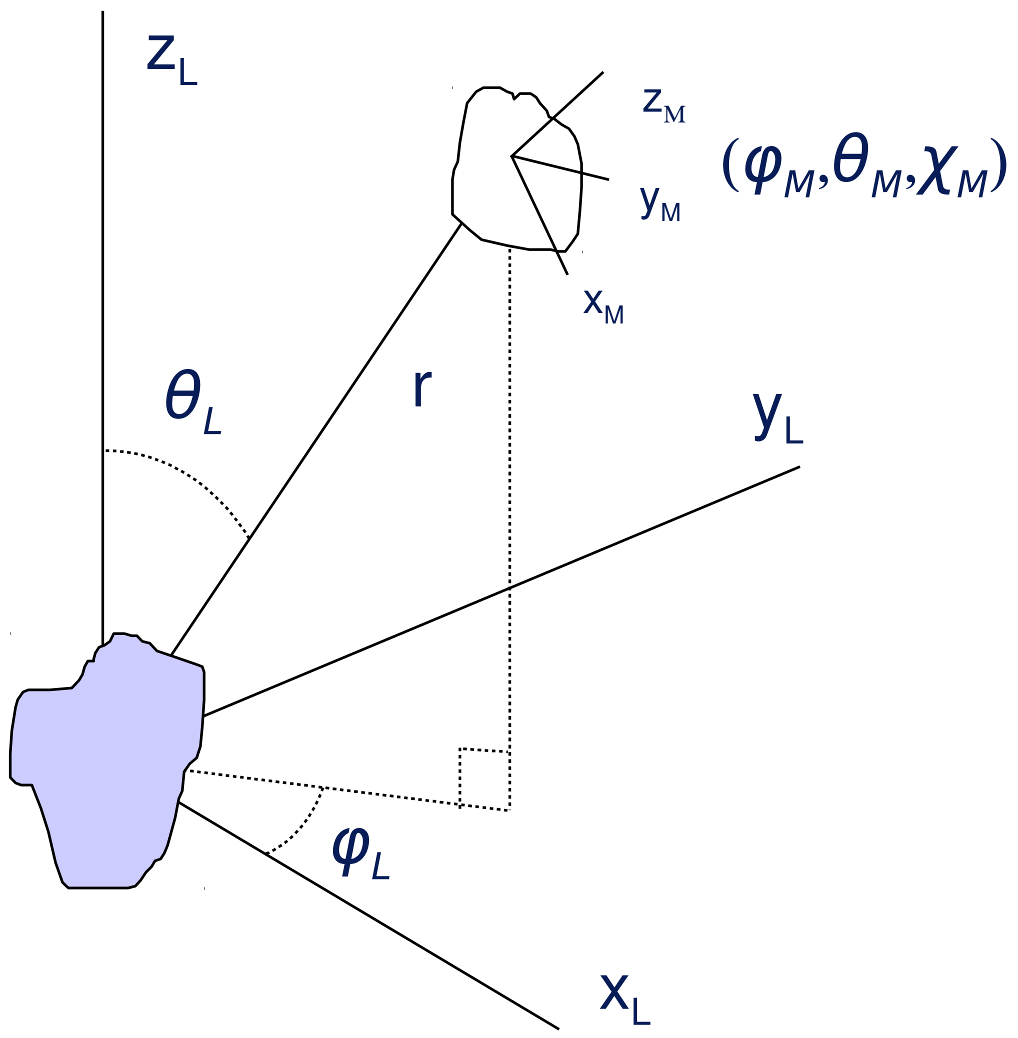


Figure 7.1

Here we see a molecule at the origin of the coordinate system at some particular orientation, given by the Euler angles $\boldsymbol{\omega}_1 = \varphi_1\theta_1\chi_1$ [4] The question is how is the second entity, at orientation $\boldsymbol{\omega}_2 = \varphi_2\theta_2\chi_2$ distributed with respect to the first, i.e. what is the density of 2nd entities as a function of **r**? This is called the “spatial density function” or SDF for short. Or we could equally ask the question how is the orientation of the second entity distributed as a function of r for a given $(\theta_1\varphi_1\chi_1)$? This second quantity is called the “orientational correlation function” or OCF.

The spherical harmonic expansion of these quantities can be written in the form

4 A description of Euler angles can be found in a number of textbooks. The definitions used here are based on *Theory of Molecular Fluids Volume 1 – Fundamentals*, C G Gray and K E Gubbins, Oxford University Press, 1984, which also gives an excellent account of the spherical harmonic functions. The order of the rotations being used here to get to the final orientation is important. The entity is first rotated by an amount φ about the initial *z*-axis, then by an amount θ about the new *y*-axis, finally by an amount χ about the revised *z*-axis that is generated by the second rotation. All rotations are in the direction of a clockwise screw along the positive axis. They can also be performed in reverse order but rotating about the (fixed) laboratory axes throughout.

$$g(r,\omega_1,\omega_2)=1+h(r,\omega_1,\omega_2)$$
$$=1+\sum_{l_1 l_2 l}\sum_{m_1 m_2 m}\sum_{n_1 n_2} h(l_1 l_2 l; n_1 n_2; r)\, C(l_1 l_2 l; m_1 m_2 m)\, D^{l_1}_{m_1 n_1}(\omega_1)^* D^{l_2}_{m_2 n_2}(\omega_2)^* D^{l}_{m0}(\omega) \qquad (7.1.1)$$

where $C(l_1 l_2 l; m_1 m_2 m)$ are the Clebsch-Gordan coefficients, which constrain the values $|l_1-l_2|\le l\le l_1+l_2$ and $m = m_1+m_2$, and

$$D^l_{mn}(\omega)=e^{-im\phi} d^l_{mn}(\theta)\, e^{-in\chi} \qquad (7.1.2)$$

are the generalized spherical harmonic functions, with

$$d^l_{\mathrm{mn}}(\theta)=\left[(l+m)!(l-m)!(l+n)!(l-n)!\right]^{\frac{1}{2}} \frac{\sum_k (-1)^k (\cos\theta/2)^{2l+m-n-2k} (\sin\theta/2)^{2k-m+n}}{(l+m-k)!(l-n-k)!k!(k-m+n)!} \qquad (7.1.3).$$

Note that the summation in (7.1.1) differs slightly from that in Gray and Gubbins (eq. 3.142) by a factor of $\left[\frac{(2l+1)}{4\pi}\right]^{12}$ due to the use of generalized spherical harmonics throughout. These generalized spherical harmonics are of course orthogonal with respect to integrals over *ω*, and in particular we note the special case

$$D^l_{mn}(000)=\delta_{mn} \qquad (7.1.4).$$

The coefficients of this series can be calculated from a simulated distribution function by inverting (7.1.1), making use of the closure rules of the Clebsch-Gordan coefficients and the orthogonality of the generalized spherical harmonics:-

$$h(l_1 l_2 l; n_1 n_2; r)=\frac{(2l_1+1)(2l_2+1)}{4\pi(8\pi^2)^2}$$
$$\times\int d\omega_1\int d\omega_2\int d\omega \sum_{m_1 m_2 m} h(\mathbf{r},\omega_1\omega_2)\, C(l_1 l_2 l; m_1 m_2 m)\, D^{l_1}_{m_1 n_1}(\omega_1)\, D^{l_2}_{m_2 n_2}(\omega_2)\, D^{l}_{mn}(\omega)^* \qquad (7.1.5)$$

where, in the case of the simulation, the integrals are actually performed as averages over all relevant pairs of molecules in the simulation box. (Note that another factor of (*2l+1*) from integrating over *ω* does not appear in this expression because it is cancelled by the sum over products of Clebsch-Gordan coefficients when inverting 7.1.1).

This calculation of the coefficients is performed by the routines **sharm** and **sdf**. Note that when the molecules are identical (e.g. water about water) then certain symmetry rules apply to the coefficients $h(l_1 l_2 l; n_1 n_2; r)$, (see Gray and Gubbins) which means the number of terms to be summed is reduced. Thus in the input files to these programs it is important to specify whether the molecules are identical or not. Fortunately the EPSRshell **setup** command checks which molecules have been specified and automatically detects whether they are identical or not.

In order to reconstruct the spatial density function or orientational correlation function from the spherical harmonic coefficients the quantity to calculate is (7.1.1). This function is in general a function of 9 variables, but 3 of these are not independent, since the spatial density function should look the same relative to the molecule at the origin, whatever the orientation of that molecule. Therefore an immediate simplification can be made to this function, without loss of generality, by rotating the laboratory axes to coincide with the coordinate axes of the central molecule, thereby setting $\omega_1 = (000)$. This immediately invokes (7.1.4) so that $m_1 = n_1$. Writing $\omega_2 \equiv \omega_1\omega_M$, i.e. rotation ω_1 followed by rotation ω_M, where $\omega_M = \varphi_M\theta_M\chi_M$ is the *relative* orientation of entity 2 with respect to entity 1, and $\omega = \omega_1\omega_L$, where $\omega_L = (\theta_L,\varphi_L)$ is the position vector which describes where the second molecule is *relative* to the one at the origin, a simplified version of 7.1.1 is formed, now a function of only 6 coordinates:-

$$g(r_L,\omega_M)=1+\sum_{l_1 l_2 l}\sum_{m_2 m}\sum_{n_1 n_2} h(l_1 l_2 l;n_1 n_2;r)\,C(l_1 l_2 l;n_1 m_2 m)\,D^{l_2}_{m_2 n_2}(\omega_M)\,D^{l}_{m0}(\omega_L) \qquad (7.1.6)$$

This result was written down here by inspection of (7.1.1) but in fact can be demonstrated quite formally by making use of the properties of the *D* coefficients under rotations.

There are still too many dimensions here to plot – the maximum number that can be accommodated is perhaps 3 dimensions at best, so some further simplification is necessary. One of the most useful simplifications is to sum only the terms for which $l_2=m_2=n_2=0$. This removes any dependence of the result on the orientation of the second entity, i.e. the function is plotted after averaging over the orientations, ω_M. In that case $l=l_1$ and $m=n_1$ from the properties of the Clebsch-Gordan coefficients, so that this function simply maps out the distribution of 2nd entities, averaged over the orientations of the latter, with respect to the entity at the origin. This is what Kusalik and Svishchev[13] called the spatial density function.

Another possibility is to set $m_2=n_2=0$, but allow l_2 to adopt the values specified in the coefficients. In this case we remove any dependence of the result on (φ_M,χ_M), but leave the dependence of θ_M, so this would correspond to the distribution of dipole orientations if the *z*- axis of the second entity were to lie parallel to its dipole moment axis. In this latter case it is necessary to fix $(\varphi_L\theta_L)$, in order to be able to plot this orientational distribution in 3D. Such an orientational plot would correspond to an orientational correlation function, and a number of such plots could be envisaged. The point is that by choosing to fix the different values of l_1, l_2, n_1, m_2, and n_2 one can control which particular distribution functions are plotted. Note that when plotting orientations of anything more complicated than linear molecules, it will always be necessary to average over one degree of freedom of the molecule, since there are typically 3 orientational degrees of freedom, plus the radial coordinate, making 4 in total, but we can sensibly plot only 3 of those degrees of freedom.

7.2 sharm – calculates the spherical harmonic coefficients for the spatial density function and orientational pair correlation function

The input file for **sharm** is set up in the usual way, “setup sharm <filename>”. A typical input, which lists all the variables is given below:-

```
h2o298tot.SHARM              Title of this file
fnameato    h2o298tot_s.ato              Name of .ato file
nr          130              Number of radius values (max 200)
rmax        13               Maximum radius for spherical harmonic coefficients
nsumt       1                Number of configurations already accumulated
ncoeffs     158              Number of coefficients (program calculates this)
l1values    0 1 2 3 4            L1 values (separated by spaces)
l2values    0 1 2 3 4            L2 values (separated by spaces)
lvalues     0 1 2 3 4            L values (separated by spaces)
```

```
n1step     2            Step in N1 values
n2step     2            Step in N2 values
atom-c     OW           Central molecule - list of centre atom types
axisc1     z 2 3        First axis definition for central molecule
axisc2     y 2          Second axis definition for central molecule
atom-s     OW           Second molecule - list of centre atom types
axiss1     z 2 3        First axis definition for second molecule
axiss2     y 2          Second axis definition for second molecule
q
```

When setting up this input file you need to be aware of a few things that may not be obvious. **n1step** and **n2step** are the rotational symmetry of the molecules (1 and 2 respectively) being defined. So if the central molecule is defined as an NH_3 group [i.e. -z axis is C-N and x or y axis is an H], then **n1step** would be 3 (as the C-N-H group has 3 fold symmetry irrespective of the symmetry of any other part of the molecule attached to the NH_3 group as it has not been included in the calculation), and if the second molecule is a water molecule (and you intend to plot the orientations of the second molecule), then **n2step** would be 2 (as it has 2 fold symmetry). If the orientations of the second molecule are not going to be plotted, then **n2step** can be left at 0. The variables **atom-c** and **atom-s** are used to define the centres of the central and secondary molecules respectively. The program expects a list of atom types in each case and this can be a single atom atom type or it could be several, separated by spaces. The program takes the arithmetic mean of all the positions of atoms of these types to define the centre of each molecule.

The molecular axes are defined via the **axisc1** and **axisc2** variables (for the central molecule) and via the **axiss1** and **axiss2** variables (for the secondary molecule). Each variable consists of a letter which defines which axis is being set up, and a set of atom numbers for the molecule in question which will be used to define the specified axis. For the first axis (z in this example) the specified axis is assumed to run from the centre of the molecule to the mid-point of the specified atoms. (Several atoms can be specified.) For the second axis it may not be possible to assign a set of atoms which lie along the specified axis, so instead a vector is drawn from the centre of the molecule to the point defined by the set of specified atoms, and the second axis is assumed to lie in the *plane* defined by the this vector and the first axis: its precise direction is determined from the requirement that it must be orthogonal to the first axis. Hence in the above example atom 2 on the water molecule is set to lie in the z-y plane of the molecule, which allows the y-axis to be set up as orthogonal to the z-axis.

In **sharm** it is only necessary to define two of the Cartesian axes, since the third axes can always be found from the first two. It does not particularly matter which order the axes are entered, but you should be aware that currently **sharm** only estimates the REAL coefficients in equation (7.1.1). This means the molecule as defined MUST have at least one plane of mirror symmetry and at least one of the mirror symmetry planes must be coincident with the *z-x* plane. If such a plane does not exist in the real molecule, then mirror symmetry about the *z-x* plane will be imposed on the estimated distribution functions, and it likely they could be misleading.

Note that **sharm** will only actually calculate the coefficients every 5 times it is called. This is because calculation of the coefficients can be time consuming if there are a lot of them, and one needs to be sure that before adding to an existing accumulation of coefficients the simulation box has moved to a completely new region of phase space before calculating them again. Hence the number of accumulations of the coefficients is actually given by **nsumt**/5. **nsumt** is listed in the **sharm** input file.

Note also that the option **nsumt** = -1 does not exist for **sharm**. The idea is that we would not want to spend computer time calculating coefficients when a simulation has not reached equilibrium or the EP is still being adjusted.

The program is run in the usual way “sharm <filename>”. The individual coefficients, **.SHARM.h01**, etc.**,** can be plotted from the **plot** menu, and the spatial density functions shown via the **plot2d** and **plot3d** commands.

7.3 sdf – spatial density function and orientational correlation function for an arbitrary system

As it stands **sharm** is written specifically for molecules. However a simple extension of the ideas there can be used to generate a spherical harmonic representation of the triple body correlation function or higher order correlation functions. The basic idea is that while **sharm** restricts you to specifying atoms on the same molecule, there is no intrinsic reason why this has to be so. For example we could use *any* pair of atoms within a specified distance range to define an origin and *z*-axis and then look at the density of 3rd atoms of a specified type as a function of (r,θ_L). This would give a 2-D map of the triple body correlation for a pair of atoms at the origin of the specified separation.

A further extension of this idea is to define a “molecule” at the origin to be *any* geometry of atoms which we may wish to construct, and plot the distribution of the same or other geometries of atoms as a function of position or orientation, just as we did in **sharm**. For example in amorphous silica, one could envisage our central “molecule” being a Si atom with two of the neighbouring O atoms at a specified distance to define the *z* and *y* axes, in an analogous way to what was done for water with **sharm** in the previous section. The spatial density of other SiO_2 molecules could then be plot in an entirely analogous way. The only real distinction is the extra task of defining which sets of SiO_2 triplets can be categorised as “molecules”. The routine to do this is called **sdf.**

Below is shown a typical input to the **sdf** routine. This can be set up in the usual way by typing

setup sdf <file name>:-

```
sio2_095.SDF            Title of this file
fnameato    sio2_095.ato             Name of .ato file
nr         130              Number of radius values (max 200)
rmax       13               Maximum radius for spherical harmonic coefficients
nsumt      1               Number of configurations already accumulated
ncoeffs    16               Number of coefficients (program calculates this)
l1values   0 1 2 3 4 5 6             L1 values (separated by spaces)
l2values   0               L2 values (separated by spaces)
lvalues    0 1 2 3 4 5 6             L values (separated by spaces)
n1step     2               Step in N1 values
n2step     0               Step in N2 values
atom-c     Si O O          List of centre atom types. First will be origin.
pairdist-c  1.6 1.6 2.6     List of pair distances for centre molecule.
fracdist-c  0.2             Acceptance half-width (fraction of pair distance).
axisc1     z 2 3            First axis definition for centre molecule.
axisc2     y 2              Second axis definition for centre molecule.
atom-s     O                List of second atom types. First will be origin.
pairdist-s  0               List of pair distances for second molecule.
fracdist-s  0.2             Acceptance half-width (fraction of pair distance).
axiss1     z               First axis definition for second molecule.
axiss2     y               Second axis definition for second molecule.
q
```

This input file will calculate the distribution of O atoms about a central SiO_2 molecule. The theory behind **sdf** is identical to that of **sharm**. The only difference is the way the “molecules” are specified. Hence the first few lines of the input file down to **n2step** are identical to **sharm**, with exactly the same discussion about the meaning of values.

However for **atom-c** we now have a list of all the atoms that are to make up the central molecule. Only the **first** of these atoms will form the origin of the coordinate system. The other will be used to define the molecule. In the above case we have three atoms to define the molecule, a central Si accompanied by 2 O atoms.

The distances needed to define the "molecule" are given in the next line, **pairdist-c**. This makes a list of the expected pair distances in the "molecule". They are given in the order atom1 – atom2, atom1 – atom3, and finally atom2 – atom3. Hence they are given here as each O should be 1.6Å from the central Si, with the two Os separated by a distance of 2.6Å.

Of course in the real box the chances of finding three atoms which satisfy these requirements exactly is slim, so we allow some range of distances to be included. This is defined by the value of **fracdist-c**, and is expressed as a fractional width of the respective bond distance, 0.2 in the present instance. Hence according to this criterion, any Si-O distance in the range 1.6 ± 0.32Å would be regarded as bonded. Equally any O-O distance in the range 2.6 ± 0.52Å would be regarded as bonded.

Having thus defined the molecule, the following two lines define the axes as in **sharm**, in exactly the same way. Hence in this example the molecular *z* axis will pass centrally between atoms 2 and 3, i.e. the two oxygen atoms, while the y-axis will lie in the plane defined the *z* axis and atom 2 (also oxygen).

The axes of the second molecule are defined in exactly the same way as the first. In this example we are simply plotting the distribution of oxygen atoms around the SiO_2 molecule, so there are no axes to define.

All the comments about definition of axes, symmetry and the use of **nsumt** that were given for **sharm** above apply equally to **sdf** at the present time.

The program is run in the usual way "sdf <filename>". The individual coefficients, **.SDF.h01**, etc**.,** can be plotted from the **plot** menu, and the spatial density functions shown via the **plot2d** and **plot3d** commands.

7.4 plot2d and plot3d– plotting the results from 7.2 and 7.3

N.B. The programs **plot2d** and **plot3d** have been superseded by **splot2d** and **plot3djmol**. The inputs to these new programs are closely analogous to **plot2d** and **plot3d**, but the newer routines do not require access to the PGPLOT library, which is difficult to provide access to on all platforms. Hence the newer programs should now be used instead. Support for the PGPLOT routines **plot2d** and **plot3d** is not guaranteed from now on.

As usual these programs are run from EPSRshell, and they require input files that need to be setup using **setup.** As their names imply these two programs allow you to plot the results of the spherical harmonic calculation in 2D or 3D respectively. The 2D plot is essentially a surface contour, and is plotted as a simple contour map. They are based on the PGPLOT routines, together with Devinder Sivia's PGXTAL routines. The present implementation of calling these routines from within EPSRshell seems to avoid many of the previous problems with getting them operational. The output is normally to a **.gif** file called **pgplot.gif**, however if the system command, system_pgout, read in from system_commands.txt, is set to /cps, then postscript output is produced instead, **pgplot.ps**. There is no way of getting a screen plot using this implementation of PGXTAL with pgplot on Windows.

On LINUX the story is different, since full implementation of PGPLOT is usually available for LINUX. Hence the full range of PGPLOT output options should be available, including /XS which should produce an x-screen with the plot on it.

To setup the input files you need to type "**setup plot2d**" or "**setup plot3d**".

They can be run from the command prompt with the command “**plot2d** <filename> or “**plot3d** <filename>” as usual. Here is an example of the **plot3d** input file, called ‘sio2_095.plot3d.txt’, used to plot the coefficients generated in the example given in the previous section on **sdf**:-

```
sio2_095.SDF.h01
16             no. of coefficients - determined from coefficients file
1            = 0 for identical molecules, else 1 if different
1            0 sets first coefficient to zero - normally 1
4            number of smoothings on coefficients
6            maximum radius of plotting box
1 1             no. of plots along x- and y-axis [set at 1 1]
1.0             aspect ratio of plot [1.0]
1 6             minimum and maximum radius of plot
0.2             fractional isosurface level (-ve for absolute)
1 0             use l1 and l2 (1 or 0)
1 0             use n1 and n2 (1 or 0)
0            use m (1 or 0)
1            vary (thetal, phil) (1), (thetam, phim) (2), (thetam, chim) (3)
0 0 0
3            number of spheres at centre of plot (max 25)
1.3 0.0 0 0 0.9 0.9 0.9          sphere radius, (r,theta,phi), (r,g,b colour indices)
1.0 1.6 54 90 1.0 0 0          sphere radius, (r,theta,phi), (r,g,b colour indices)
1.0 1.6 54 270 1 0 0          sphere radius, (r,theta,phi), (r,g,b colour indices)
1.5 2 1             axes character size, line width and colour (separated by spaces)

0.1 -1.3 2             (x,y) coords. of title, and character size (separated by spaces)

0.8 0.8 1.0             red green blue fractions for background (separated by spaces)
-1             ishade (1-8): 0 means no shading, -ve means inverted shading
1 1 0             red green blue fractions for object (separated by spaces)
2 2 0             (x,y,z) coordinates for light source (separated by spaces)
1.0             fade factor (0 = no fading, 1=full fading)
2            transparency of object (0=0%,1=25%,2=50%,3=75%)
1.0 0.0 1.0 1.0             difuse, shine, polish and contrast
25 15             rotation and elevation of viewing point (deg.)
0            extra lines (0) - cannot be set
0            extra text (0) - cannot be set
.SDF.h01
```

Note the very last line which can be used to determine whether .SHARM or .SDF coefficients will be searched for if ‘search’ is typed opposite the **shcoeffs** variable in **setup.**

To understand which l1,l2, l, n1 and n2 values to set you will need to understand a little about the spherical harmonic representation of the orientational pair correlation function, as given in Section 7.1. For **plot2d** the corresponding input file, called ‘sio2_095.plot2d.txt’for the same coefficients is slightly different:-

```
1            Background colour index [1]
sio2_095.SDF.h01
16             no. of SHARM coefficients - determined from coefficients file
1            = 0 for identical molecules, else 1 if different
1            0 sets first coefficient to zero - normally 1
4            number of smoothings on coefficients
16             x dimension of plotting square
16             y dimension of plotting square
2.0             minimum radius of plot
3            Number of circles at centre of plot (max 25)
0.0 0 0 0.3
0.0 1.31 0.92 0.3
0.0 -1.31 0.92 0.3
```

```
1 1           no. of plots along x- and y-axis [set at 1 1]
1.0           aspect ratio of plot [1.0]
1 0           use l1 and l2 (1 or 0)
1 0           use n1 and n2 (1 or 0)
0           use m2 (1 or 0)
1           vary thetal (1), phil (2), phim(3), thetam(4), chim(5)
90 0 0 0
1.5 2 0           axes character size, line width and colour (separated by spaces)

0.1 -1.3 2           (x,y) coords. of title, and character size (separated by spaces)
3           phim, thetam, chim (degrees - separated by spaces)
1 5           range of intensites to be plotted (separated by spaces)
1.0           Contrast factor [0.0 - 1.0]
N
0           extra lines (0) - cannot be set
0           extra text (0) - cannot be set
.SDF.h01
```

This is because you are now taking a slice out of the 3D plot and showing the variation in a particular plane. Hence for the case where **nvary** = 1 as above, the plot has been chosen to show the variation in z-y plane (phil = 90) of the central molecule. The corresponding outputs from these two input files are shown in Figures 7.2 and 7.3.

The commands **plot2d** and **plot3d** are like some of the other EPSRshell routines, specifically **plot, plotato**, in that the program that generates the plots is external to the shell itself. The above '.plot2d.txt' and '.plot3d.txt' files are input files for the routines defined by the variables **system_plot2d** and **system_plot3d** which are read in from the 'system_commands.txt' when EPSRshell starts up. Typically these variables are set to the executables named map2dplot.exe and map3dplotquad.exe which are stored in the EPSR binaries folder. Hence either of these routines can be run from outside the shell, using the plot2d and plot3d .txt files as input. Particularly for **map2dplot** this gives you access to several other features of these programs that are not available from within the shell.

In addition there is the opportunity to plot the 3D graphs without axes showing (see the "noaxes" versions of the programs) and also if orientational correlation functions are being plotted, to rotate the central molecule to the most probable orientation (see the "rot" versions of the map3d program).

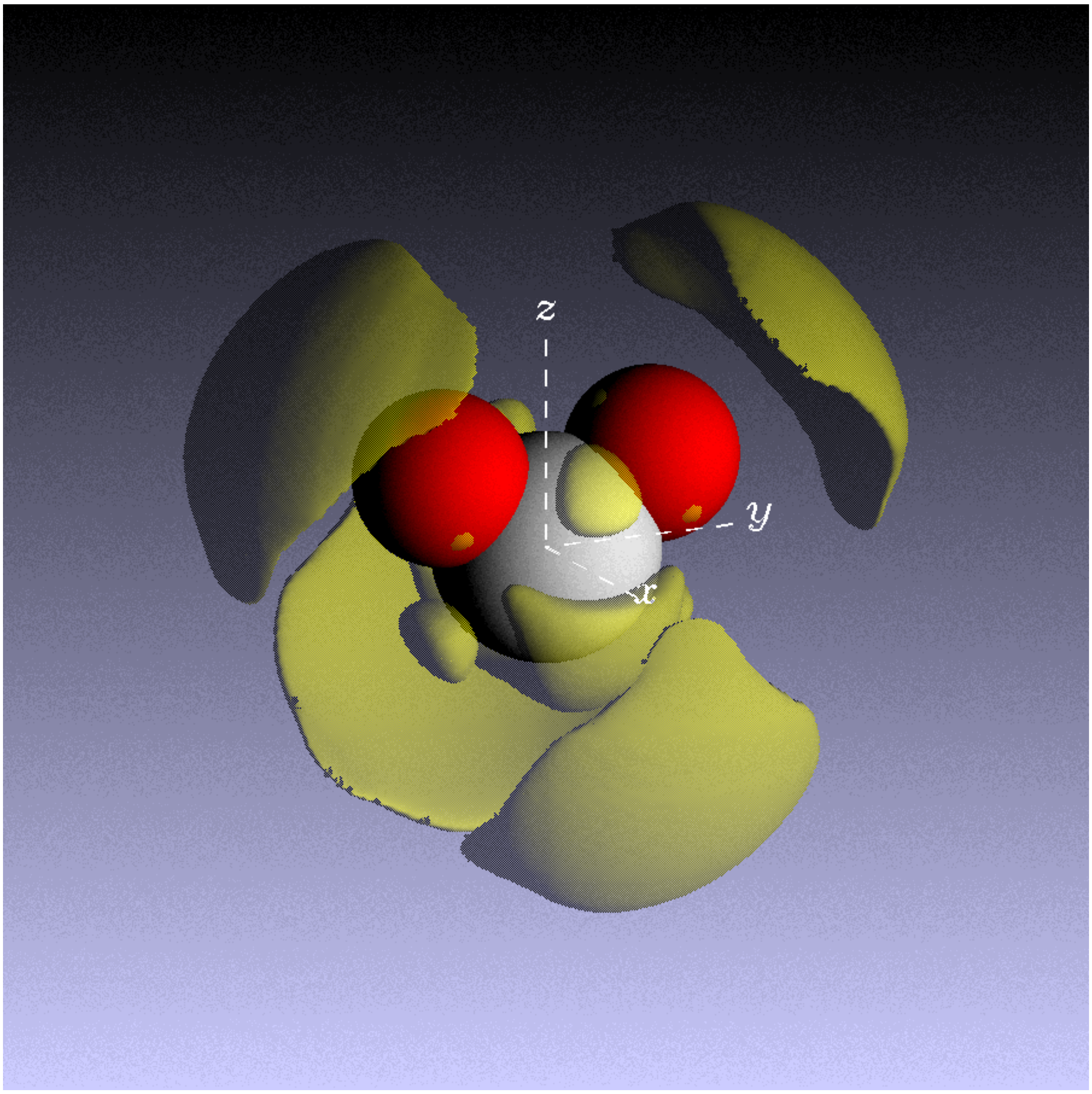


Figure 7.2 Result of running **sdf** on an EPSR simulation of amorphous silica, then plotting the results via **plot3d**. The central “molecule” of SiO_2 is shown as the spheres near the origin of the coordinate system. The local, tetrahedral coordination is very visible in these plots. Perhaps not so obvious is the fact that the second coordination shell also shows strong tetrahedral coordination.

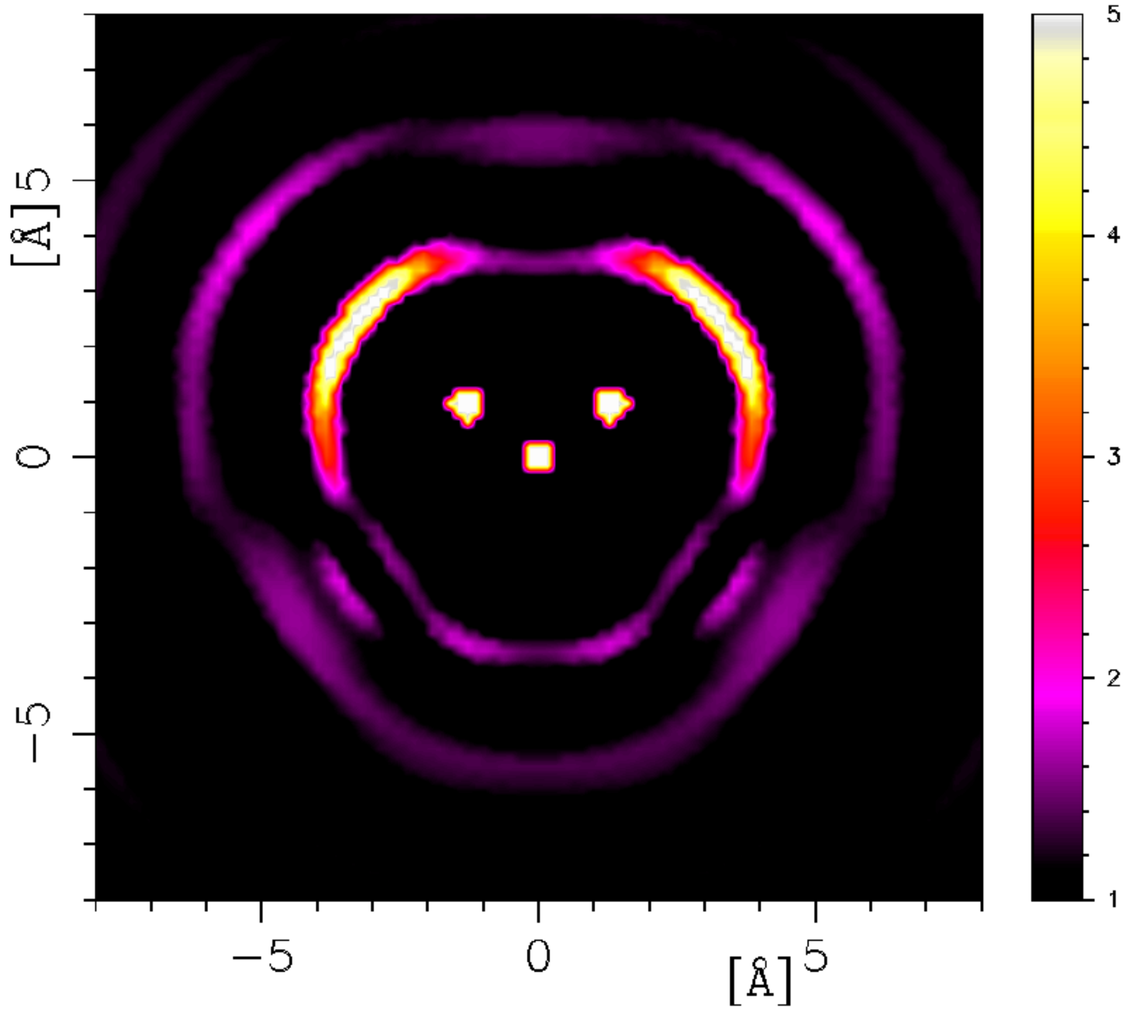


Figure 7.2 Result of running **sdf** on an EPSR simulation of amorphous silica, then plotting the results via **plot2d**. The central “molecule” of SiO_2 is shown as the circles near the origin of the coordinate system. Notice how we get better intensity information from the contour plot, but lose its 3-dimensional aspect. A number of other 2D surface plot options are available with **plot2d**, not all of which can be accessed from EPSRshell.

8. Some examples and exercises

The following exercises are purely a suggestion so please feel free to substitute anything that you would prefer to attempt. The idea was simply to allow the opportunity for the user to get the “feel” of the programs before attempting any real data. The idea is that in each case you would set up the **.ato** file, perhaps make a mixture, make the **.ato** bigger, run **fmole** and **randomise**, set up the **.wts**, **.inp** and **.pcof** files and run EPSR, initially without structure refinement, but then with it. Thanks to a summer student there are a few worked examples here to give a feel of how it runs from the point of view of a non-expert.

8.1 Single component Lennard-Jonesium**.**

Assume that the number density of the data supplied (single normalized structure factor) is 0.0334 atoms/Å^3

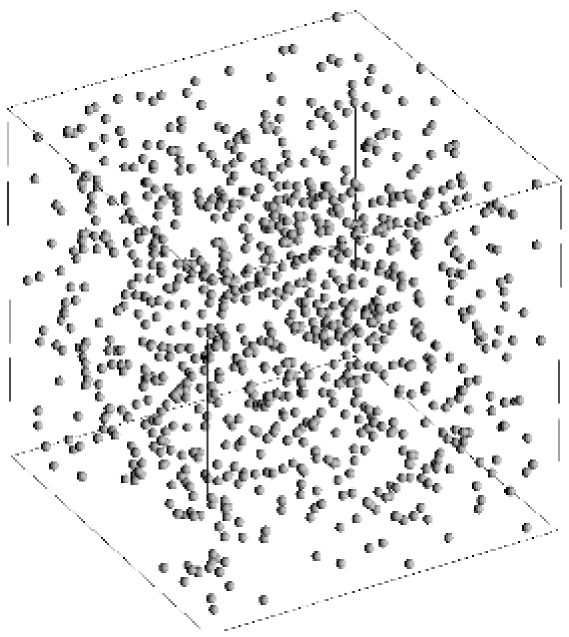
After loading EPSR, and getting to/creating the working folder needed, enter **makeato**. This will guide you through a series of inputs needed to create the atom or molecule you are creating. It will ask you to enter values for the epsilon and sigma. If you don’t have these, an educated guess will do. (Try a sigma of around 3, and epsilon of about 0.6, and a mass of 16.)

The program will now ask you for the coordinates of the atoms you have created, and their positions in relation to each other. As you have only created one atom, enter the value as 0 for everything that it asks for. The temperature comes next, and as before, this value is entirely up to you. Note: Standard room temperature is 298K.

The next value to input is the atomic number density. The **ecore** and **dcore** are used in another part of the program, that we will come onto later. For now, enter the values as 1 and 3 respectively, as shown in the brackets by each value on the command prompt.

Now you have created a single atom. To make this into a more realistic and accurate simulation, you need to create more. The simple way to do this is to mix this file with itself a set number of times. This effectively adds more atoms, to a set amount dictated by you. Using the command **mixato**, you can specify how many atoms you want in the mixture. A suitable number is between 1000 and 500, depending on your processor speed.

These atoms are now arranged in the same point in space, and so you need to spread them out randomly. Use the command **randomise** to space them out randomly. If you like, now is a suitable time to create an image file showing the distribution clearly. To do this, use the command **plotato**, and specify the options wanted. A suggested viewing angle is 30, 30.



The next step is to set up a **.wts** file. Make sure that you have the **.ato** and data files in the folder that you are working in, and enter the command ‘**setup nwts** <filename>’. Simply follow the instructions to complete this. When complete, type ‘**nwts** <filename>**.**

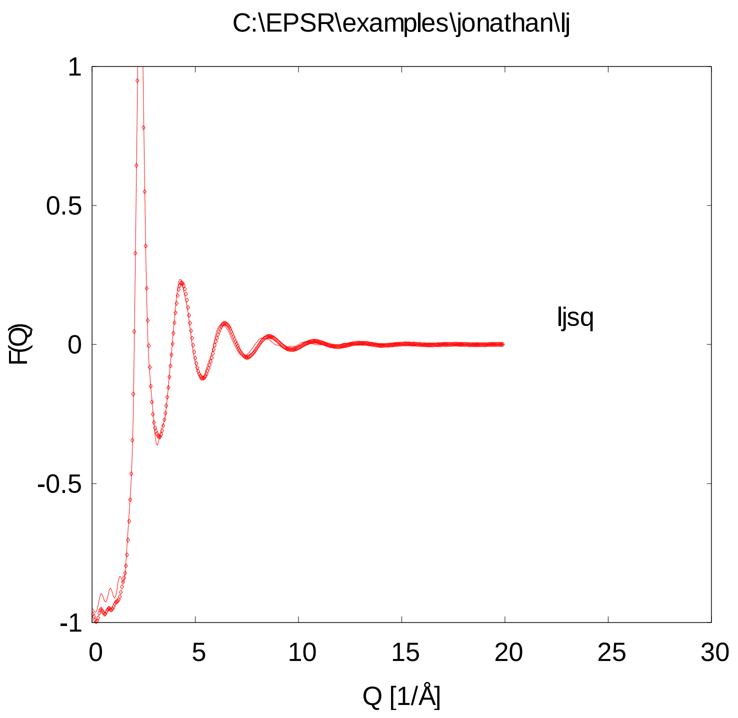
Use the ‘setup epsr’ command. All of the values already in place are correct, and the only thing needed for input is the undefined entries. These you will need to enter yourself. When it is all set up, save and exit. This will have created a .inp and .pcof file.

Next, a .txt file needs to be set up in the folder where EPSRshell is running. Name it runepsr.txt, and in it write the following lines:

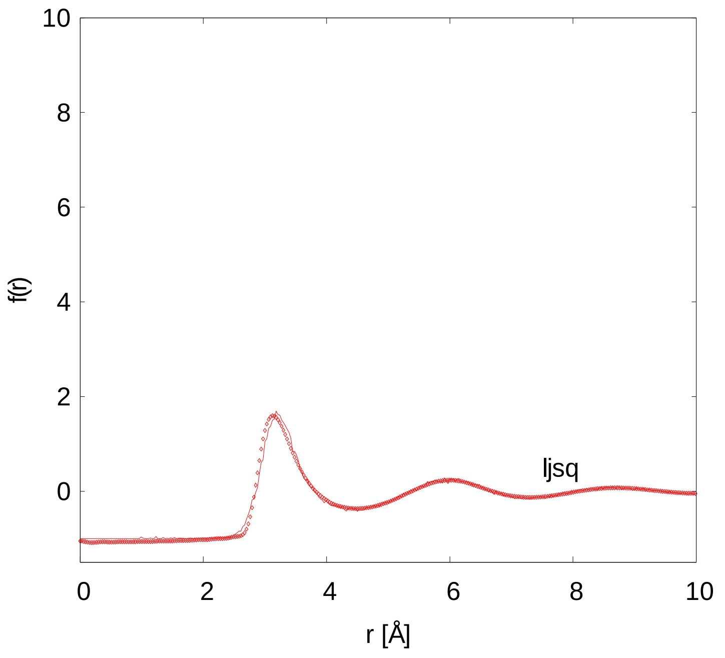
```
cd .\[name of folder data is in]
epsr [name of .inp file]
```

Start the simulation now, by entering ‘ss runepsr.txt’ into the command prompt. This will run EPSR. Open a new command prompt window, from the same folder as the one used for the first command prompt. To edit the data, type ‘ps’ into the command prompt, and use ‘changeato’ to change the values necessary. Typing ‘ss’ again in the window that was running the simulation will restart it.

When you are happy with the data, you can see the results on a graph. To do this type ‘plot’, then p (number) to specify the type of plot requested. (You will need to ensure you a copy of the file ‘plot_defaults.txt’ in your working folder before you can do this.) The graph below shows EPSR F(Q) Fit and data in Q-space.



In real space the graph should look something like this:



8.2 Two component charged Lennard-Jonesium – NaCl.

Aim: Using the synthetic data in the NACL folder perform an EPSR simulation on these data and review the results.

8.2.1 Using **makeato** to make two .ATO files, one called **na.ato** the other called **cl.ato**. The number density is 0.032071 atoms/Å^3, and Lennard-Jones parameters can be 0.514 kJ/mole and 2.290Å for Na and 0.566 kJ/mole and 4.191Å for Cl. The respective atomic masses are 23 and 35.44, and you should assume the charges are zero.

8.2.2 Using **mixato**, make a mixture of 500 Na atoms with 500 Cl atoms. Run **randomise** to randomise the atoms, run **fcluster** to generate a starting configuration of atoms. (If your processor seems to run slowly, use only 250 atoms of each – it will not make a great deal of difference to the results.)

8.2.3 Using the **.ato** file you have created as input create the **.wts** file using the **Nwts** program. (For this example pretend you have extracted the Na-Na, Na-Cl and Cl-Cl partial structure factors by chlorine 35-37 isotope substitution, so have extracted each partial structure factor separately. The mixture ratio for the 3rd sample required by **Nwts** can be 0.5.)

8.2.4 Using the setup command set up the EPSR **.inp** and **.pcof** files, with the appropriate **.ato, .wts**, and data files, and give them a suitable name, e.g. nacl.

8.2.5 In the home folder where EPSRshell is running create a batch file **runepsr.txt** with the following lines in it (this assumes the home folder is the examples folder from the CD:

```
cd .\NaCl
epsr nacl (or whatever name you gave the .inp file)
```

8.2.6 Start the EPSR simulation by typing “ss runepsr.txt”. It’s a good idea to set the priority for this routine to low if your operating system allows it so that the EPSR simulation does not interfere too much with other things you may want to do on your computer.

8.2.7 Meanwhile open another EPSRshell window in the SAME home folder where the simulation is running. Once the simulation has gone through a few iterations, plot some of the outputs to see what it looks like. If you decide you want to change something in the simulation, then pause it by typing ‘ps’. Then use **changeato** to change parameters in the **.ato** file. Finally restart the simulation by typing ‘ss’ in the window where the simulation was originally running.

8.2.8 Once you are convinced that you cannot improve the fit by changing the parameters of the reference potential on their own, then set the value of **potfac** to 1.0 so than the empirical potential comes into sway. You can adjust the influence of the EP by adjusting the value of **efacm** in the **.inp** file.

8.2.9 Once you are happy with the fits, set the accumulator **nsumt** positive, and accumulate some configurations. At the same time if you have time introduce some other calculations into the EPSR loop, such as **coord** or **triangles** or **partials.**

8.3 Amorphous silica

Although the amorphous silica refinement has been performed many times it is worth going through the exercise of starting from scratch, as in the previous example, i.e. generate an **.ato** file, run the simulation WITHOUT structure refinement and calculate the site-site radial distribution functions. See how far you can get WITHOUT potential refinement. HINT: it’s a good idea to run the simulation at a very high temperature, e.g. 10,000K until you are sure it is in equilibrium, otherwise you may spend a long time equilibrating at 300K.

The number density of the neutron data supplied is 0.06834 atoms/Å^3 and it HAS been normalized to the sum of the neutron weights (this is nrtype 3). Equally the x-ray data from Mozzi and Warren have been (re-)normalised to the single atom scattering.

8.4 Water.

Number density is 0.1002 atoms/Å^3. Total neutron differential cross section files are supplied (not normalized). These were run for pure H2O (sls18498.mdcs01), H2O:D2O 75:25 (sls18499.mdcs01), H2O:D2O 50:50 (sls18500.mdcs01), H2O:D2O 25:75 (sls18501.mdcs01), and pure D2O (sls18502.mdcs01). These are all **nrtype** = 5 datasets (Gudrun histogram format). Also included is an x-ray dataset obtained from [14]. These x-ray data have been normalised to the single atom scattering. Here are the basic steps:-

8.4.1 Set up a water molecule. To do this you first create a ‘water_template.mol’ file. Typical bond OH bond distance is ~0.98Å. Typical H-O-H angle is 104.5°. For the potential parameters you can set σ_O = 3.2Å and ε_O = 0.65 kJ/mole. The Lennard-Jones parameters for H can both be set to zero. (You might want to label the atoms OW and HW to make it clear these atoms belong to a water molecule.) The charges on the oxygen atom can be -1e and on each hydrogen atom +0.5e. Finally run **makemole** to create the ‘water_template.ato’ file.

8.4.2 Running **mixato** create a new **.ato** file called ‘water.ato’, using as the input file ‘water_template.ato’. Put just 1 molecule in this new file. Now run **fmole** on ‘water.ato’ for a few hundred iterations until the intramolecular energy is small and stable. View ‘water.ato’ with **splotato** or **plotato** – make sure it appears correctly.

8.4.3 Run **mixato** again, using ‘water.ato’ as the input file, and make a box containing say 500 or 1000 of these water molecules. The output can be to the same ‘water.ato’ that you read the molecule from in the first place.

8.4.4 Run **randomise** on this expanded ‘water.ato’ file – this randomises the molecular positions and orientations. Finally run **fmole** on this randomised box a large number of times, e.g. 1000 times. This is to ensure that the molecules are all truly different from one another. It will run faster if the neighbour list is updated less frequently, but will not be so accurate and give strange energies.

8.4.5 Prepare the **.wts** files. For the neutron data bear in mind that the hydrogen atoms will exchange with one another in the liquid. For the x-ray data you can try using different values of q_O and q_H for the modified atomic form factors (MAFFs) as discussed in Section 5.2 and then make sure these correspond to the partial charges used in the reference potential.

8.4.6 Setup EPSR for this simulation, using as input the box of water molecules you have created, the six diffraction datasets (or you could use a subset of these), and the corresponding **.wts** files.

8.4.7 Run the simulation once. Then set any minimum distances you think may need to be set. Ensure **potfac** is set to zero, then set the simulation running from a script file.

8.4.8 Start another version of **EPSRshell** in the same home folder, go to the working folder, make sure you have a ‘plot_defaults.txt’ file installed in this folder and try plotting the results. Appendix 9.3 shows the typical list of plot types.

8.4.9 From time to time when the simulation is at equilibrium, pause it, change some of the parameters, either of the reference potential (Lennard-Jones parameters) or the minimum distance parameters. See how far you can get with fitting the data without invoking the empirical potential.

8.4.10 Finally switch on the EP by setting **potfac** to 1. Start the simulation running.

8.4.11 Once equilibrium is reached, a number of other quantities can be calculated such as triangles distribution, ring and chain distribution functions, and of course the spherical harmonic coefficients. The results of the latter functions can be plotted using plot2d or plot3d.

8.5 Other examples

There are a number of other examples in the **examples** folder, the parameters required to run them are in a text file in the same directory containing the data files. In addition there are some worked examples in the **tutorials** folder

9. Appendices

9.1 Files you need to run **EPSRshell**

EPSR25distribution can be downloaded from the site **http://www.facebook.com/disord.matt** as a .zip file, **EPSR25distribution.zip.** Unzip this file in a suitable location, e.g. C:\ whereupon two folders should be created, **EPSR25distribution\EPSR** and **EPSR25distribution\EPSRfiles**. **EPSR** contains several additional folders, typically **bin**, **doc**, **examples**, **mol**, **run**, **startup**, **tutorials**. **EPSRfiles** contains the source code and make files to build **EPSRshell** and all its auxiliary routines. On Windows 7 (and hopefully Windows 8) you should not need to rebuild the binaries, but if you are running under Linux or OSX you will almost certainly need to re-compile all the binaries for your system. A **readme.txt** is provided in the **EPSRfiles** folder which explains how to do this. The routines can be compiled using the supplied makefiles but you may need to edit these files to have the correct paths and library names for your installation. In addition it is necessary to have all the appropriate PGPLOT and other graphics libraries in place before the 2- and 3-D plotting programs can be compiled and run. Three makefiles in the **EPSRfiles\newsrc\src** folder give the details of what is needed – makefile_windows, makefile_linux, makefile_mac for the corresponding operating systems respectively, but you will almost certainly need to edit these files before they will run correctly on your system.

The executables for EPSR are stored in the **bin** folder, The distribution includes a version of GNUplot for the Windows version, the PGPLOT libraries for Windows 7, and Jmol.jar, to run the Jmol program. folder tells you how to do this.

In order to ensure all the definitions are correct, the best option is to go to the **EPSR** folder and run the batch file, **setupepsr.bat** (for Linux and OSX equivalent is to type **sh setupepsr** in a terminal window). This only needs to be done once. This will set up **epsr25.bat** (**epsr25** for Linux/OSX) with all the required definitions for your system. Thereafter this file can be copied to each folder where EPSR is to be run so that running the **epsr25.bat** file in a folder (**./epsr25** for Linux) will start EPSRshell in that folder.

epsr25.bat:

```
set EPSRroot=C:\AlansStuff\EPSR\EPSR25distribution\EPSR
set EPSRbin=%EPSRroot%\bin
set EPSRstartup=%EPSRroot%\startup
set EPSRgnu=%EPSRbin%\gnuplot\binary
set PGPLOT_DIR=%EPSRbin%\PGPLOT\PGPLOTlib\
set PGPLOT_FONT=%EPSRbin%\PGPLOT\PGPLOT_LIB\grfont.dat
set epsrpath=%PATH%
set path=%PGPLOT_DIR%;%epsrpath%
set EPSRrun=%CD%
if not [%1]==[] (set EPSRrun=%1)
if not exist %EPSRrun% (mkdir %EPSRrun%)
echo %EPSRrun%
cd %EPSRrun%
copy %EPSRroot%\epsr.bat epsr.bat
if not exist plot_defaults.txt (copy %EPSRstartup%\plot_defaults.txt plot_defaults.txt)
copy %EPSRstartup%\system_commands_windows.txt system_commands.txt
copy %EPSRstartup%\f0_WaasKirf.dat f0_WaasKirf.dat
copy %EPSRstartup%\gnuatoms.txt gnuatoms.txt
copy %EPSRstartup%\gnubonds.txt gnubonds.txt
copy %EPSRstartup%\vanderWaalsRadii.txt vanderWaalsRadii.txt
title EPSR in %CD%
%EPSRbin%\epsrshell.exe
set path=%epsrpath%
copy gnuatoms.txt %EPSRstartup%\gnuatoms.txt
copy gnubonds.txt %EPSRstartup%\gnubonds.txt
copy vanderWaalsRadii.txt %EPSRstartup%\vanderWaalsRadii.txt
```

For Linux, the **epsr25** script looks like:-

```
export EPSRroot=/home/aks45/EPSR/EPSR25distribution/EPSR
export EPSRbin="$EPSRroot"'/bin'
export EPSRstartup="$EPSRroot"'/startup'
EPSRrun=$(pwd)
if [ $# -gt 0 ]
then
  EPSRrun=$1 ;
fi
# Check if the specified folder exists. If not create it in the current folder.
if [ ! -e "$EPSRrun" ]
then EPSRrun=$(pwd)/"$EPSRrun";
fi
#echo "$EPSRrun"
# Check if the specified folder exists. If not create it.
if [ ! -e "$EPSRrun" ]
then mkdir "$EPSRrun"
fi
cd "$EPSRrun"
# Copy the EPSR startup file
cp "$EPSRroot"/epsr epsr
chmod +x epsr
# Check that the specified folder has plot_defaults.txt. If not then copy a version in.
if [ ! -e plot_defaults.txt ]
then cp "$EPSRstartup"/plot_defaults.txt plot_defaults.txt
fi
cp "$EPSRstartup"/system_commands_linux.txt system_commands.txt
cp "$EPSRstartup"/gnuatoms.txt gnuatoms.txt
cp "$EPSRstartup"/gnubonds.txt gnubonds.txt
cp "$EPSRstartup"/f0_WaasKirf.dat f0_WaasKirf.dat
cp "$EPSRstartup"/vanderWaalsRadii.txt vanderWaalsRadii.txt
"$EPSRbin"/epsrshell
cp gnuatoms.txt "$EPSRstartup"/gnuatoms.txt
cp gnubonds.txt "$EPSRstartup"/gnubonds.txt
cp vanderWaalsRadii.txt "$EPSRstartup"/vanderWaalsRadii.txt
```

EPSR can also be started by typing ‘epsr <new folder name> (./epsr <new folder name> in Linux) in a command prompt, after first going the EPSR folder. This will start EPSR in the stated folder, creating a new folder if it does not exist, and copy all the required files, including **epsr25.bat** (or **epsr25** for Linux), into that folder.

Under LINUX you can place an alias in the .bashrc startup file

```
alias epsr=’sh <EPSRroot>/epsr $(pwd)’
```

where <EPSRroot> is the full path to the folder **EPSR25distribution,** so that when you type ‘epsr’ directly in a terminal window it will start **EPSRshell** in the folder where the terminal is open. The necessary files are then copied to that folder before **EPSRshell** begins.

For the shell to start correctly, it is necessary the file “**system_commands.txt**” is available in the directory where **EPSRshell** is to run. Again there are different versions for Windows and Linux, so the above **epsr** scripts copy the appropriate version for the operating system.

For Windows, the **system_commands_windows.txt** looks like the following:

```
system_ls  dir/b
system_cd  cd/d
system_md  md
system_edit  "c:\Program Files\Windows NT\Accessories\wordpad"
```

```
system_del  del
system_termination  \
system_join  &&
system_mopac  %EPSRbin%\MOPAC_7.exe
system_gnu  %EPSRgnu%\wgnuplot.exe
system_jmol  java -jar %EPSRbin%\Jmol.jar
system_binaries  %EPSRbin%
system_bin_ext  .exe
system_plot2d  map2d.exe
system_plot3d  map3drot.exe
system_plotato  pgplotato.exe
system_pgout  /GIF
gnu_pointtype  8
gnu_pointsize  0.3
```

For **system_edit** you can substitute your own editor if you prefer. **gedit** is a good editor for both Windows and Linux and is available for free on both systems. Options for **system_plot3d** include **map3d.exe**, **map3d_noaxes.exe**, **map3drot.exe**, **map3drot_noaxes.exe** (assuming they all exist in the **system_binaries** folder). The **rot** versions will allow the rotation of the central molecule to the most likely orientation when plotting the orientational correlation function. The **noaxes** versions do not display the coordinate axes.

while for Linux the **system_commands_linux.txt** looks like:

```
system_ls   ls -g
system_cd  cd
system_md  mkdir
system_edit  gedit
system_del  rm
system_termination  /
system_join  &&
system_mopac  run_mopac7
system_gnu  gnuplot
system_jmol  java -jar $EPSRbin/Jmol.jar
system_binaries $EPSRbin
system_bin_ext
system_plot2d  map2d
system_plot3d  map3drot
system_plotato  pgplotato
system_pgout  /png
gnu_pointtype  6
gnu_pointsize  0.4
```

Options for **system_plot3d** include **map3d**, **map3d_noaxes**, **map3drot**, **map3drot_noaxes**.

system_ls is the system command used to get a folder listing. It should be set up to produce a single column of entries. If multiple columns are produced, the EPSRshell routines which access file structure – **search** is the main one - will not work correctly.

system_cd is the system command to change directory.

system_md is the system command to make a directory.

system_edit is the system command to start an editor. You should ensure this editor is somewhere in the path used by the system to search for executable programs, or else specify the full path with the command.

system_del is the system command to delete a file.

system_termination is the character(s) used to signify the beginning or ending of folder names. In Windows this is '\'. In UNIX it is '/'.

system_mopac is the command to run Mopac. This will vary depending on how Mopac is installed on your system. On Ubuntu 14.04 it appears run_mopac7 if mopac7 is installed from the repositories. On OS X it will probably require the full path name to the binary.

system_join is the character(s) used to join system commands together on one line.

system_gnu is the command to start GNUplot.

system_jmol is the command to start Jmol.

system_binaries is the folder where the binaries are stored. It is shown here as an environment variable name, but can be set to any valid folder description.

system_bin_ext is needed to specify the extension **.exe** for Windows 8 and can be used in the same way for Windows 7. For other operating systems it should be left blank, unless the binaries have been created with a particular extension.

system_plot2d is the path and filename for the **plot2d** executable.

system_plot3d is the path and filename for the **plot3d** executable.

system_plotato is the path and filename for the **plotato** executable.

system_pgout is the output file type to be used by the PGPLOT routines.

system_quotes is used by Windows to pass folder and file names containing spaces. For non-Windows based systems it should be left blank.

gnu_pointtype is the character type used to plot points in GNUplot plots

gnu_pointsize is the size of the character used to plot points in GNUplot

NOTE: The above versions of these scripts and files are different from those used with earlier versions of EPSR but hopefully mean the initial startup is straightforward.

Files for the **home** folder where EPSRshell is to be started:-

epsr.bat or epsr	As above
system_commands.txt	As above
f0_WaasKirf.dat	Supplies information on the x-ray form factors
gnuatoms.txt	
gnubonds.txt	
plot_defaults.txt	Stores the various plot options
run.txt	Optional – used as a script file for running EPSR and the auxiliary routines. It can have any name you choose

If used with the **epsr25.bat** or **epsr25** scripts in the EPSR folder these will be copied from the **startup** folder before **EPSRshell** starts.

1. A K Soper, *Chem. Phys*. **202**, 295-306 (1996); A K Soper, *Chem. Phys.*, **258**, 121-137 (2000); A K Soper, *Mol. Phys.*, **99**, 1503-1516 (2001); F Bruni, M A Ricci, and A K Soper, in Conference Proceedings Vol 76, *Francesco Paolo Ricci: His Legacy and Future Perspectives of Neutron Scattering*, M Nardone and M A Ricci (Eds.) (Società Italiana di Fisica, Bologna, 2001); A K Soper, *Phys. Rev. B*, **72**, 104204 (2005).
2. D A Keen and R L McGreevy, *Nature*, **344**, 423-425 (1990)
3. M P Allen and D J Tildesley, "Computer Simulation of Liquids", (Oxford University Press, 1987)
4. D Frenkel and B Smit, "Understanding Molecular Simulation: From Algorithms to Applications", (Academic Press, 1996)
5. Y.-G. Chen, J. D. Weeks, *Proc. Nat. Acad. Sci U.S.A.*, **103**, 7560-7565 (2006)
6. R. Buckingham, Proc. Roy. Soc. A Math. Phys. Sci., **168**, 264-283 (1938)
7. F. London, *Trans. Faraday Soc.*, **33**, 8b-26, (1937)
8. V. Molinero, E B Moore, *J. Phys. Chem. B*, **113**, 4008-4016 (2009)
9. F H Stillinger, T A Weber, *Phys. Rev. B*, **31**, 5262, (1985)
10. G Hura, J M Sorensen, R M Glaeser, T Head-Gordon, *J Chem Phys*, **113**, 9140 (2000); J M Sorensen, G Hura, R M Glaeser, T Head-Gordon, *J Chem Phys*, **113**, 9149 (2000).
11. A K Soper, K Page and A Llobet, *J. Phys. Condens. Matter*, **25**, 454219 (2013)
12. D S Franzblau, Phys Rev B, **44**, p4925-4930 (1991)
13. I M Svishchev and P G Kusalik, *J Chem. Phys.*, **99**, 3049 (1993)
14. G Hura, D Russo, R M Glaeser, T Head-Gordon, M Krack, and M Parinello, *Phys. Chem. Chem. Phys.* **5**, 1981-1991 (2003)