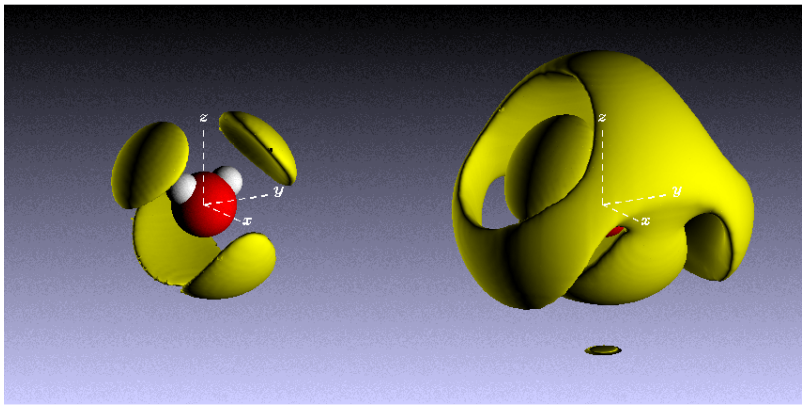
Empirical Potential Structure Refinement

- EPSRshell



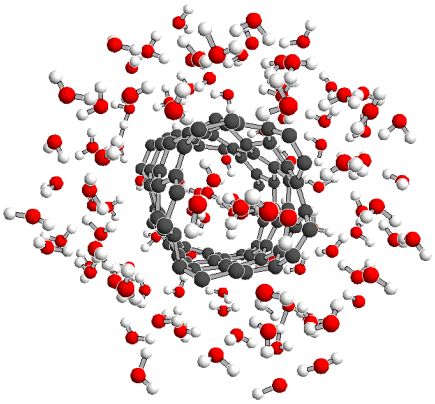
A User's Guide

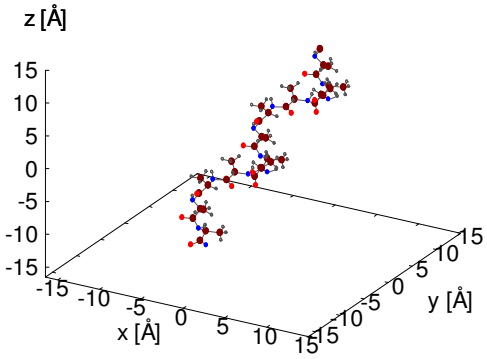
Version 18 - April 2009

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, plus many others. Without the 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.

2. Empirical Potential Structure Refinement.

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.

3. EPSRshell.

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.

4.1. Building the .**ATO** file – single atom molecules – **makeato**.
4.2. Making a molecule – **makeato.**
4.3. Making a molecule, role of the template file – **makemole**.
4.4. Running **fmole** to generate molecular coordinates.
4.5. Calculating intra-molecular atomic distances - **bonds**.
4.6. Modifying, mixing, growing, randomising the .**ato** file – **changeato, mixato, growcluster, introtcluster**.
4.7. Running **fcluster** to generate an initial configuration of molecules.
4.8. Building complex molecules and structures – **dockato.**

5. Operating the EPSR simulation program.

5.1. Setting up the neutron .**wts** files – **epsrwts**.
5.2. Setting up the X-ray .**wts** files – **epsrwtsx**.
5.3. Setting up the .**inp** and **.pcof** files.
5.4. Running the simulation.
5.5. Output files and data format
5.6. Reviewing the output.

6. Auxiliary routines.

6.1. Input and output data formats and running.
6.2. **partials** – calculate site-site radial distribution functions. (obsolete)
6.3. **coord** – calculate coordination numbers and 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.

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

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 **plot3d** – plotting the results from 7.2 and 7.3.

8. Some examples and exercises

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

9.1. Files you need to run EPSRshell

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 the timescale of order 0.1s so if things happen much slower than this we will not see them moving or else perceive things to be moving very slowly – like a snail for example – whereas if they happen much faster the motion is blurred or may not be visible 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 is can be regarded as a mathematical model that is used to described 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 is 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 the crowded shopping mall. Even on a crowded subway train it is rare for people to be in actual contact. Our senses prevent us approaching another person very closely, unless of course they allow 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, atomic 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 away from their initial position, 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 a crowded shopping mall in which everyone has stopped dead in their tracks, but can still shuffle, wave their arms, and talk to each other.)

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 simplest 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 diffraction 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 atom 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 diffraction 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. 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 diffraction data are introduced via yet another potential energy function, the empirical potential (EP), the contributions to this empirical potential being obtained from the difference between measured and simulated diffraction data. EPSR can therefore be thought of as a method for perturbing the interatomic potential model used in most other forms of computer simulation of liquids. (See the books by Allen and Tildesley [**3**], and Frenkel and Smit [4])

The bounding 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.

Of course 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 loose control of the number of adjustable parameters. 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 13 years by a process of trial and error. The most recent manifestation, EPSRshell was 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. EPSRshell is written with the latter group of people particularly in mind: it is designed to ensure that sensible default parameters are incorporated where ever practical, so that the number of decisions the user needs to make is kept to a reasonable limit, and to be compileable under a variety of 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 g77/g95 Fortran compilers which may be slightly different in form in other compilers.

The main EPSR routines can be compiled under the g77 or g95 Fortran compilers. Hence it runs perfectly satisfactorily on LINUX for example, although a LINUX version of the current software has not been tested recently. An exception to this occurs for those routines which call the PGPLOT library: under Microsoft Windows these are compiled using Compaq Visual Fortran. Setting this up for a Unix based system has been done in the past, but is not actively supported at present.

Other graphical plotting is done via calls to GNUplot, PGPlot, and recently .ato files, the files that contain the atomic coordinates in EPSR, can be plotted with Jmol. It is assumed these are available

under the operating system in which EPSRshell is run. 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 scattering equations since these are available in many places, such as the "ATLAS" 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, $\Delta U = U_{after}-U_{before}$ is less than zero the move is always accepted, and if it is greater than zero, the move is accepted with probability $\exp\left[-\frac{\Delta U}{kT}\right]$. 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

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

with the total potential energy of the system given by

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

where r_{ij} is the separation of atoms (i,j), and $\alpha(i)$ 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, but in general their values should not be taken too seriously. 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.

Instead of a proper 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

$$T(r) = \begin{cases} 1 & r < r_{minpt} \\ 0.5\left(1+\cos\dfrac{\pi}{2}\left(\dfrac{r-r_{minpt}}{r_{maxpt}-r_{minpt}}\right)\right) & r_{minpt} < r < r_{maxpt} \\ 0 & r > r_{maxpt} \end{cases} \quad (2.1.3)$$

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 reaction field method of truncating the Coulomb potential due to Hummer *et. al.* [5], the so-called "charged clouds" interaction, is used:-

$$T_C(r) = \left(1-\frac{r}{r_{maxpt}}\right)^4\left(1+\frac{8r}{5r_{maxpt}}+\frac{2r^2}{5r_{maxpt}^2}\right)\Theta\left(r_{maxpt}-r\right) \quad (2.1.4)$$

with $\Theta\left(r_{maxpt}-r\right)$ the Heaviside function.

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

$$U_{intra} = C\sum_i \sum_{\alpha\beta\neq\alpha} \frac{\left(r_{\alpha_i\beta_i}-d_{\alpha\beta}\right)^2}{2w_{\alpha\beta}^2} \quad (2.2.1)$$

where $r_{\alpha_i\beta_i}$ is the actual separation of the atoms α,β in molecule i, and

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

with $\mu_{\alpha\beta} = M_\alpha M_\beta \big/ \left(M_\alpha + M_\beta\right)$ the reduced mass of the atom pair $\alpha\beta$, 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:-

$$U_{\alpha\beta}\left(r_{ij}\right) = 4\varepsilon_{\alpha\beta}\left[\left(\frac{\sigma_{\alpha\beta}}{r_{ij}}\right)^n - \left(\frac{\sigma_{\alpha\beta}}{r_{ij}}\right)^6\right] + \frac{q_\alpha q_\beta}{4\pi\varepsilon_0 r_{ij}} \qquad (2.2.3).$$

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_α different types of atom in the system, there will be $N_\alpha(N_\alpha+1)/2$ 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:-

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

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

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 contain any 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 a very 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 functions:-

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

where

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

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. Obviously 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

$$n_i = \frac{r_i}{\sigma_r} - 3 \tag{2.3.3}$$

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

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

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

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

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=0.003**Å** in (2.3.5) is found to give satisfactory results, with σ_r=0.012**Å** 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.

2.4 The uniform atom distribution.

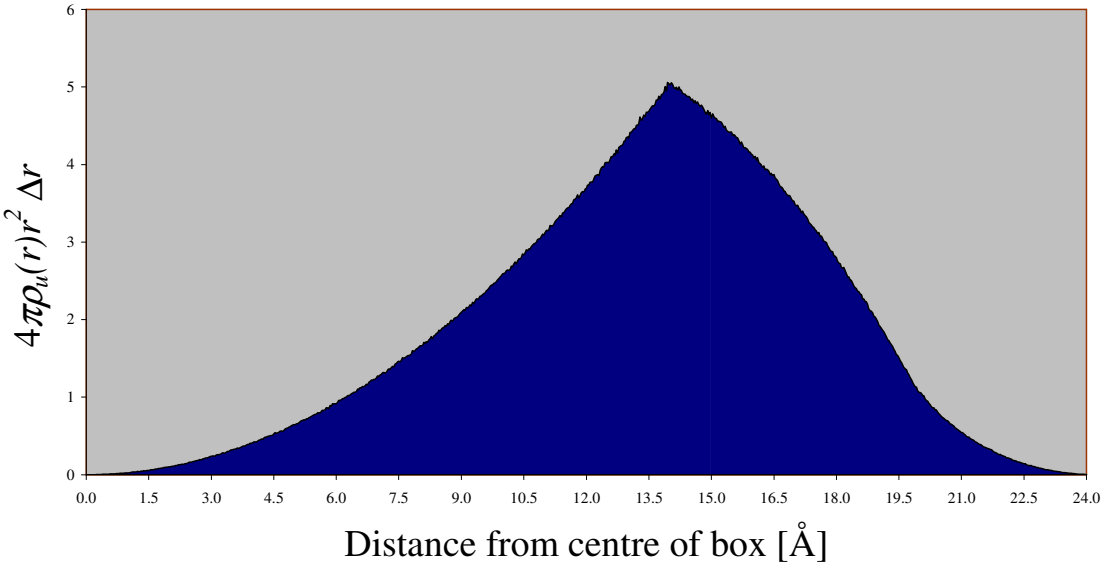
For an infinite system the radial distribution function $g(r)$ is defined as the ratio of the local density of atoms at a distance *r* from an atom at the origin, $\rho(r)$, to the average density of atoms in the whole system, ρ, i.e. $g(r) = \frac{\rho(r)}{\rho}$. For the computer simulation box the 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 $r_{\max} = 0.5D$, 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 $r \le r_{\max}$.

Nonetheless it is sometimes useful to look at the distribution of atoms beyond this sphere, effectively including all the atoms within the box, because the distribution of atoms outside the minimum image sphere often gives a strong clue as to the state of the simulation. If a significant density deficit or excess is observed as we go outside the minimum image sphere, it means the atoms cannot be evenly distributed throughout the box and may not therefore be in equilibrium. However to calculate $g(r)$ beyond $r_{\max}$ the average density of atoms itself becomes a function of distance from the central atom, since only the region *inside* the box can be included. The maximum distance that can be calculated is into the corner of the box, which for a cubic box is $r = \frac{\sqrt{3}D}{2}$. Provided one is careful only to count each atom in the box once when estimating $g(r)$ 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. Thus in EPSR, while $g(r)$ is *calculated* into the corner of the box, atoms beyond a distance $r = r_{\max}$ are not used either for estimating the structure factor of the simulated box, or for calculating the potential energy of the simulation. (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}.)

Probably there are exact formulae for calculating this "uniform atom" distribution, but EPSR cheats here by throwing a large number of particles at random into the box, and calculating the distance dependence of the resulting uniform distribution, $\rho_U(r)$. 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, to be 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. This uniform atom distribution is recalculated at regular intervals as the simulation is run to ensure that any statistical quality of this distribution does not unduly affect the estimated radial distribution functions of the simulation.

Figure 1: Uniform atom distribution for SiO2



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. Currently there are no moves which involve change of symmetry.

For individual atom moves only the change in the intramolecular potential energy, ΔU_{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

$\exp\left[-\left\{\Delta U_{Intra}+\frac{1}{k_BT}\left(\Delta U_{Ref}+\Delta U_{Ep}\right)\right\}\right]$, 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 (α,β) 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 un-normalised 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 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 will need an inversion to the matrix w_{ij}. If there are enough 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:

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

$$w'_{ij} = f w_{ij}, \quad \text{for } 1 \le i \le M \tag{2.6.2}$$

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

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

Effectively this additional set of weights is equivalent to saying we accept the data with confidence *f*, and the simulation with confidence (1-*f*). 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 any solutions. The solution can be checked from the requirement that the matrix $P_{j'j} = \sum_{i=1,M+N} w_{j'i}^{-1} w'_{ij}$ must be unitary.[2] Since the matrix w'_{ij} is always well determined from (2.6.1) and (2.6.2) 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 m^{th} iteration of the algorithm each distinct pair of atoms, *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 m^{th} iteration for any particular pair of atoms, *j*, is determined from

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

(At the outset, with $m = 0$, the coefficients $C_{k,m}^{(j)}$ are set to zero.) After the m^{th} 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:-

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

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, defined by $\overline{U} = 4\pi\rho \sum_{j=1,N} \left| U_m^{(j)}(r) \right| g_j(r) r^2 dr$ reaches some predefined limit. The latter tends to happen when 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

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

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. Indeed the need to set a cap on the empirical potential is a likely sign of appreciable systematic error in the data.

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:

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

These estimated partial structure factors can then be compared with the simulated partial structure factors.

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.

There is also scope within the input files to provide a 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 for each site-site 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 $g(r)$ 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 EP. This repulsive potential is treated as part of the reference potential, so will appear even if the EP is switched off. Its effect can be controlled by specifying the minimum allowed distance between individual atom pairs in the EPSR input file.

Once the EP has stopped changing, or the absolute energy of the EP has reached its specified limit, 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.

---

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), 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 G. Hummer, D. M. Soumpasis, and M. Neumann, *J. Phys.: Condens. Matter* **6**, A141-A144 (1994).

3. EPSRshell

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. All files in either the working folder or the home folder that are to be accessed by EPSRshell should not 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, but it is a good idea to copy one from another folder if it does not exist, since creating one from scratch could be a bit tedious. If the file does not exist in the current working folder when the first plot request is issued a set of predefined plotting defaults are automatically set by the program and these are used to create the initial version of "plot_defaults.txt" in this folder. 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 pressed. Commands will generally do something if pressed, 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 will likely develop a problem and could 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" does not work from the Main menu.

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 this should not cause any problem. 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 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), **.wts** (neutron and x-ray weights files), 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
.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))
.p01	EPSR – site-site empirical potentials
.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 energies, pressures and R-factors at each iteration 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

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 (_) 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

[3] If, 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.

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 is / instead of \ for Windows, so this needs to be remebmered 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.
makeato	makes a .ato file using a series of values input from the console. This 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 fmole, followed by introtcluster and fcluster after the mixture is set up to generate distinct molecules..
changeato	allows all the main parameters in the .ato file to be altered – it works like a standard EPSRshell menu.
introtcluster	performs a random translation and rotation of every molecule in the box.
epsrwts	sets up the .wts file which gives the relative or absolute neutron weight factors of individual PSFs in the input neutron diffraction patterns.
epsrwtsx	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, **plot3d**	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.

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.
coord	calculates the average coordination number and coordination number distribution for specified atom pairs over a specified distance range.
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>	New for 2009! 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.

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 plot3d, require a number of variables to be setup. For EPSR 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.

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
plot3d
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 much 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 currently 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 plot screen and alter various options, such as line styles, and so on. If you right click you can even open the GNUplot console to make other changes. (In LINUX you need to revert to the shell, where the GNUplot command prompt is displayed, and type the command at the GNUplot prompt. 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 whether you want to plot all of the molecules in the box (1) or a subset based on a particular molecule at the origin (2). If the latter you are also asked which molecule

you want at the origin and how far away you want to plot the surrounding atoms. If you type zero for this distance you are shown only the molecule specified.

It then will 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 the number of types and which particular types you want plotted.

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.

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 a particular system.

4.1 Building the **.ato** file – single atoms and molecules.

Simplest way to make the initial **.ato** file is to run **makeato**. By asking some hopefully fairly obvious questions, this program will allow you to build a box containing 1 atom or 1 molecule 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.

Alternatively you can build your **.ato** file by hand, but this can be tricky since the format of the file is important. Probably the easiest method, if you wish to do this is to edit an existing **.ato** file containing 1 molecule, and then expand this with **mixato** once the changes are complete.

However if the molecule is at all complicated, then it is STRONGLY recommended you use **makemole** (Section 4.3) to build the molecules if at all possible. This sets up a template molecule which can then be used to alter the molecules in your **.ato** file in a consistent manner.

The basic structure of the **.ato** file is as follows:-

Line 1: (free format) Number of molecules, Box dimension (Å), Temperature (K)

Line 2: (free format) Tol (not currently used), Step size for intramolecular translations, Step size for headgroup 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.). Setting any of the step sizes to zero will inhibit trial moves of that type.

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 (not really used in present program), and the molecule number. This last number is given for information purposes only and is not read on input, 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 distance 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.

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.

Line 8: (format(1x,A3) – only present if line 7 is not zero) 1 space and the word (in upper case) ROT. This is to specify an intramolecular headgroup or side chain rotation. Currently no other moves are recognized.

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

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 headgroup in this molecule.

This completes the input for the first molecule. Lines 3 – 10 are then repeated for each subsequent molecule in the box. Is 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 **epsrwts** 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) Must contain two numbers. These are used in **fmole** to keep non-bonded atoms as far apart as possible, but are not currently used by any other programs

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. Also on this are two step sizes for these molecule types, one for whole molecule rotations and one for whole molecule translations. This 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.

IMPORTANT NOTE: When the **.ato** file is read each molecule is given a type, as listed on line 15 above. This molecule type is determined from the atom type of the first atom in each molecule. Thus in mixtures of molecules it is important to ensure the first atom of each molecule has a distinct atom type, otherwise the molecule will not be recognized as a different type and an error will occur.

4.2 Creating a molecule **– makeato.**

This program is run by simply typing "makeato" in the Main menu of EPSRshell (after setting the working folder to where the .ato file is to be created). Note however that if the molecule is at all complicated it is generally much better way to generate a molecule via the command "makemole" – see Section 4.3. The input to **makeato** is hopefully self explanatory, however note that when you type the name of the file at the outset, you need only to type the filename, not the extension, which the program will automatically set to **.ato**. This program creates the **.ato** file for a single atom or molecule – it is a good idea to sketch out the molecule on a piece of paper and assign atom types and atom numbers. Here is an example of such a sketch for a methanol molecule. There are six atoms in the molecule and four atomic types. Each atom has a label and a number. Relevant intramolecular distances are defined:-

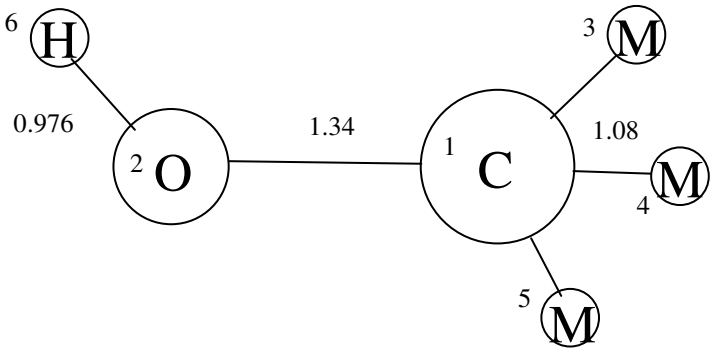


Figure 4.1

- there is a total of 5 such bonds to be defined. In addition not shown here are the bond angles, which for the purposes of demonstration can be set to the tetrahedral angle, 109.47º - there are 7 such angles to be defined, those between atoms (1,2,6), (2,1,3), (2,1,4), (2,1,5), (3,1,4), (3,1,5), and (4,1,5). makeato will use these input numbers and angles to define the distance between the first and last of the triplet, e.g. between atoms 1 and 6 for triplet (1,2,6). Next it will ask for 2 atom numbers to make a rotational group, atoms 1 and 2 in this case. Then it will ask you to type the number of atoms in this rotational group and the corresponding atom numbers. **makeato** will also ask for some parameters, like the temperature and atomic number density, and the Lennard-Jones parameters.

At the end **makeato** asks you for the vibrational weighting. This is the value of $C/2$ in equation (2.2.1).

4.3 Creating a molecule **– makemole**

This is a another way to build a molecule. Basically a template file is created with the editor – this must have the file extension **.mol**. From this template file the program **makemole** determines the

number of atoms, the atom types and the appropriate bonding pattern from the position of asterisks in the **.mol** file.

The first line of the **.mol** file determines the size of the grid (number of spaces in each column) to be used in the file, and the second line is a ruler line to help position atoms and bonds correctly. Each line starts with a line number in the first (zeroth) column to help identify atom locations.

The easiest way to understand how the program works is to study some of the examples supplied with the distribution of EPSRshell. Atoms are specified by a label and bonds by asterisks. To be bonded any two atoms must have the same asterisk in an adjacent square. Here is an example of a simple **.mol** file to get you started – it sets up the **.ato** file for a water molecule.

```
 4
|0  |1  |2  | 3 | 4 | 5 | 6 |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |
  1                           OW
  2                       *       *
  3                   HW              HW
bond OW  HW     0.97600
angle HW  OW  HW   104.50000
potential OW     0.65000    3.16600   16.00000   -0.84760 O
potential HW     0.00000    0.00000    2.00000    0.42380 H
temperature  0.300000E+03
density  0.100200E+00
ecoredcore    1.00000    3.00000
```

Thus the OW atom is bonded to two HW atoms, but there is no bond between the HW atoms. The atom positions are referenced by a grid location (row number, column number), for example atom OW is at position (1,7), the first HW atom is position (3,5), and so on.

There is one feature which will not be obvious from the example **.mol** files. This is the way the program reads the data and decides whether each pair of atoms is bonded or not. Basically it reads the data from top to bottom and from left to right. When it finds an atom (an element with at least one letter in it) it looks at all the PREVIOUS adjacent squares to see if any contain a *. Thus in the example above when the program comes across the atom OW at location (1,7) it will look ONLY at location (1,6) for an adjacent *. It will not at that stage see the bonds to the HW atoms. When the program gets to the first * at (2,6) it notes the fact that there is a * at this position, adds it to the list of * positions, and will also note that this * has an atom OW at the adjacent position (1,7). It will NOT however see the HW at (3,5) at this point. Next when the program gets to position (3,5) it will find the star at the previous position (2,6) add it that *'s list of neighbours, and hence find it is bonded to the OW at (1,7). This procedure is quite deliberate and was done so as to ensure that atom bonds do not get doubly generated while keeping the reading algorithm as simple as possible.

If for example the first HW had been placed differently like so:

```
 4
|0  |1  |2  | 3 | 4 | 5 | 6 |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |
  1                           OW
  2                   HW    *       *
  3                                 HW
```

then on reading the file, the atom HW now at (2,5) will not see the * at (2,6) and will not form any bonds. When the program reads the * at (2,6) it will add OW and HW to its list of neighbours, because they are both in previous and adjacent elements, but this is not sufficient to generate a bond between the first HW and OW**,** since bonds are only created when an atom is found, not when a * is found.

The following example creates a cyclohexane molecule:-

```
 4
```

```
|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
  1           M                   M
  2                   *       *
  3                       C
  4                   *       *
  5 2M    *   C                   C   *  2M
  6           *                   *
  7 2M    *   C                   C   *  2M
  8                   *       *
  9                       C
 10                       *
 11                      2M
bond C    C       1.54000
bond C    M       1.08000
angle M    C    M     109.47000
angle M    C    C     109.47000
angle C    C    C     109.47000
potential C       0.20000    3.70000   12.00000    0.00000 C
potential M       0.00000    0.00000    2.00000    0.00000 H
temperature  0.200000E+02
density  0.900000E-01
ecoredcore    1.00000    3.00000
```

Note the use of a number in front of a symbol to generate multiple bonds of a particular type.

There is one other useful feature which is used for complex molecules which cannot easily be represented on a 2D sheet like this. These are the mapped atoms, specified by the word “map” followed by a grid reference. The reference can be either absolute in the form (row number, column number) or relative to the current grid point, + or – number of rows and columns. For the relative reference there is no need to surround the numbers with (). Here is the **.mol** file for creating a C60 molecule:-

```
 4
|   |1  |2  | 3 | 4 | 5 | 6 |7  |8  |9  |10 |11 |12 |13 |14 |15 |16 |17 |18 |19
  1                                       C1              C2
  2                                   *       *       *       *
  3                               C3              C4              C3
  4                               *               *               *
  5           C2                  C3              C4              C3
  6       *       *       *       *       *       *       *
  7  C3              C4              C1              C2
  8  *               *               *
  9  C3              C4              C1              map-2+0
 10       *       *       *       *       *       *       *
 11           C2              C3              C4              map-6+0
 12                           *               *               *
 13           map-2+0         C3              C4              C3
 14       *       *       *       *       *       *       *
 15  map-6+0          C4              C1              C2
 16  *                *               *
 17  C3               C4              C1              map-2+0
 18       *       *       *       *       *       *       *
 19           C2              C3              C4              map-6+0
 20                           *               *               *
 21           map-2+0         C3              C4              C3
 22       *       *       *       *       *       *       *
 23  map-6+0          C4              C1              C2
 24  *                *               *
 25  C3               C4              C1              map-2+0
 26       *       *       *       *       *       *       *
 27           C2              C3              C4              map-6+0
 28                           *               *               *
 29           map-2+0         C3              C4              C3
```

```
 30      *       *       *       *       *       *       *
 31  map-6+0         C4              C1              C2
 32  *               *               *
 33  C3              C4              C1              map-2+0
 34      *       *       *       *       *       *       *
 35          C2              C3              C4              map-6+0
 36                          *               *               *
 37          map-2+0         C3              C4              map(3,15)
 38      *       *       *       *       *       *       *
 39  map-6+0         C4              C1              map(1,13)
 40  *               *               *
 41  map-34+0        C4              map(1,9)
 42      *       *       *
 43          map(5,3)        map(3,7)
bond C1  C1     1.50000
bond C3  C3     1.50000
bond C4  C4     1.50000
bond C1  C3     1.50000
bond C1  C4     1.50000
bond C3  C4     1.50000
bond C2  C3     1.50000
bond C2  C4     1.50000
bond C2  C2     1.50000
bond C3  C3     1.50000
angle C1  C3  C3   120.00000
angle C1  C4  C4   120.00000
angle C3  C1  C4   120.00000
angle C1  C1  C3   120.00000
angle C3  C4  C4   120.00000
angle C1  C3  C4   120.00000
angle C2  C3  C3   120.00000
angle C2  C4  C4   120.00000
angle C3  C2  C4   120.00000
angle C1  C1  C4   108.00000
angle C1  C4  C2   108.00000
angle C4  C2  C4   108.00000
angle C3  C3  C4   108.00000
angle C3  C4  C2   108.00000
angle C3  C3  C3   108.00000
changelabel C1  C
changelabel C2  C
changelabel C3  C
changelabel C4  C
potential C      0.65000    3.00000   12.00000    0.00000 C
temperature  0.200000E+02
density  0.500000E-01
ecoredcore    1.00000    3.00000
```

For example the atom at position (9,13) is in fact the C2 atom at (7,13) while the atom at (43,7) is in fact the C3 atom at (3,7).

The other parameters needed for the **.ato** file are set up using the variables as listed in these examples. As for other commands each command is followed by a list of values separated by some spaces, typically up to about 5 spaces are allowed – more than this and the subsequent variables on a line may not get read. These lines could come anywhere after the first two lines in the **.mol** file, but it is probably easiest to list them after all the atoms and bonds have been defined. Note that in order for the program to work correctly it may be necessary to define a set of different atom types in the **.mol** file, which are in fact to be the same atom type within the **.ato** file. The command "changelabel" is useful for setting the atom labels and types back to what will be needed

for the **.ato** file. The last letter on the "potential" line is the atomic symbol for that atom and has to be one of the recognised atomic symbols as listed in the Period Table.

The **.ato** template file is generated by typing "makemole <filename> where <filename> is the name of the **.mol** file that has been created. This generates a **.ato** file and also a **.atm** file which lists the atom numbers. This is useful for specifying extra bonds, rotational groups, and dihedral angles.

Other **makemole** commands that are useful are:-

extra defines an extra bond distance between specified atom numbers. Format is "extra <atom number 1> <atom number 2> <bond length>". The relevant atom numbers can be obtained by looking at the **.atm** file created after the first pass through with **makemole**. This feature can be used to stabilise a molecule when the near neighbour bond distances and angles are not sufficient by themselves.

rot defines a "ROT" group – headgroup or sidechain rotation. Format "rot <atom number 1> <atom number 2>. The two atom numbers specify the axis of rotation. **makemole** will sort out which atoms will need to be rotated. However the order of the two atoms is important. **makemole** uses the second atom of the pair to determine which atoms will be rotated, you should ensure the second atom has the fewer atoms bonded to it than the first. It would be crazy for example to have a methyl group attached to a long polymer molecule, and then rotate the polymer about the common axis, instead of rotating the methyl group!

dihedral defines a dihedral angle within the molecule. Format is "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. **makemole** converts these numbers and the bonds between each pair of atoms to a distance between atoms 1 and 4. If any of the pairs 1-2, 2-3, 3-4 are not bonded, or either of the angles 1-2-3, 2-3-4 are not defined then nothing is generated. Note that since the angle is being converted to a distance, there may be some loss of information when the molecule is actually formed by **fmole**, so if this is likely to happen it would be advisable to define two dihedral angles for particular rotation axis involving different atoms 1 and 4.

qradius allows you to set the charge radius of a particular atom type. If not set by this command the charge radius for an atom is set to zero. Format is "qradius <atom type symbol> <charge radius for this type>

vibtemp allows you to set the vibrational weighting. This is the value of $C/2$ in equation (2.2.1). A value around 65 seems to give realistic intramolecular atom distributions, but you may need to change this for particular cases. However when initially setting up a molecule it is a good idea to use a much larger number, e.g.1.0e5, so that the molecule is forced close to its ideal conformation. It can always be relaxed once it is clear the bonding constraints have been set up correctly. Format is "vibtemp <vibrational weighting>. Note that increasing this weighting has the effect of making the bondlength variance between molecules smaller.

The best way to generate a **.mol** file is to take an existing one and edit it – there is currently no feature within EPSRshell which will help create this file. But since the format is reasonably graphic it is hoped that this will no present a major obstacle to using it. It is also a good idea to set a low density, e.g. 0.001 when setting up a molecule, particularly if it is quite big – this will help to keep all the atoms of the molecule inside the simulation box! Note that the minimum image convention is applied to the centre of mass coordinates of each molecule in EPSR, so that it is perfectly possible for individual atoms on any given molecule to occur outside the box walls, even

though the centre of mass of the molecule is inside the box. (Of course when calculating interatomic separations, the minimum image convention is always applied to atom pair distances.)

makemole is run simply by typing "makemole <filename> where the filename is the name of **mol** file. If the file is not specified, **makemole** enters the search menu to look for files of the correct type.

4.4 Running **fmole** to generate molecular coordinates.

Having specified the molecule, it now needs to be brought to equilibrium since **makeato** or **makemole** will not have done anything to define the actual coordinates of the atoms – they will all be set to zero. So far they will only have defined the bonding within the molecule and the reference potential parameters. To run **fmole** you type "fmole <.ato filename> <ntimes> <nupdates> in the Main menu. <ntimes> is the number of times you want it to run the simulation. <nupdates> is the frequency you want to update the neighbour list. When running **fmole** on a single molecule <nupdates> can be set to 0 since any atoms bonded to any other atom in a molecule will automatically appear in the neighbour list. If any argument is not specified you will be prompted for it.

Having made the molecule it is a good idea to check that it looks like what you might expect, that there are no atoms in strange places. To do this, run **plotato** (Section 3.7) on the resulting **.ato** file from the EPSRshell menu. This will generate a picture of the current molecule.

A new feature of **fmole** is that before it runs the molecular refinement, it first runs **makemole** on any **.mol** files that are specified at the end of the **.ato** file. It then reads the bonding, rotational groups and potential parameters from the **.ato** file generated by **makemole** and inserts them into the appropriate molecules in the present **.ato** file. This allows the **.mol** file to maintain the template for the molecules of a particular type in this file. For this reason it is a good policy NOT to run **fmole** directly on the file produced by **makemole**, but to make a copy of it, with a different name – you can for example use **mixato** (see Section 4.6) to create a "mixture" of just 1 molecule from the template file and give it a new name on exit. Since **makemole** will have set all the coordinates in the .**ato** file to zero, if **fmole** is run on that file, it will be continually generating a new molecule from scratch. If **fmole** is run on a copy of the **.ato** file produced by **makemole**, the bond and potential parameters will be updated in the copy but not the atomic coordinates.

Note that it is perfectly sensible to set up a **.mol** file even when there is only one atom in a molecule: this means you have record of what potential parameters and symbols you are using for that atom type and makes it easy to change those parameters using **fmole**.

4.5 Calculating intra-molecular atomic distances - **bonds**.

Another check you can do is to see that the bonds you have specified have indeed been obtained by **fmole**. To measure bond lengths you type "bonds <.ato file name>", and a new file will be produced called '<.ato file name>.bonds' which lists the interatomic separations of each of the different molecule types in the simulation box. If there is more than one molecule of each type in the simulation box then the distances shown are the average values of the interatomic distances for all the molecules of that type in the box. The matrix printed is triangular in that the upper triangle shows the bond distances, while the lower triangle shows the r.m.s. deviation of those distances. Below is a simple example of **.bonds** file:-

```
          1OW    2HW    3HW

1OW     0.000  0.981  0.983
2HW     0.075  0.000  1.546
3HW     0.075  0.110  0.000
```

4.6 Modifying, mixing, growing, randomising the .**ato** file – **changeato, mixato, introtcluster**.

As described in Section 4.4, one way to change the parameters for a **.ato** file is to edit the appropriate **.mol** files and then run **fmole**. This procedure is not always possible, particularly if a particular molecule type does not have a corresponding template **.mol** file associated with it. In this case it is necessary to 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, bond length, add bonds, subtract bonds, but for changing the bonds you will need to specify not only the types of the atom pairs you wish to modify, but also their atom numbers. Therefore even if a molecule has 3 bonds which are identical and all of which you wish to change, you will need to specify the change to each of these bonds separately.

Having made the molecule(s), you now need to make the box bigger to contain more molecules. You can do this with **mixato**. This basically starts you from 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 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 as large as you want. **mixato** however will only use the FIRST molecule in each **.ato** file and produce multiple carbon copies of that one molecule. Hence after running **mixato** it is essential that **fmole** is run quite a few times (typically 1000 to 10000 times) to disorder the molecules.

Having made the simulation box the size you want it with **mixato** you need to randomize the molecule positions and orientations, otherwise you are may have atom clumping. This can be removed by using the program **introtcluster**. 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.

After **introtcluster** you need to run **fmole** again to disorder the molecules. IMPORTANT NOTE: After **mixato** you must run **introtcluster** BEFORE running **fmole**, otherwise the program will find all the molecules on top of each other and so likely overflow the neighbour list.

4.7 Building complex molecules and structures – **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.

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 defines a selenium atom with two 'QSe' atoms attached, forming an internal angle of 120°. Note that for **dockato** it is a good idea 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:-

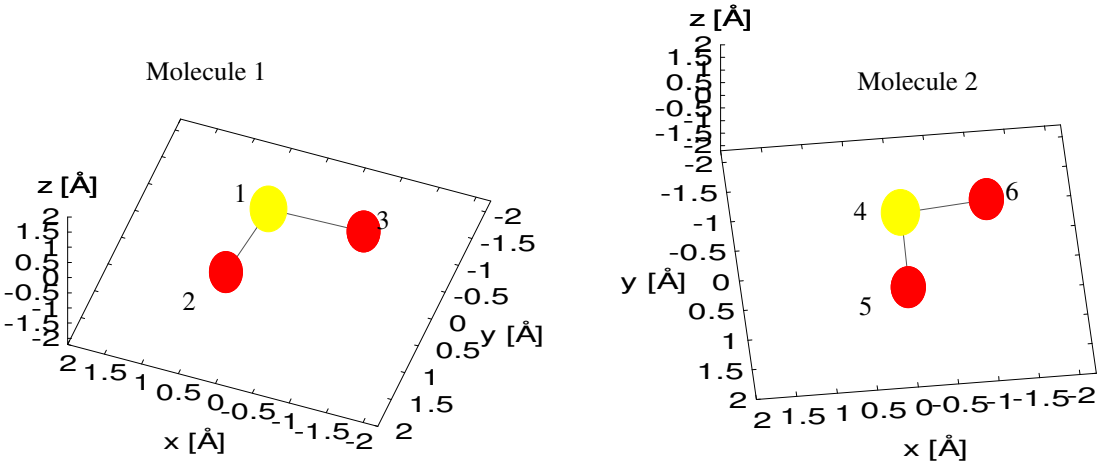


Figure 4.2

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:-

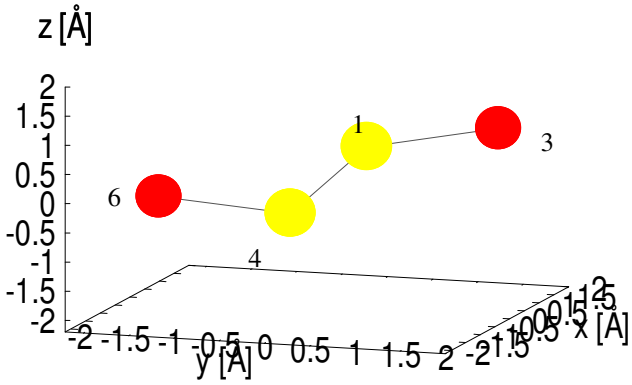


Figure 4.3

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.

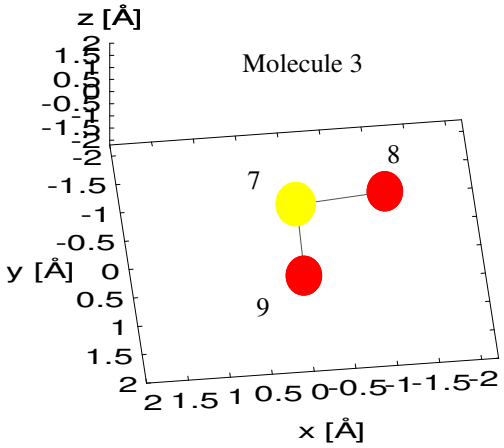


Figure 4.4

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 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.

Argument 3: Atom type in the **docking** molecule to be used for rotating this molecule on to the **docked** molecule.

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.

Argument 9: Atom type in the **docked** molecule to be used for rotating the **docking** molecule on to this molecule.

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 Name of **.ato** file in which to save the new molecule.

If defined the dihedral angle is constrained by a new bond between the two atoms specified by arguments 4 and 10 of the appropriate length.

The **dockato** command probably does not work perfectly at this stage, but it seems to work in some simple instances. 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 se_add_12

And this is the result, 'se_add_12.ato':

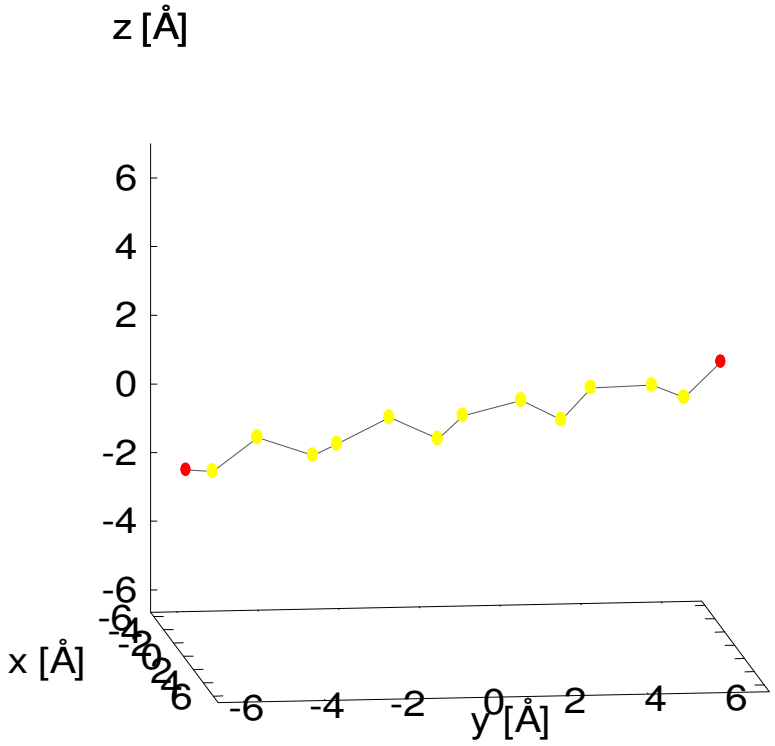


Figure 4.5

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

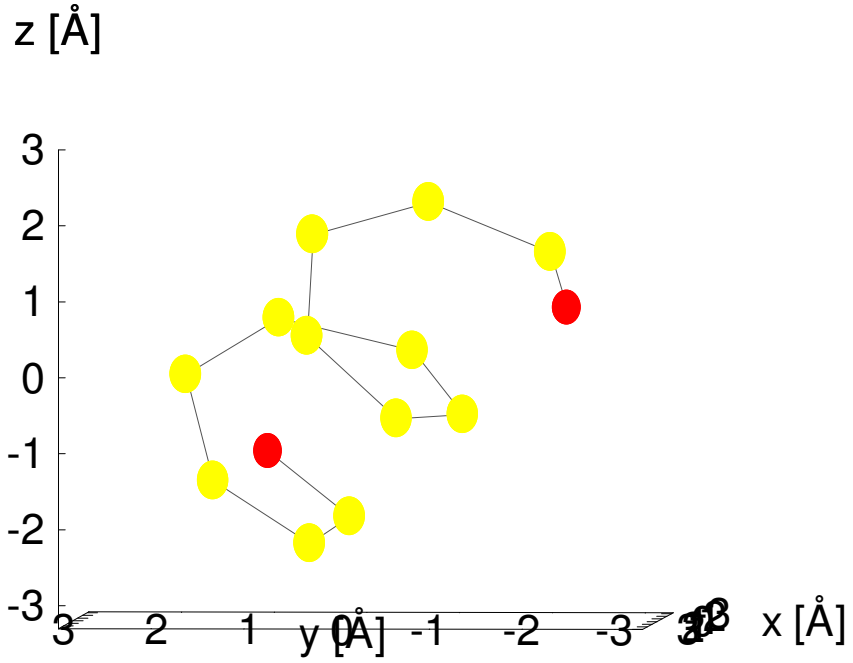


Figure 4.6

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 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 ($\sigma_{\alpha\beta}$ in equation (2.2.4)), 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 'XX' to signify that there is no template file for this molecule.

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 - **epsrwts**

By far the safest and easiest way to do this is to run the program **epsrwts**. This delivers the .**wts** in the correct format appropriate to EPSR. Since you need a **.ato** file to run EPSR, **epsrwts** 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. **epsrwts** 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 discussed above has 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 **epsrwts** 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:-

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

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

Unfortunately **epsrwts** is a rather unwieldy program: if anything goes wrong while typing the responses to the numerous questions, there is no alternative but to abandon it and start again.

5.2 Setting up the X-ray .**wts** files – **epsrwtsx**.

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:-

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

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:-

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

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 $B(Q)=\frac{\sum_{\alpha} c_\alpha f_\alpha^2(Q)}{\left(\sum_{\alpha} c_\alpha f_\alpha(Q)\right)^2}$. The figure below shows an estimate of this function for SiO_2, based on the independent atom form factors:-

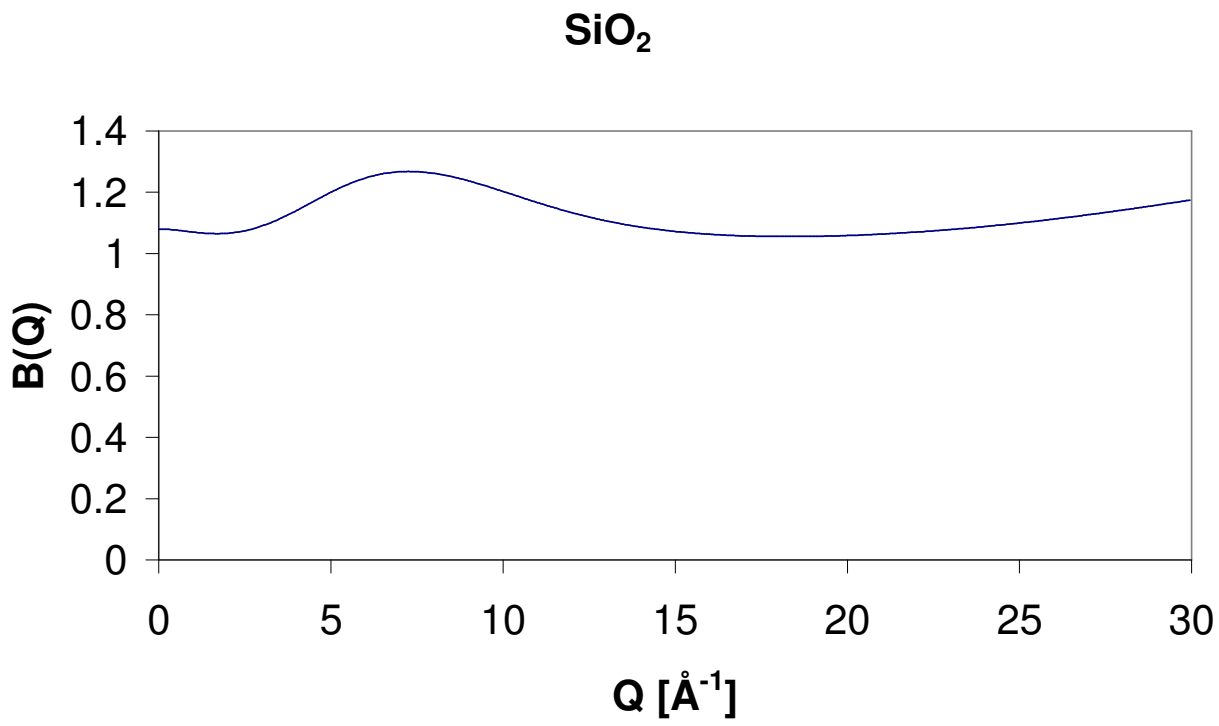


Figure 5.1

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

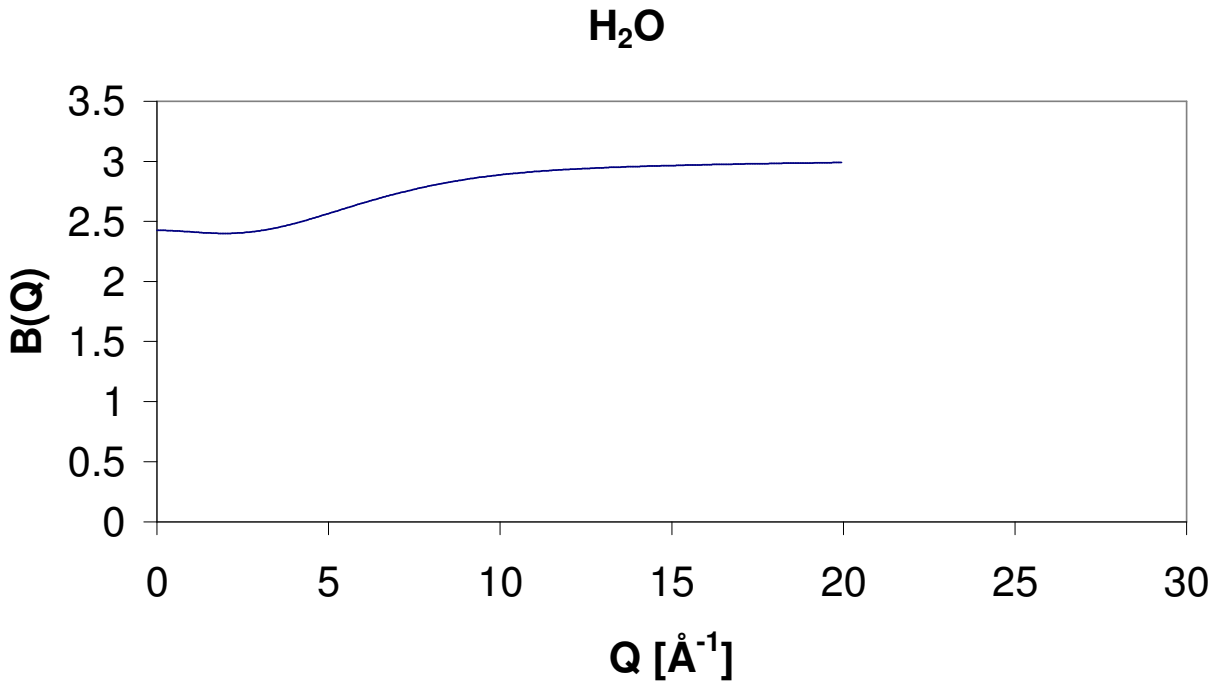


Figure 5.2

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 **epsrwtsx** 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. Perhaps in the future, all normalisations, whether X-ray or neutron should be done according to (5.2.5)?

However EPSR can deal with either case, and the first question that is asked when running **epsrwtsx** is whether the data have been normalised per atom or per molecule.

epsrtwts 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 [6] 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:

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

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

$$\sum c_\alpha q_\alpha = 0 \qquad (5.2.9)$$

is rigorously satisfied.

IMPORTANT NOTE: currently **epsrwtsx** 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 0.5e to shift off each hydrogen atom onto the neighbouring oxygen atom, then q_O = -1 and q_H = +0.5.

NOTE that in **epsrwtsx** the ratio $\frac{q_\alpha}{f_\alpha(0)}$ is 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, 6]

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 **epsrwtsx** 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>X-ray.wts'. The end of this filename is crucial as it is the only way 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).

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: EPSR 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.

```
sio2.EPSR               Title of this file
feedback    0.8           Confidence factor - should be < 1. [0.8]
potfac      0.0           1.0 to enable potential refinement, 0.0 to inhibit
ereq        5.0           Overall requested energy amplitude - overrules efilereq
sizefactor  1.0           Multiplying factor for box dimension. [1.0]
nq          400           Number of Q values. [400]
qstep       0.05           Size of Q step [1/A]. [0.05]
ireset      0            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 molecule moves between molecule shakes. [100]
inter       5            Number of iterations in running averages. [5]
rho         0.0683396161            Atomic number density - will be derived from .ato
file
cellst      0.03           Size of r step [A]. [0.03]
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    sio2.ato            Name of .ato file
fnamepcof   sio2.pcof            Name of potential coefficients file.
qmin        0.05           Minimum value of Q used for potential fits. [0.05]
ndata       1            Number of data files to be fit by EPSR

data   1

datafile    sio2sq.doq            Name of data file to be fit
wtsfile     sio2tot.wts            Name of weights file for this data set
nrtype      3            Data type - see User Manual for more details
rshmin      0.7           Minimum radius [A] - used for background subtraction
szeros      0.0           Zero limit - not used - automatically set to 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 a **.ato** has been 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 full 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. Initially set to 0 to allow you to see how well the reference potential is doing on its own. Once you are convinced you have done all you can with the reference potential, you set **potfac** to 1 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 how large the EP can get when it is refined. The value to use depends very much on the particular system being looked at and can only be found by trial and error. Typically you might want to aim for the contribution to the configurational energy of the simulation from the EP being around 10-20% of the total, but this will vary individual cases. You inspect the absolute value of the empirical energy by inspecting the **.out** file.

It is possible to override the value of **ereq** by setting it to zero. When this is done then the values of **efilereq** for individual datasets take over. This allows you to give different weightings to different datasets when refining the potential. In either case if **efilereq** is set to zero the EP will not be refined against that dataset (although the program will continue to calculate the fit and radial distribution functions corresponding to that data set).

sizefactor Normally set to 1.0, 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 sizefactor, in the early stages of the simulation until any evidence of overlaps has disappeared. Then it can be increased back gradually to it's normal value of 1.0

nq Controls the number of *Q* values used in the simulation. It defaults to 400 and has maximum value of 600.

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

ireset Value either 1 or 0. This (1) resets the empirical potential to zero, plus sets various other accumulators to zero, sets **nsumt** to -1, and calculates the inverse of the weights matrix. It must ALWAYS be done when starting a new simulation, and is set automatically whenever parameters which define the potential are altered. It will also happen automatically if you change the value of **feedback**, change the number of datasets to be fit, or number of *Q* or *r* values. It is a good idea to reset the potential 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.

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

$\Delta(Q) =$ **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.

rshmin This is specified for each dataset and represents the minimum distance in *r* space that this dataset is expected to contribute useful information. It defaults to 0.7Å, but can be set to larger or smaller values than this. It is only used when the variable **nsmoop** is set to unity. If this is so, 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 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.

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

```
roverlap  0.9000000     Minimum allowed intermolecular separation between two
atoms
rminfac   0.6000000     Factor to set the minimum separation between pairs.
[0.6]
rminpt    9.000000      Radius at which potential truncation gets to 0.5
rmaxpt    12.00000      Radius at which potential truncation goes to 0.0
rbroad    0.0000000     (NOT USED NOW)
expecf    0.3000000     Potential decay for short distance repulsive term.
[0.3]
rcharge   0.0000000     Calculates energy due to molecular polarisation. [0.0]
power     12.00000      Repulsive power in Lennard-Jones potential. [12]
ncoeffp   100           Number of coefficients used to define the EP. [100]
pdmax     12.00000      Maximum distance of Empirical Potential (CANNOT BE
ALTERED – set by rmaxpt
pdstep    0.1200000     Spacing between coefficients in r. Set by program
npitss    1000          Number of steps for refining the potential. [1000]
paccept   0.0005000000  Acceptance factor for potential refinement. [0.0005]
psigma2   0.00090000    Width of Poisson curves. [0.003]
```

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 hardness of the interatomic exponential repulsive potentials (**expecf**) used to prevent atomic overlap.

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 **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

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

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 minimum distances can play a powerful role in helping to avoid the empirical potential from generating spurious structure at low 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.

In addition to specifying minimum distances, it is possible to specify up to 5 Gaussians in the EP for each atom pair. If one or more Gaussians are specified, then no changes are allowed to the EPS for that particular atom pair until the Gaussians are removed. This is to prevent the EP from overriding the effect of the Gaussian. Each Gaussian requires a position (P), width (W) and a height (H), specified in that order on the same line as the minimum distance.

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 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 are not so reliable due to the short range nature of the potentials, but even there a pressure of 1000kbar should be treated with some suspicion! It may mean things are not right in the simulation.

The program also produces a **.EPSR.erg** file which lists the energy, pressure, absolute energy (of the empirical potential only) and chi-square as the simulation proceeds. This is rewound whenever

ireset = 1, 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.

5.6 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 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 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).

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 neccssary. 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:

```
h2o298tot.PARTIALS              Title of this file
fnameato    h2o298tot.ato              Name of .ato file
nz          600               Number of r values [max: 1000]
nsumt       54               Number of configurations already accumulated
ndist       3               Number of site-site distributions
q
     1     OW  OW
     2     OW  HW
     3     HW  HW
```

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 half the box dimension.

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
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

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

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
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.

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:-

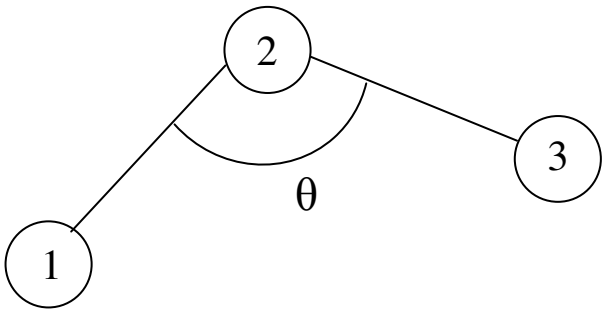


Figure 6.1

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
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
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.

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.

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 2 distance ranges (4 numbers) to be specified. In order to be classified as "bonded" the 1st and 3rd atoms have to be within the 1st specified distance range AND the 2nd and 3rd have to be within the 2nd specified distance 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 [7], 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
rmin1       1               Minimum distance for atom1-atom3
rmax1       3.4               Maximum distance for atom1-atom3
rmin2       1               Minimum distance for atom2-atom3
rmax2       2.5               Maximum distance for atom2-atom3
q
   1  0.17900E+01  0.63511E+00  0.58000E+01  0.48229E+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 cluster 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.

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 the ethanol-water simulation previously:

```
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
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

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      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
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 these quantities 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.

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, similar to that given by **clusters**, **chains**, or **rings**. The volume of the void in this case is expressed in units of the size of the simulation box divided by the number of histogram points, **nsize**.

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

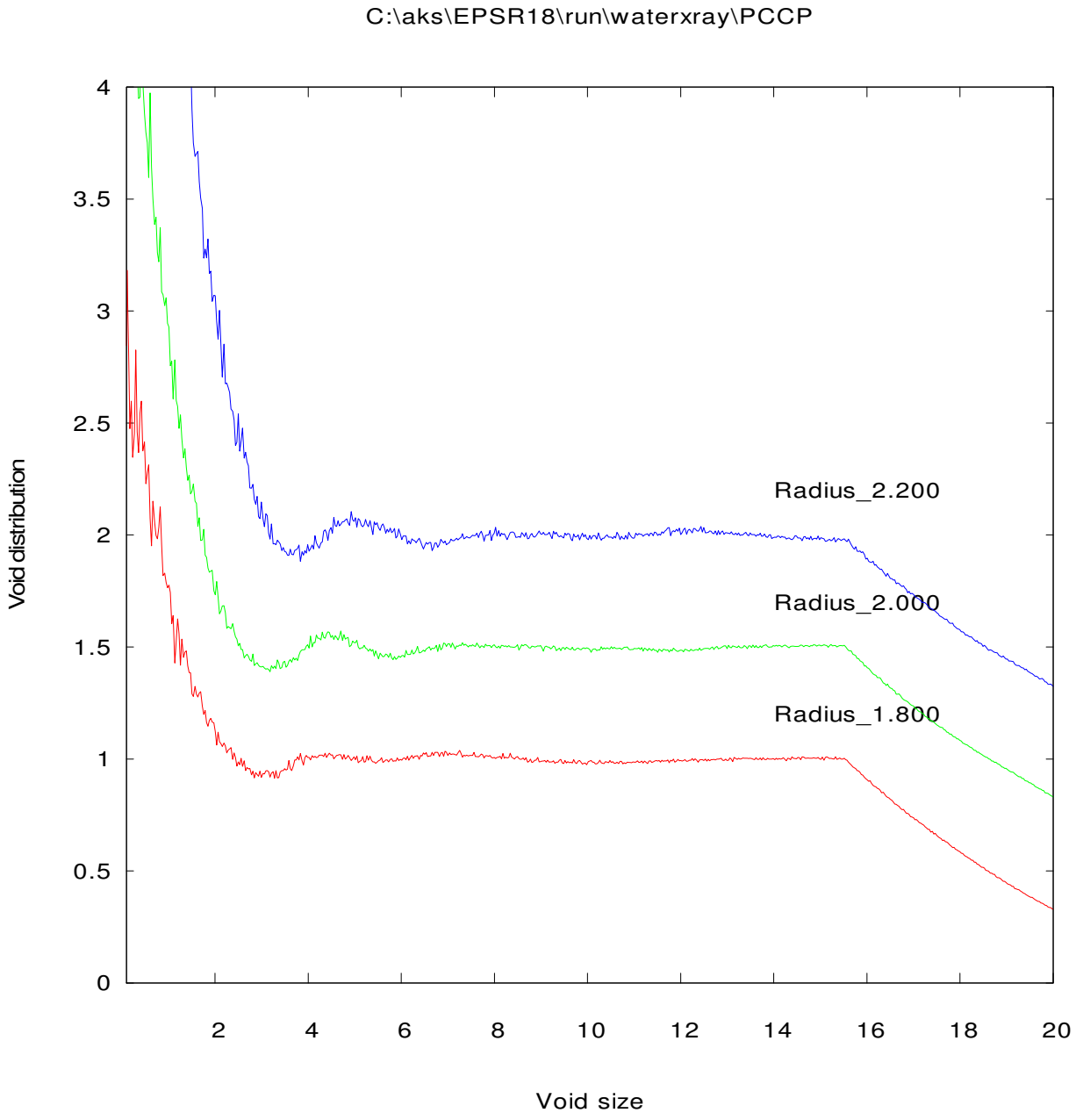
voidmax is not used in the current version of the program

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.

7 D S Franzblau, Phys Rev B, **44**, p4925-4930 (1991)

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 intellectual 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 [8] 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 the Fig 7.1 can help you to see the problem:

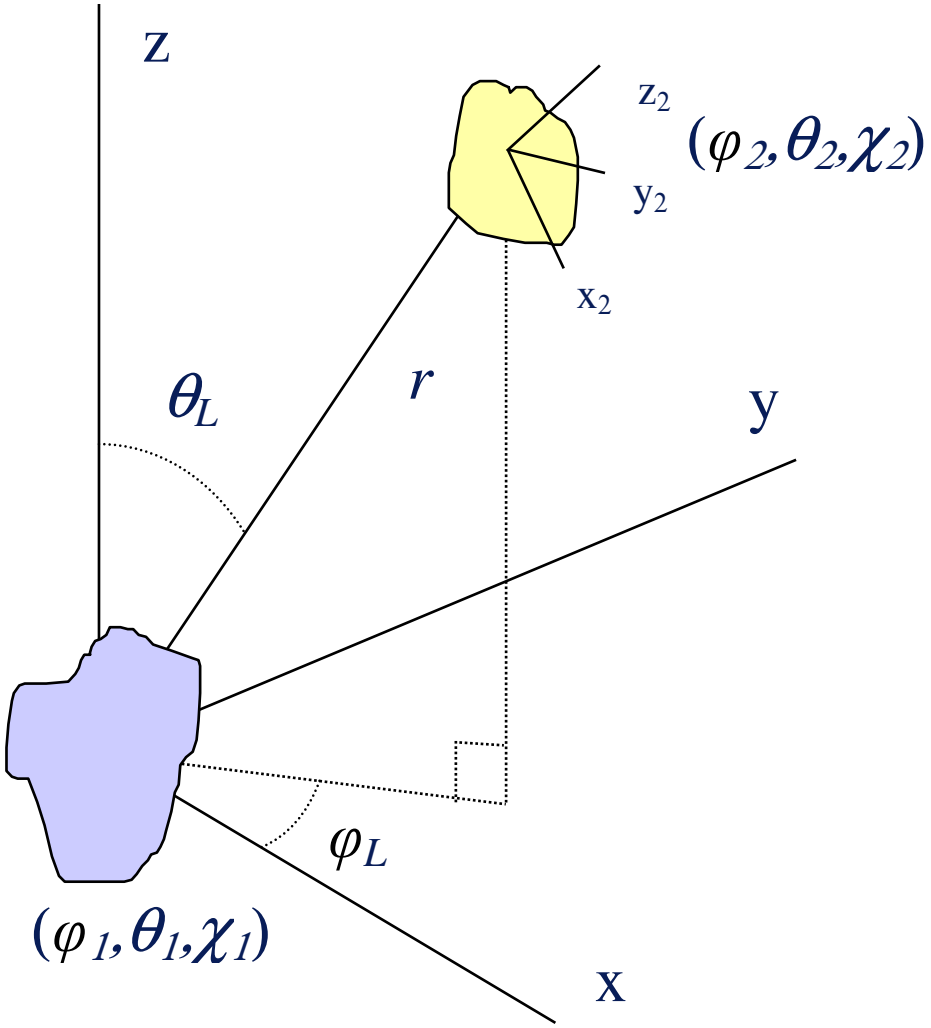


Figure 7.1

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.

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

(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

$$D^{l}_{mn}(\omega) = e^{-im\phi} d^{l}_{mn}(\theta) e^{-in\chi} \tag{7.1.2}$$

are the generalized spherical harmonic functions, with

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

Note that the summation in (7.1.1) differs slightly from that in Gray and Gubbins (eq. 3.142) by a factor of $\left[\frac{(2l+1)}{4\pi}\right]^{1/2}$ 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

$$D^{l}_{mn}(000) = \delta_{mn} \tag{7.1.4}.$$

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:-

$$h(l_1 l_2 l; n_1 n_2; r) = \frac{(2l_1+1)(2l_2+1)}{4\pi(8\pi^2)^2}$$
$$\times \int d\omega_1 \int d\omega_2 \int d\omega \sum_{m_1 m_2 m} h(\mathbf{r},\omega_1,\omega_2) C(l_1 l_2 l; m_1 m_2 m) D^{l_1}_{m_1 n_1}(\omega_1) D^{l_2}_{m_2 n_2}(\omega_2) D^{l}_{m0}(\omega)^*$$

(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 $g(l_1 l_2 l; n_1 n_2; r)$, (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 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:-

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

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[8] 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. 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

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).

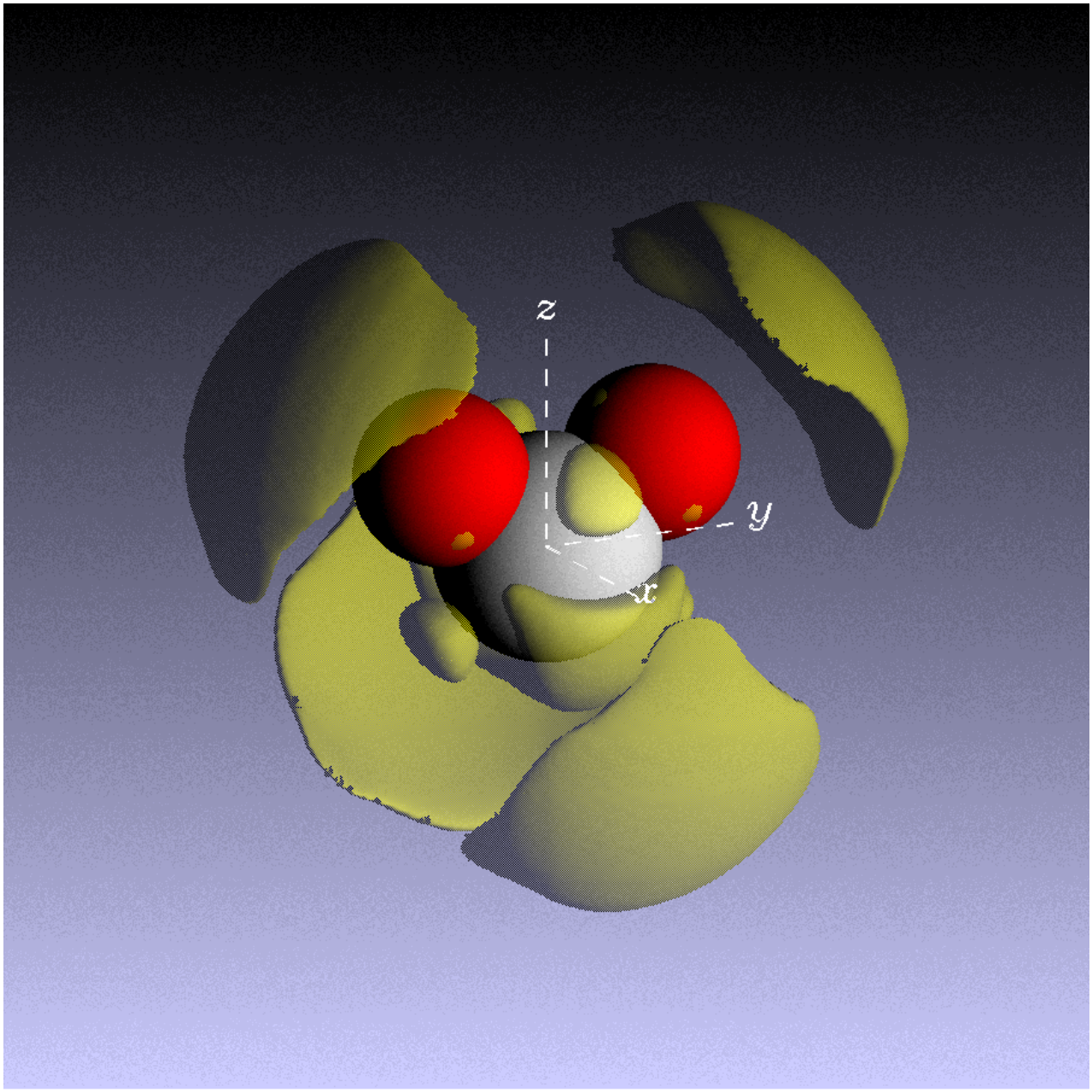


Figure 7.2 Result of running **sdf** on an EPSR simulation of amorphous silica, then plotting the results via **plot3d**. The central "molecule" of SiO_2 is shown as the spheres near the origin of the coordinate system. The local, tetrahedral coordination is very visible in these plots. Perhaps not so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

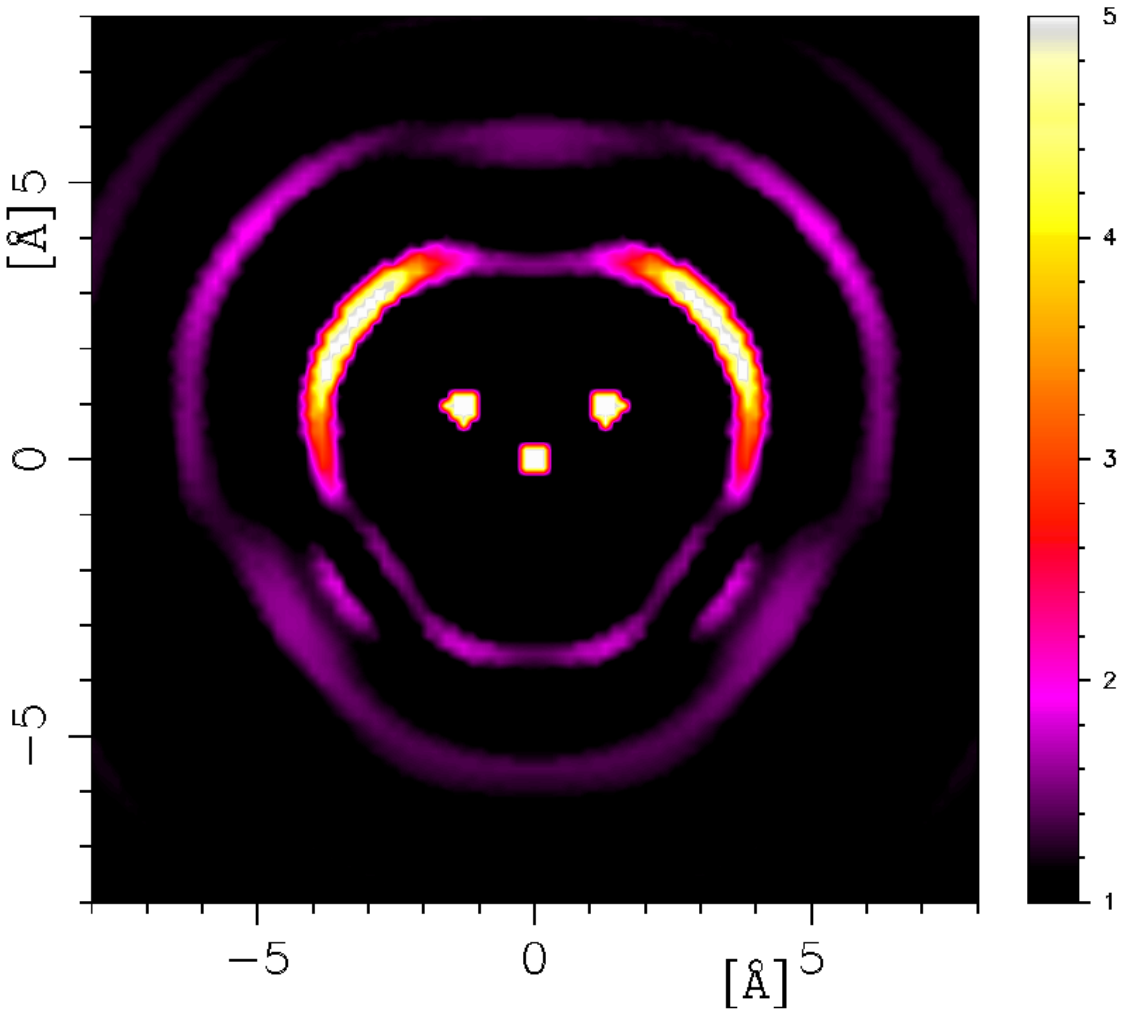


Figure 7.2 Result of running **sdf** on an EPSR simulation of amorphous silica, then plotting the results via **plot2d**. The central "molecule" of SiO_2 is shown as the circles near the origin of the coordinate system. Notice how we get better intensity information from the contour plot, but lose its 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 I M Svishchev and P G Kusalik, *J Chem. Phys.*, **99**, 3049 (1993)

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 **introtcluster**, 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

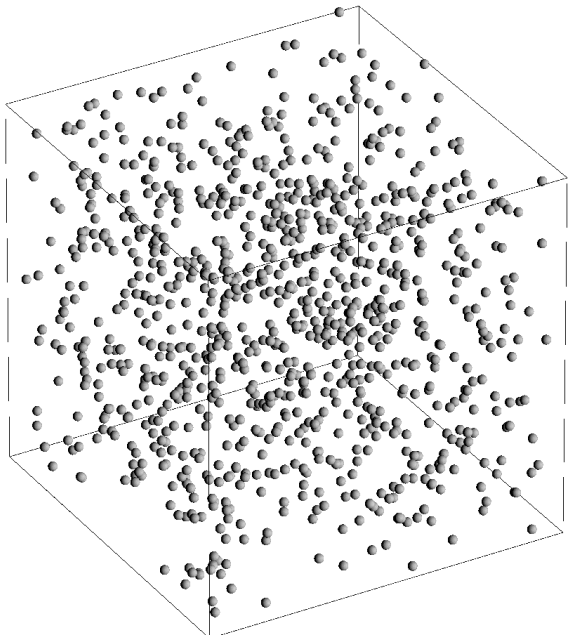
After loading EPSR, and getting to/creating the working folder needed, enter **makeato**. This will guide you through a series of inputs needed to create the atom or molecule you are creating. It will ask you to enter values for the epsilon and sigma. If you don't have these, an educated guess will do. (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 **introtcluster** 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 .doq file in the folder that you are working in, and enter the command 'epsrwts'. Simply follow the instructions to complete this.

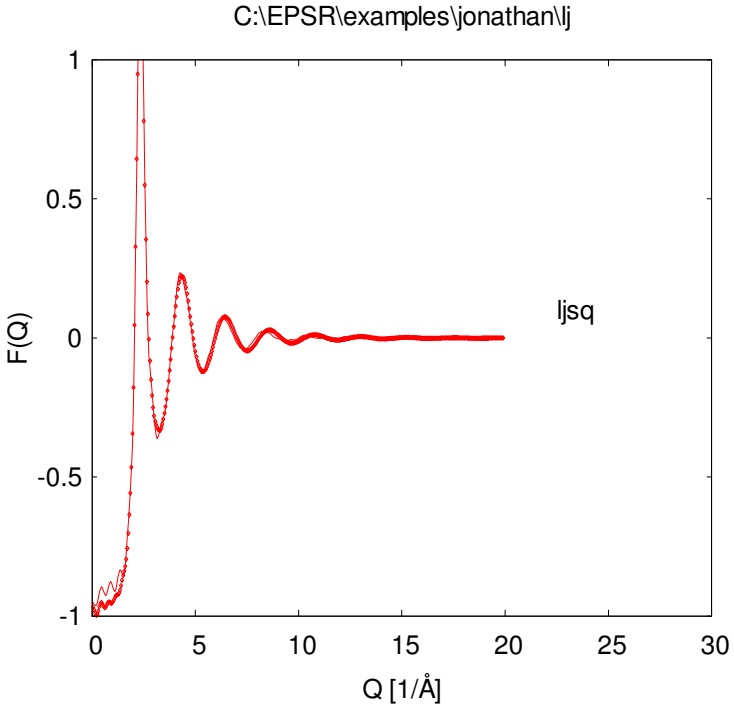
Use the 'setup epsr' command. All of the values already in place are correct, and the only thing needed for input is the undefined entries. These you will need to enter yourself. When it is all set up, save and exit. This will have created a .inp and .pcof file.

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:

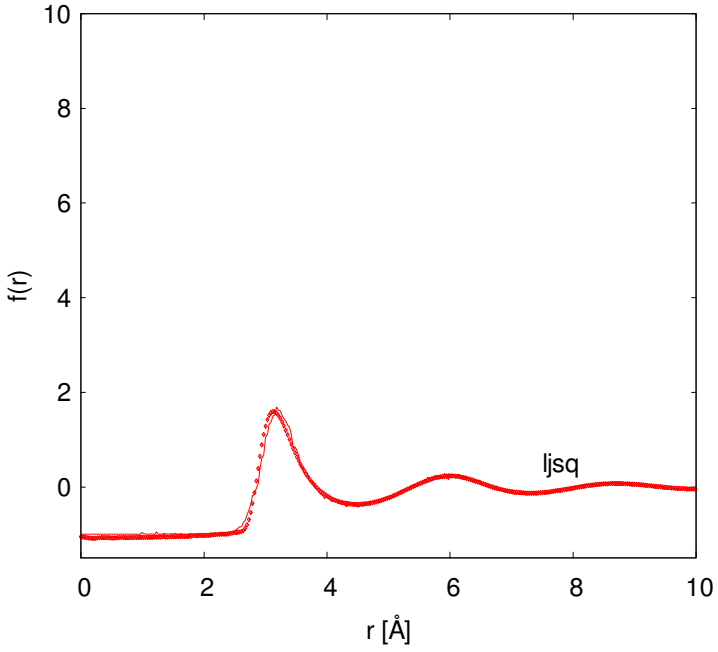
cd .\[name of folder data is in]
epsr [name of .inp file]

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 **introtcluster** 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 **EPSRwts** 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 **EPSRwts** 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 [9]. 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 **introtcluster** 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.

---

9 G Hura, D Russo, R M Glaeser, T Head-Gordon, M Krack, and M Parinello, *Phys. Chem. Chem. Phys.* **5**, 1981-1991 (2003)

9. Appendices

9.1 Files you need to run EPSRshell

The executables for EPSR are normally stored in a special binary folder, \bin, which is itself contained in the main EPSR folder. The current version is EPSR18, so the simplest option is to set up an EPSR folder somewhere in your file system, and copy the entire EPSR18 folder from the download source into that folder. Then set up an environment variable, EPSRroot, to refer to this folder. The distribution includes a version of GNUplot for the Windows version in a separate folder to the main executables, and Jmol.jar, to run the Jmol program, is stored with the other binaries. In addition to get the PGPLOT routines to work, you need to refer correctly to the PGPLOTlib libraries in Windows. For LINUX it will be necessary to have all the appropriate PGPLOT and MOTIF libraries in place before the program can be run. A set of source files is included with the distribution so that compilation (using the makeepsr scripts contained in the top folder) should be straightforward. For LINUX the PGPGLOT routines are compiled using the makefile in the source folder, but for Windows it is currently not possible to compile these routines since the present distribution requires access to the Compaq Visual Fortran compiler.

In order to ensure all the definitions are correct, the simplest option is to put the following EPSRsetup.bat file in your main EPSR18 folder:-

epsrsetup.bat:

```
if defined epsrpath set path=%epsrpath%
set epsrpath=%path%
title EPSRsetup
set EPSRroot=%CD%
set EPSRbin=%EPSRroot%\bin
set EPSRgnu=%EPSRbin%\gnuplot\bin
set PGPLOT_DIR=%EPSRbin%\PGPLOT\PGPLOTlib\
set PGPLOT_FONT=%EPSRbin%\PGPLOT\PGPLOT_LIB\grfont.dat
set path=%PGPLOT_DIR%;%epsrpath%
```

On LINUX an equivalent shell script is created in the top folder, epsrsetup.sh:

```
export EPSRbin=$(pwd)'/bin'
```

Under LINUX environment variables are treated differently from on Windows so the methods are slightly different for the two operating systems. Thus under LINUX the EPSRroot environment variable has to be set in the Bash startup script (typically .bashrc) if it is to be available to other routines.

To run EPSRshell, you go to the folder where you want to run EPSRshell, and paste a copy of the epsr.bat or epsr.sh files into that area and run the script in the appropriate way for each operating system. These files respectively look like:-

```
set currentdir=%CD%
cd %EPSRroot%           Here you need to type the name of the folder where
EPSR18 is stored
call epsrsetup
cd %currentdir%
copy system_commands_windows.txt system_commands.txt
title EPSR in %CD%
%EPSRbin%\epsrshell
```

(epsr.bat)

and

```
currentdir=$(pwd)
cd $EPSRroot
```

```
. $EPSRroot/epsrsetup.sh
cd $currentdir
cp system_commands_linux.txt system_commands.txt
$EPSRbin/epsrshell
```

(epsr.sh).

Both scripts assume the environment variable, EPSRroot, has been set to the top directory where EPSR18 is stored before the scripts are run. This way ensures the correct environment variables are in place to run the programs, but other methods may be appropriate in particular cases. Under LINUX you set the environment variable by having the line

```
export EPSRroot=<top folder where EPSR18 is stored>
```

in the .bashrc that runs the command prompt. If you also have the line

```
alias epsr='sh $EPSRroot/epsr.sh'
```

then it will start EPSRshell in the folder where you type "epsr", always assuming of course that the necessary files (see below) exist in that directory.

For the shell to start correctly, it is necessary the file "**system_commands.txt**" is available in the directory where it is to run. Again there are different versions for Windows and Linux, so the above examples 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_gnu   %EPSRgnu%\wgnuplot.exe
system_jmol  java -jar %EPSRbin%\Jmol.jar
system_binaries %EPSRbin%
system_plot2d  %EPSRbin%\map2dplot.exe
system_plot3d  %EPSRbin%\map3dplotquadrot.exe
system_plotato %EPSRbin%\mypgplotato2006-11-13.exe
system_pgout /GIF
gnu_pointtype 8
gnu_pointsize 0.3
```

while for Linux it looks like:

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

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_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_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 in detail from those used with EPSR17, and so earlier versions cannot be used with EPSR18.

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

epsr.bat or .sh	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	(as a backup)
runepsr.txt	(this is the script file – it can have any name you choose)

Files for the **working** folder:-

plot_defaults.txt	This can be copied from the home folder.