Comparison of model potentials for molecular dynamics simulation of crystalline silica

Dissertation

zur Erlangung des Grades
„Doktor der Naturwissenschaften“

am Fachbereich Physik
der Johannes Gutenberg–Universität
Mainz

vorgelegt von

Daniel Herzbach

geboren in Frankfurt (Main)

Mainz, im Juni 2004

Datum der mündlichen Prüfung: 05.07.2004

Abstract

Simulating SiO_2 with the two-body potential developed by van Beest, Kramers, and van Santen (BKS) produces many satisfactory results but also characteristic flaws. We investigate these failures of the BKS potential and compare the performance of the BKS potential with that of two recently suggested potential energy surfaces that effectively incorporate many-body interactions. One approach, which is called the fluctuating-charge model, allows the ionic charges to adjust depending on the chemical environment. The other approach assumes fixed, effective charges but allows for inducible dipole moments on the oxygen atoms.

The emphasis in this work is placed on situations where BKS fails. We show that an anomaly in the ratio of quartz's two independent lattice constants a and c, which is observed experimentally at the transition between α and β-quartz, is missing with BKS. Cristobalite and tridymite appear mechanically unstable with BKS when periodic boundary conditions are employed that are compatible with competing high-density silica polymorphs. Lastly, the BKS density of states (DOS) shows characteristic discrepancies from the true DOS.

The fluctuating-charge model slightly improves the phononic density of states and correctly produces stable cristobalite, but it does fail to show the experimentally observed c/a anomaly at the α-β transition. Moreover, many properties are reproduced much less satisfactorily than with BKS.

The fluctuating dipole model remedies all mentioned artifacts. We confirm the view that the proper behaviour in the c/a ratio is due to the distortion of SiO_4 tetrahedra. These distortions can in turn be shown to be due to the many-body effects incorporated in the fluctuating dipole potential.

In addition, the pressure-induced phase transition in α quartz is studied. All three models show a transition at similar pressures to the same crystalline phase, which is probably the same as that found experimentally for the high-pressure polymorph called quartz II. On decompression, the BKS potential predicts the formation of an unknown phase at ambient pressures, while in the two other approaches the quartz II phase reverts to α-quartz with an intermediate phase similar to quartz II.

Furthermore we show that the fluctuating-charge potential is the only known model potential to predict the right pressure for a phase transition between two sixfold coordinated stishovite polymorphs.

We suggest two different methods to calculate piezoelectric coefficients in classical molecular dynamics simulation in the constant stress ensemble. We find that BKS reproduces experimental data reasonably well. However, the results for the fluctuating dipole potential turn out to underestimate the experimental data by more than 50%, unless the coupling of the electrical field to the dipoles is switched off. We also show that there is a strong correlation between the magnitude of the dipoles and the bond-bending angles on the oxygen atoms and speculate that quantum chemical effects that are non-electrostatic in nature were (successfully) parametrized as inducible dipoles. With this interpretation, the fluctuating dipole potential always appears to be closest to available experimental data out of the three approaches.

Contents

Introduction

Almost a century ago, the discovery of quantum mechanics allowed one to describe the fundamental interactions of ions and electrons in matter. Even some decades earlier, the equations of statistical mechanics were formulated, which relate the microscopic interactions of ions and electrons to the resulting macroscopic properties of a given material. However it is generally not possible to solve these equations analytically for complex materials in order to predict their properties. Dirac expressed the problem in 1929 as follows: "The fundamental laws necessary for the mathematical treatment of large parts of physics and the whole of chemistry are thus fully known, and the difficulty lies only in the fact that application of these laws leads to equations that are too complex to be solved." This statement still holds despite the increasing power of computers. Simplified models for the interactions need to be formulated for analytical theories. More approximations are required if one wants to predict the behaviour of materials beyond the harmonic approximation. If theory and experiment disagree, it is often difficult to say at what point theory failed.

The use of computers made it possible to reduce this uncertainty, as computer simulations made it possible to solve the equations for microscopic empirical model potentials with (theoretically) arbitrary accuracy, thus allowing us to see how the macroscopic system behaves as a whole without the need to rely on further assumptions. The numerical computation of macroscopic observables can be realized by sampling randomly over phase space in a Monte-Carlo (MC) simulation [58] or by following the trajectory of Newton's equations of motion, and thus sampling over phase space as well, in a molecular dynamics (MD) simulation [28].

It is nevertheless not yet possible to solve the quantum mechanical equations exactly except in an extremely limited number of cases. In fact, one can show that it is beyond any conceivable computational capacity to exactly solve the Schrödinger equation in its standard form for a system containing only ten electrons [84]. However it became possible to handle quantum mechanics in a computer simulation with the development of density functional theory (DFT), which states that it suffices to know the average number of electrons located at any one point in space instead of the motion of each individual electron [70]. The practical application of this theory relies on good approximations for the functional as the true functional is not known in a solid. Here the local density approximation (LDA) has produced very accurate results for a wide

range of materials. In 1985 Car and Parrinello [11] developed a very efficient method to use DFT in the force calculation of an MD simulation, which made this so called Car-Parrinello molecular dynamics (CPMD) one of the most widely used techniques for the calculation of dynamic and electronic properties of condensed matter. This kind of simulations, where a real (but approximative) potential without adjustable parameters is used, is called first-principle or ab-initio simulations.
So these two options, the ab-initio approach and the empirical model potentials, can be implemented in a computer simulation, which involves different possibilities and restrictions respectively. Of course ab-initio techniques lead to much higher chemical accuracy, as the calculation of the interatomic stresses and forces used for propagation of the particles is close to reality. On the other hand it is limited due to its computational cost and thus restricted to small length and time scales. This is a major problem as it severely limits the range of properties that one can calculate and the number of phenomena that can be simulated.
In contrast an empirical model is parametrized to certain experimental properties under certain thermodynamic conditions, and its reliability has to be questioned when it is used away from the conditions under which it was parametrized or if properties have to be calculated that did not enter the parametrization of the adjustable model parameters. But they provide much faster calculation, which is beneficial for the calculation of thermal properties if the relaxation times in the system are large so that thermal averages require sampling over long periods of time. Moreover, in a wide range of applications model potentials have been employed very successfully.
The crucial questions with an empirical potential are to incorporate the relevant physical effects into the formulation of the functional form and to have a proper adjustment of the free parameters. In this work different approaches of model potentials will be tested in the simulation of silica (SiO_2) polymorphs. (Polymorphic materials are materials that possess different crystalline structures for a fixed stoichiometry.) The force fields studied in this work are the BKS two body pair potential [101], a model that allows for fluctuation of charge between the atoms [20] and a third that employs fixed charges but allows for inducible dipole moments on the oxygen atoms [90]. For the problem of the parametrization the authors of the last potential used a new approach, which does not fit the potential to macroscopic experimental values but aims to match the microscopic forces, stresses and energies with values obtained from DFT calculations [25, 57].

Silica or silicon dioxide is one of the most widely and intensively studied of all materials. There are many reasons for this. First of all, it is one of the most common materials in nature. Silicates, which is a compound consisting of silicon and oxygen, one or more metals and possibly hydrogen, make up more than 90% of the minerals in the earth's mantle and crust. Additionally it is used extensively in industrial appli-

cation.An obvious example is that it makes up approximately 75% of the composition of the glass that is used for everything from window panes to optical fibres. It is also used as an insulator or substrate in the semiconductor electronics industry.
On the other hand it is interesting to study as crystalline silica has an extremely rich phase diagram with a large number of polymorphs, shown in Fig. 0.1 and the fluid phase, which remains metastable in the form of silica glass for a long time when cooled down to ambient temperatures. The best known polymorphs are quartz, cristobalite,

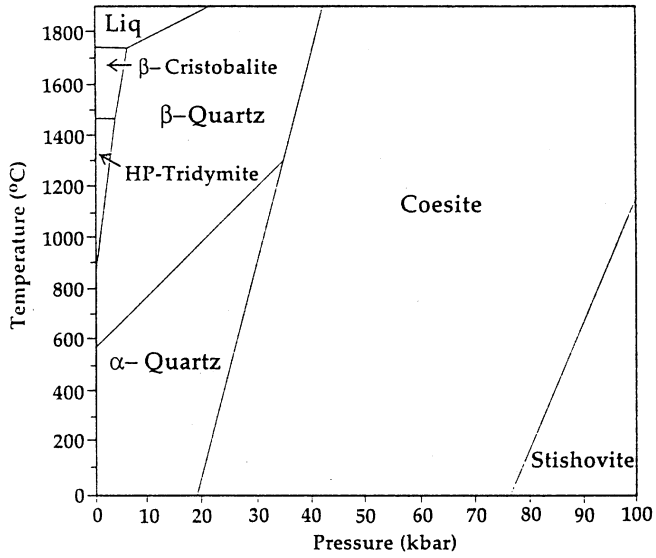


Figure 0.1.: Phase diagram for silica polymorphs. From [54]

tridymite, coesite, and stishovite. The crystalline phases make up a good test case for empirical potentials, as due to the reduced symmetries there are more independent susceptibilities, while in disordered systems effects can be averaged out. Isotropic materials have for instance only two independent elastic constants, while in quartz there are six independent coefficients.

All the low pressure crystal structures are composed of corner-sharing SiO_4 tetrahedra [39] and even in the glass and the liquid, almost all Si atoms are tetrahedrally coordinated. In the higher pressure phases six-fold coordination becomes more and more favorable for the Si atoms, which lead to octahedral coordination in stishovite and other high-pressure phases. The high-pressure phases are of particular interest to understand the earth's interior, where silica is exposed to extreme conditions of temperature and pressure. The high-pressure behaviour of quartz is complicated to investigate in experiment as well as in theory, due to the large number of six-coordinated structures that can possibly form out of four-coordinated structures [52], the slow kinetics, metastabilities and certainly experimental difficulties [52, 105]. As shown in Fig. 0.1,

the stable phases with increasing pressure on quartz would be coesite and stishovite, however these transitions are kinetically inhibited at room temperature. Consequently, quartz can be kept metastable up to 21 GPa. At this pressure, a phase transition occurs. For the new forming phase different crystal structures [38, 95, 96], amorphous structures [40] or partly crystalline structures [52] were proposed experimentally and theoretically. In computer simulations, this phase transition is a good candidate for a reliable model potential, as the system size and the time scales needed to simulate the transition are out of reach for an ab-initio approach. Still the high pressure is not what the model potentials were made for, so the comparison between different approaches increases the credibility in this case.
But also in the quartz phase, which is prevalent at ambient conditions, there are unresolved issues. When heating α-quartz (or low quartz), the crystal turns to β-quartz (or high quartz) at around 573°C [39]. The average structure of both quartz phases has been known for several decades, and the displacive transition between them can be easily understood in terms of tilting of certain groups of SiO_4 tetrahedra. However the real picture of the transition is far more difficult, as is the real structure of β-quartz. It is clear that the oxygen atoms do no fluctuate harmonically around their average positions, but the dynamical disorder that leads to the average structure is still an unanswered problem. There are currently two competing interpretations. One picture of the β-quartz phase sees the oxygens oscillate between two potential minima at the positions of the left-handed and the right-handed α-quartz [107, 87, 100], while the other picture supports an ordered structure that has the atoms vibrate in a librational motion around the mean positions [3, 92, 13, 24, 81, 49]. These two interpretations favor different pictures of the transition, either a disorder-order type transition or a weak first-order transition with a soft mode initiating the transition.
The described dynamical disorder makes the β-quartz phase and obviously the transition difficult to simulate with an ab-initio method. It is definitely not sufficient to find the average structure and extract the relevant properties of β-quartz by harmonic approximations. It was shown that for example the bulk modulus in β-quartz turned out to be twice as large as in experiment with local density approximation and subsequent harmonic approximation [21, 65]. In contrast it is necessary to find the thermodynamic observables by averaging over a long simulation run in order to sample the dynamic disorder (which leads to a bulk modulus comparable to experiment). This makes it necessary to use an empirical model potential. However the standard two-body potential generally employed for the simulation of silica turned out to be flawed in different aspects of the transition. Thus a new potential modelling different microscopic effects could improve the two-body potential in the simulation of the α-β transition.

At the beginning of the thesis in chapter 1 the three potentials used will be introduced and explained, as well as different MD algorithms and techniques used in the simula-

tions.
The following chapter 2 will be concerned with the α-β transition and the performance of the different potentials at this transition. After the determination of the transition temperature, the emphasis is on the course of the lattice constants with temperature. Due to the failure of the BKS potential in this respect, the distortion of the tetrahedra will be investigated in more detail. The phonon density of states and the elastic constants are compared with experiment.
The simulation of the high-pressure post quartz phases in chapter 3 concentrates of the determination and comparison of the transition path ways produced by the different potentials. As available the phases will be compared with experiment in terms of diffraction spectra.
Chapter 4 focuses on an instability of cristobalite in the BKS potential and the comparison to the other models.
As the new potential approaches incorporate electrostatic properties, namely fluctuating charges and fluctuating dipole moments, the investigation of dielectric properties in chapter 5 should highlight the question if these are real electrostatic variables and thus improve the prediction for the dielectric properties of the system. A method is developed to measure dielectric and piezoelectric properties efficiently.
The conclusions at the end will give a final assessment of the suitability of the new approaches for silica simulations.

1. Simulation Techniques and Model Potentials

1.1. Molecular Dynamics Techniques

The simulation scheme being used, a classical molecular dynamics (MD) computer simulation, is a numerical method to calculate the trajectory of a many-atom system. The starting point of an MD simulation is Newton's equations of motion for the system under consideration:

$$m_i \ddot{\underline{R}}_i = -\nabla_i V(\{\underline{R}_j\}) = \underline{F}_i \tag{1.1}$$

where $\underline{R}_i$ are the coordinates of the atoms, m_i their masses, V is the total potential energy and $\underline{F}_i$ the force acting on atom i.
These equations form a system of N coupled partial differential equations. Their solution is the phase space trajectory of the many-particle system. The aim of an MD simulation is to numerically solve the equations of motion and to extract macroscopic observables from the resulting trajectory.
To obtain meaningful time thermal averages for macroscopic quantities, it is important to start with an equilibrated configuration. This ensures that time averages do not change in the course of the simulation. In order to simulate a crystal in thermal equilibrium, one has to start with a configuration being close to the expected crystal structure and to run the simulation until the macroscopic quantities are reasonably stable. Two aspects are particularly important for the simulation of a system. One is the model potential V from which the forces follow, the other aspect is which algorithm is used to integrate the equation of motion and how external imposed conditions such as temperature, stress, or an electric field can be taken into consideration.
A variety of standard methods is available for for numerical details such as the integration scheme. In this work the so-called Gear predictor-corrector algorithm up to fifth order was used for the integration of the equations of motion. [33]. While Gear predictor-corrector algorithm of order higher than two do not conserve energy, they do produce more accurate trajectories than the second order Gear predictor-corrector algorithm, which is also called velocity Verlet. The 2^{nd}-order predictor-corrector is known to be energy and phase space conserving. However in the context of this work this attribute is relevant for testing purposes only, when the correct implementation of

the routines was checked by means of energy conservation within an unthermostatted, i.e. microcanonical ensemble. For succeeding production runs a Langevin thermostat was used, which enables simulations in the canonical ensemble. The thermostat constantly adds and removes energy to the system in order to maintain a constant temperature. This renders the energy conservation of the integrator an unnecessary quality. Also higher-order integration schemes give more accurate averages at a given time step, once the thermostat is introduced, which is why they were used for the production runs.
To allow for simulations in the isothermal-isobaric (NpT) ensemble, the Parrinello Rahman method was used. This method as well as the integrator and thermostat algorithms will be described in detail in the following.
For acceleration of the time consuming force calculation standard binning and Verlet neighborhood list techniques were used [1, 28].

1.1.1. Parrinello-Rahman Barostat

The Parrinello-Rahman method of pressure and stress control allow one to simulate molecular systems under externally applied stress. This is useful for studying the stress-strain relationship of materials and phase transitions with non-isotropic shape variations. Both the shape and the volume of the cell can change, so that the internal stress of the system can match the externally applied stress. In this work only the presence of uniform pressure is needed, so it is not necessary to allow for an arbitrary stress tensor. It is yet important to allow for an arbitrary strain.
The method was presented in detail by Parrinello and Rahman [71] and is only summarized here. The fluctuating shape of the cell is given by the cell matrix $\underline{\underline{h}}$ containing the cell vectors so that the real space coordinates $\underline{R_j}$ of a particle j are related to the reduced coordinates $\underline{r_j}$ by $\underline{R_j} = \underline{\underline{h}}\,\underline{r_j}$. While the coordinates $\underline{R_j}$ represent the real positions of the atoms, the reduced coordinates $\underline{r_j}$ only live in a unit cube.
The Langrangian of the system is modified such that a term representing the kinetic energy of the cell depends on a user-defined mass M_{box}. An elastic energy term $p \det \underline{\underline{h}}$ is related to the pressure p and the volume $\det \underline{\underline{h}}$ of the system. These two terms lead to an extended Lagrangian:

$$L = \frac{1}{2}\sum_{i=1}^{N} m_i \underline{\dot{r}}_i' \underline{\underline{h}}' \underline{\underline{h}} \underline{\dot{r}}_i - V_{\text{interaction}} + \frac{1}{2} M_{\text{box}} \text{Tr} \dot{\underline{\underline{h}}}' \dot{\underline{\underline{h}}} - p \det \underline{\underline{h}} \tag{1.2}$$

The equations of motion for the reduced coordinates and the cell vectors can be derived from this Lagrangian by using the regular Lagrange formalism. The motion of the cell vectors, which determine the cell shape and size, is driven by the difference between the target and the internal stress.

The choice of the fictitious mass M_{box} is governed by the following considerations. A large M_{box} means a heavy, slow cell. In the limiting case, infinite M_{box} reverts to constant-volume dynamics. A small M_{box} means fast motion of the cell vectors. Although the target stress can be reached faster, a too small mass can result in artifical periodic movements of the cell and requires a smaller time step. As the $\underline{\underline{h}}$ matrix is kept symmetric, it can be written in Voigt notation [102], which is a standard to write symmetric crystallographic 3×3 tensors as a 6 dimensional vector with entries $\underline{\underline{h}} = (h_{xx}, h_{yy}, h_{zz}, h_{xz}, h_{yz}, h_{xy})$. Once all equations of motion are derived, a thermostat can be added to them. This can be done for the cell vectors in a similar way as for the reduced atomic coordinates.

1.1.2. Gear Predictor Corrector Integrator

Let $\underline{r}_i(t)$ be the position of particle i at time t. First, in the predictor step the positions and derivatives of the particle positions are predicted,

$$\frac{(\Delta t)^n}{n!}\frac{\partial^n}{\partial t^n}\underline{r}_i(t+\Delta t) = \frac{(\Delta t)^n}{n!}\frac{\partial^n}{\partial t^n}\underline{r}_i(t) + \sum_{m=n+1}^{N_{\text{order}}} C^{\text{p}}_{n,m}\frac{(\Delta t)^m}{m!}\frac{\partial^m}{\partial t^m}\underline{r}_i(t)\,,\ n = 0, \ldots, N_{\text{order}}\,, \tag{1.3}$$

where $C^{\text{p}}_{n,m}$ is the m-th predictor coefficient of n-th order. The predictor coefficients are just the Taylor expansion coefficients

$$C^{\text{p}}_{n,m} = \frac{m!}{n!(m-n)!}\,. \tag{1.4}$$

For efficiency, the powers of the time-step and faculties are saved on the arrays holding the derivatives of the positions. When configurations are written out, they are however divided out, to obtain the time-derivatives of the positions independently of the time-step. They can be used to propagate saved configurations with another time-step Δt. The force calculation in the following MD step is now based on these predicted positions. After this, the propagation step is completed by the correction step. In this step, positions and derivatives are corrected proportionally to the difference between computed and predicted force (or equivalently acceleration),

$$\frac{(\Delta t)^n}{n!}\frac{\partial^n}{\partial t^n}\underline{r}_i(t+\Delta t) := \frac{(\Delta t)^n}{n!}\frac{\partial^n}{\partial t^n}\underline{r}_i(t+\Delta t) + C^{\text{c}}_n\frac{(\Delta t)^2}{2}\left(\frac{\underline{f}^{\text{d}}_i(t+\Delta t)}{m} - \frac{\partial^2}{\partial t^2}\underline{r}_i(t+\Delta t)\right)\,. \tag{1.5}$$

Here, C^{c}_n is the n-th corrector coefficient. Gear devised the correction coefficients to minimize errors such that the local truncation error is of $\mathcal{O}(\Delta t^{N_{\text{order}}+1})$ for linear differential equations [33]. For second-order differential equations the global error is then $\mathcal{O}(\Delta t^{N_{\text{order}}-1})$. The corrector coefficients are listed in table 1.1.2 following Ref. [36].

	$N_{\text{order}} = 2$	$N_{\text{order}} = 3$	$N_{\text{order}} = 4$	$N_{\text{order}} = 5$
C_0^{c}	0	1/6	19/120	3/16
C_1^{c}	1	5/6	3/4	251/360
C_2^{c}	1	1	1	1
C_3^{c}	–	1/3	1/2	11/18
C_4^{c}	–	–	1/12	1/6
C_5^{c}	–	–	–	1/60

Table 1.1.: Corrector coefficients for the Gear predictor corrector algorithm.

1.1.3. Langevin Thermostat

The Langevin thermostatting routine couples the system via friction and noise to a heat bath. In doing so, the simulation, which was based on Newton's equation up to now, now integrates a Langevin equation

$$m_i \ddot{\underline{R}}_i = \underline{F}_i - m\gamma \dot{\underline{R}}_i + \underline{F}_i^r. \tag{1.6}$$

Therefore this method is also referred to as "stochastic dynamics". The friction constant γ controls how fast the system relaxes into equilibrium. The simulation scheme now produces states which are distributed according to the canonical ensemble, and together with the barostat even the isobaric-isothermal ensemble. The temperature results as the ratio of noise strength to friction, via the fluctuation-dissipation theorem. The stochstic force $\underline{F}_i^r$ is a random variable with mean zero and a width σ^{r} chosen to satisfy the fluctuation-dissipation theorem [1]:

$$\sigma^r = \sqrt{3}\sqrt{\frac{2Tmk_{\text{B}}\gamma}{\Delta t}} \tag{1.7}$$

The uniform distribution is prefered to the Gaussian distribution for efficiency and stability reasons, as the details of the distribution function do not matter. The friction constant γ is set to 0.1 in almost all simulations unless noted otherwise.
The thermostat was used also in production runs, even though an equilibrated configuration should not change in temperature, when an energy conserving integration scheme is employed. Nevertheless the thermostat decorrelates the system and is even more stable than simple microcanonical MD with the Verlet algorithm, because the stochastic dynamics thermostats every degree of freedom individually. As the stochastic force individually heats or cools single particles, it can prevent particles with a kinetic energy much higher or lower than the thermal average from causing instabilities. This is in contrast to the Nose-Hoover thermostat [69], in which only the overall system is thermostatted.

1.1.4. Interactions

Interaction forces are calculated using the positions from the predictor step, including the random contribution. In all models being used the leading order in the forces are given by pair potentials. In the case of the fluctuating-charge potential these forces also depend, via the charges, on the positions of all the other atoms, and are therefore many body interactions. In the case of the fluctuating-dipole potential the two-body forces are accomplished by additional dipole interactions. The values of the dipoles depend on all the other dipoles, and therefore the interactions between the dipoles represent a many body contribution.

1.1.5. Ewald Summation

All the potentials dealt with in this work contain a Coulomb interaction part. As with all the other parts in the potentials, the calculation of the total energy of a three-dimensional system theoretically involves the evaluation of interactions between all species within the unit cell and their periodic replications. Even when the interaction was evaluated for an infinite number of replications, the electrostatic interaction would not be well defined as it still depends on how the limit is carried out. One conisitent way is to sum up $\exp(-r/a)/r$ to infinity before taking $a \to \infty$. Nevertheless in numerical computations some finite cutoff must be placed on the interactions. For those interactions decaying quickly with distance, the summation can normally be readily converged directly in real space. For long-range interactions however the convergence might be not so quick, particularly since the number of interactions increases with r^2. For the electrostatic energy in particular, the number of interactions increases more rapidly with distance than the potential, which is proportional to $1/r$, decays. An accurate evaluation can be achieved through the Ewald summation [26, 93] in which the inverse distance is rewritten as its Laplace transform and then split into two rapidly convergent series, one in reciprocal space and one in real space. The distribution of the summation between real and reciprocal space is controlled by a parameter α. The resulting expression for the energy is:

$$V^{rec} = \sum_{\substack{j>i \\ R_{ij}<r_{\text{cut}}^{\text{ew}}}} \frac{Q_i Q_j e^2}{R_{ij}} \text{erfc}(\alpha R_{ij}) - \frac{\alpha}{\sqrt{\pi}} \sum_j q_j^2 e^2 \tag{1.8}$$

$$+ \frac{2\pi}{V} \sum_{\substack{\underline{q}=2\pi \underline{\underline{h}}^{-1}(n_x,n_z,n_z) \\ 0<q\leq q_{\text{cut}}}} \frac{\exp\left(-\left(\frac{q}{2\alpha}\right)^2\right)}{q^2} \left| \sum_j Q_j e \exp\left(2\pi i \underline{R}_j \cdot \underline{q}\right) \right|^2 \tag{1.9}$$

The cutoffs $r_{\text{cut}}^{\text{ew}}$ in real space and q_{cut} in reciprocal space can be set in dependency of α in a way that the error is minimized:

$$r_{\text{cut}}^{\text{ew}} = s/\alpha, \qquad q_{\text{cut}} = (2s\alpha)^2 \tag{1.10}$$

A value of $s = 3$ has proven to be a good compromise between accuracy and speed. For the distribution between calculations in real and reciprocal space, a good balance can be found at about $\alpha = 0.3$ for system sizes of around 1000 particles which were considered in the present work. Both of these values were used throughout the simulations.

1.2. BKS Potential

The BKS Potential was proposed in 1990 by van Beest, Kramer and van Santen [101] and has been succesfully applied to a range simulations involving crystalline and vitreous silica since [16, 68, 46, 103, 48, 91]. It consists of pure pair interactions with different strength for different pairs of atom types. By combining suitable attractive and repulsive forces for the different atom types it is possible to reproduce effects that are usually caused by many-body interactions. So the BKS potential causes the stable formation of SiO_4 tetrahedra without explicitly caring for the effects of covalent bonding.
The total potential energy is thus a sum of pure pair potentials

$$V(\{\underline{R}_i\}) = \sum_{i<j} \phi_{ij}(R_{ij}). \tag{1.11}$$

The pair potential ϕ_{ij} is composed out of a long-range Coulomb term and a short-range Buckingham term:

$$\phi_{\alpha\beta}(R) = \frac{q_\alpha q_\beta e^2}{R} + A_{\alpha\beta} e^{-B_{\alpha\beta}R} - \frac{C_{\alpha\beta}}{R^6}. \tag{1.12}$$

The charges are fixed on the values $q_{Si} = 2.4$ and $q_O = -1.2$, while the parameters A, B and C in the Buckingham interaction were optimized for the different pairs of atom types with respect to ab intio calculations as well as to macroscopic experimental quantities like the elastic constants of α-quartz. The parameters given in ref. [101] are listed in table 1.2. The Si-Si interaction is assumed to be purely Coulombic in the BKS potential, which is why the Buckingham parameters are all 0 for Si-Si interaction.

	$A_{\alpha\beta}[\mathrm{eV}]$	$B_{\alpha\beta}[\mathrm{\AA}^{-1}]$	$C_{\alpha\beta}[\mathrm{eV\AA}^{6}]$
O-O	1388.7730	2.76000	175.0000
Si-O	18003.7572	4.87318	133.5381

Table 1.2.: BKS Parameters

1.3. Fluctuating Charge Potential

A fluctuating charge potential is conceived as a model potential that makes use of variable charges on the atomic sites when calculating the Coulomb interaction. As the total charge of the system needs to be conserved, this charge adjustment may be understood as relocation of charges within the system and indeed it implies a change in the polarisation of the simulation box. This explains the common notation "polarizable potential", which will not be used in the following for better distinction between fluctuating charge and fluctuating dipole potentials.
The purpose of fluctuating point charges in the simulation is to implicitly account for electronic degrees of freedom. The effective charge of an atom in a fixed charge simulation can be chosen with respect to its type and ionization, but it cannot change in response to changing electrostatic fields which arise from movement of the atoms during the simulation. Thus the charges used in simulations based on fixed charge force fields must reflect average charge values for the particular phase and are in general not transferable to different thermodynamic states. The fluctuating charge approach, in contrast, adjusts the charge according to the instantaneous configuration of the surrounding atoms. Therefore it should be more adaptable to different phases and especially to varying coordination numbers than a fixed charge approach.

1.3.1. Charge Equilibration Algorithm

The basic concept used for the fluctuating charge approach is the concept of electronegativity equalization[82] according to which charge is transferred between atomic sites in such a way that electronegativities are equalized. In doing so, the electronegativity is dependent on the atom type and charge as well as the electronegativities of the neighboring atoms.
Following the Mulliken definition, the electronegativity of an isolated atom i is the negative of the chemical potential (μ_i) of the electron gas surrounding the nucleus:

$$\chi_i = -\mu_i = -\frac{\partial U}{\partial N_i} = e\frac{\partial U}{\partial Q_i} \tag{1.13}$$

In this equation, use had been made of the fact that the ground state energy E, the number of electrons N in the atom and the charge on the atom Q fulfill the equation $Q = -e(N - Z)$, where Z is the atomic number of the atom.
The energy of an isolated atom dependent on the effective charge can be expanded around a neutral reference point as

$$E_i(Q) = E_{i0} + Q_i \left(\frac{\partial E}{\partial Q}\right)_{i0} + \frac{1}{2} Q_i^2 \left(\frac{\partial^2 E}{\partial Q^2}\right)_{i0} + \dots \tag{1.14}$$

Neglecting higher-order terms, one can ask for the energy needed to remove or add a single electron. In units of e and with $E_{i0} = 0$ one obtains

$$E_i(+1) = E_{i0} + \left(\frac{\partial E}{\partial Q}\right)_{i0} + \frac{1}{2} \left(\frac{\partial^2 E}{\partial Q^2}\right)_{i0} \tag{1.15}$$

$$E_i(-1) = E_{i0} - \left(\frac{\partial E}{\partial Q}\right)_{i0} + \frac{1}{2} \left(\frac{\partial^2 E}{\partial Q^2}\right)_{i0}. \tag{1.16}$$

As $E_i(+1)$ is just the ionization potential IE and $E_i(-1)$ is the electron affinity EA, one finds for the first coefficients exactly the Mulliken definition of electronegativity

$$\left(\frac{\partial E}{\partial Q}\right)_{i0} = \frac{1}{2}(IE + EA) = \chi_i^0. \tag{1.17}$$

The second coefficent turns out to be a quantity called idempotential, representing the self repulsion between two electrons:

$$\left(\frac{\partial^2 E}{\partial Q^2}\right)_{i0} = (IE - EA) = J_{ii}^0 \tag{1.18}$$

This leads to the following expression for the charge dependence of the atomic energy, where the parameters χ_i and J_{ii}^0 can be derived from atomic data:

$$E_i(Q) = \chi_i Q_i + \frac{1}{2} J_{ii}^0 Q_i^2 \tag{1.19}$$

The total energy in the system is obtained by adding this energy to the interaction energy, which is given by a two body term Φ consisting of a Coulomb interaction part and a short range non-Coulombic interaction part.

$$\begin{aligned} U(\{Q_i\}, \{\underline{R}_i\}) &= \sum_i E_i(Q_i) + \sum_{i<j} \Phi_{ij}(\underline{R}_{ij}) \\ &= \sum_i \left[\chi_i Q_i + \frac{1}{2} J_{ii}^0 Q_i^2\right] + \sum_{i<j} \left[J_{ij}(\underline{R}_{ij}) Q_i Q_j + V(\underline{R}_{ij})\right] \\ &= \sum_i \chi_i^0 Q_i + \frac{1}{2} \sum_{i,j} J_{ij}(\underline{R}_{ij}) Q_i Q_j + \sum_{i<j} V(\underline{R}_{ij}) \end{aligned} \tag{1.20}$$

Here the Coulomb interaction was assumed to obey $J_{ii}(R) \rightarrow J_{ii}^0$ as $R \rightarrow 0$[1]. This means that $J_{ij}(R)$ does not represent a pure $1/R$ Coulomb potential but has to be a shielded potential with a finite value at $R = 0$.
After all, we end up with an equation that relates the total energy to the positions and charges of all the atoms in the system. Equations (1.13) and (1.20) can be used in two different ways to find the right charges:

- treat the charges as parameters[77]: directly solve for the exact charges given by the spatial configuration in every MD step, or
- treat the charges as generalized coordinates[78]: extend the Lagrangian for the charges, propagate them in the same way as spatial coordinates and thermostat them at 0 K.

A third possibility is to solve the equations selfconsistently. This solution would give the possibilty to control the characteristics of the charge trajectory between the one which is always in the energy minimum, like in the first case, up to a retarded behaviour, like in the second case. Nevertheless it turned out that this solution was difficult to control numerically and converged very slowly. As there are no obvious differences in the trajectories of the two solutions introduced above, this third possibility was skipped.

1.3.2. Direct solution

A set of charges forms the ground state of a given spatial configuration when the chemical potentials on all atom sites are equal. This implies the $N - 1$ conditions

$$\chi_1 = \chi_2 = \cdots = \chi_N. \tag{1.21}$$

Adding the condition on charge neutrality

$$\sum_{i=1}^{N} Q_i = 0 \tag{1.22}$$

leads to a total of N equations for the N equilibrium charges. In order to express the electronegativity in terms of the charges, one has to insert expression (1.20) into eq. (1.13):

$$\chi_i = \frac{\partial U}{\partial Q_i} = \chi_i^0 + J_{ii}^0 Q_i + \sum_{j \neq i} J_{ij} Q_j \tag{1.23}$$

[1] In Ref. [77] one can actually recalculate analytically that this condition is not fulfilled. To obtain better fits to dipole moments, it was dropped by the authors[76].

Equations (1.21) and (1.22) can be merged into a matrix equation

$$\underline{\underline{C}} \cdot \underline{Q} = -\underline{D}, \tag{1.24}$$

where

$$D_i = \chi_i^0 - \chi_1^0 \tag{1.25}$$

and

$$C_{1i} = 1 \tag{1.26}$$
$$C_{ij} = J_{ij} - J_{1j} \tag{1.27}$$

Solving these linear equations with standard procedures leads to the sought-after charges. A more elaborate version of this matrix including all terms from Ewald summation can be found in appendix A.1.1

1.3.3. Extended Lagrangian

The idea for treating charges as generalized coordinates is based on an extended Lagrangian approach. While a charge dependent potential energy is obviously available (eq. (1.20)), one has to define additionally a kinetic energy expression. This is done by assigning a fictitious mass to the charges. The choice of an appropriate value will be discussed later on. Again the charge neutrality has to be obeyed, and the simplest way to implement this is to treat the charges as independent and use a Lagrange multiplier to enforce the constraint. The Lagrangian is

$$L = \sum_i \frac{1}{2} m_i \dot{R}_i^2 + \sum_i \frac{1}{2} M_Q \dot{Q}_i^2 - U(\{Q_i\}, \{\underline{R}_i\}) - \lambda \sum_i Q_i \tag{1.28}$$

While the Euler Lagrange equation for the positions remains unchanged, one gets an additional expression for the charges:

$$M_Q \ddot{Q}_i = -\frac{\partial U}{\partial Q_i} - \lambda \tag{1.29}$$

As the total charge is zero, the expression $\sum_i \ddot{Q}_i$ also vanishes. Therefore

$$\sum_i M_Q \ddot{Q}_i = -\sum_i \frac{\partial U}{\partial Q_i} - N\lambda = 0 \Longrightarrow \lambda = -\frac{1}{N} \sum_i \frac{\partial U}{\partial Q_i}, \tag{1.30}$$

which defines the Lagrange parameter. Substitution into the equation of motion yields

$$M_Q \ddot{Q}_i = -\frac{\partial U}{\partial Q_i} + \frac{1}{N} \sum_j \frac{\partial U}{\partial Q_j} = -\frac{1}{N} \sum_j \left(\frac{\partial U}{\partial Q_i} - \frac{\partial U}{\partial Q_j} \right) \tag{1.31}$$

The descriptive interpretation of this equation is that the electrons are attracted by atoms with higher electronegativity, which complies with the expected behaviour.
This equation of motion can now be propagated by the MD integrator, provided that parameters M_Q for the inertia of the charges and γ_Q for the friction in the thermostat have been defined.
The choice of these parameters is detailed below for the present case. The general consideration is to separate the time scales of ionic motion and electronic motion in order to suppress coupling between internal charge modes and ionic motion. This is a similar idea as in the Car Parrinello approach, where as well a ficitious mass is assigned to the electronic orbitals, which should be small enough that the motion of the orbitals will be very fast relative to the motion of the ions. This way it is ensured that the frequency spectra of the electronic orbitals and the ions are well separated from one another.

1.3.4. Morse-Stretch QEq Potential

The potential that we used for MD simulations was parametrized for SiO_2 by Demiralp et al.[20]. In this model the fluctuating charge was implemented through the charge equilibration (QEq) procedure developed by Rappé and Goddard[77], which features a direct matrix diagonalization approach. In agreement with Rappé and Goddard the QEq parameters were choosen as shown in Table 1.3. Following Ref. [77],

	$\chi(\mathrm{eV})$	$J(\mathrm{eV})$	$R^{\mathrm{QEq}}(\mathrm{Å})$
O	8.741	13.364	0.669
Si	4.168	6.974	1.176

Table 1.3.: QEq parameters for O and Si

the shielded Coulomb interaction $J_{ij}(R)$ is taken to be the Coulomb overlap integral between Slater orbitals centered on each atomic site

$$J_{ij}(R) = \int d\underline{R}_i \int d\underline{R}_j \left|\phi_{n_i}(R_i)\right|^2 \frac{1}{\left|\underline{R}_i - \underline{R}_j - \underline{R}\right|} \left|\phi_{n_j}(R_j)\right|^2 . \tag{1.32}$$

The Slater orbitals are given by

$$\phi_{n_i}(R) = A_i R^{n_i - 1} e^{-\zeta_i R} \tag{1.33}$$

and are determined by the principal quantum number n_i and an exponent ζ_i. This exponent is given by $\zeta_i = 0.4913 \cdot (2n_i + 1)/(2R_i^{\mathrm{QEq}})$, with R^{QEq} from the table above[2].

[2]The differing formula for ζ given in ref. [20] is not correct[19].

A detailed description of how to solve the two-center overlap integral can be found in appendix A.1.2.
The $J(R)$ interaction models a shielded Coulomb potential that is asymptotically a $1/R$ point-charge Coulomb interaction, when the particles are at a sufficient distance, but approaches a finite value for $R \to 0$.
The non-electrostatic short-range interactions are modelled by a Morse-Stretch (MS) term

$$U_{ij}^{\mathrm{MS}}(R_{ij}) = D_0 \left[e^{\gamma(1-R_{ij}/R_0)} - 2e^{\gamma/2(1-R_{ij}/R_0)} \right]. \tag{1.34}$$

The short range interaction is truncated at 9.0Å. The MS parameters were optimized by Demiralp et al. to describe properties like density, cohesive energy, elastic moduli of α quartz and stishovite. The values can be found in table 1.4. It is of course always

	R_0(Å)	D_0(kcal/mol)	γ
O-O	3.7835	0.5363	10.4112
Si-Si	3.4103	0.2956	11.7139
Si-O	1.6148	45.9970	8.8022

Table 1.4.: Morse Stretch parameters

assumed implicitly that the potential energy surface used to find the optimum charges, by setting $\partial V/\partial Q_i = 0$ at fixed values of $\underline{R}$ with the constraint $\sum_i Q_i = 0$, should be the same as the potential energy surface used to calculate the forces $\partial V/partial\underline{R}_i$ at fixed values of Q.
As the description of the energy surface in Ref. [20] does not state anything different, we naturally assumed the only mentioned electrostatic potential energy in this paper, which is the $J(R)$ given by (1.32), as the one which is consistently used. When we implemented the potential as stated by the authors, none of their results could be reproduced, and even the nearest-neighbor peak of the Si-O pair correlation function in the α-quartz structure deviated significantly from the experimentally known 1.6Å. This flawed behaviour was exactly reproduced when we employed the MD simulation program GULP[30, 29, 31] for comparison, which was provied free of charge by the author Julian D. Gale.
The reason for these discrepancies finally turned out be an inconsistent treatment of the potential energy, which was disguised in the publication. Even though they never comment on this in their paper or in private communication, Demiralp et al. did not use the shielded Coulomb potential $J_{ij}(R_{ij})$ for the ion interaction. Instead they used a pure $1/R_{ij}$ Coulomb potential. The only verification for this fact was that only when we tried this inconsistent treatment, we could reproduce the results for the equation of state in α-quartz as they published it in the paper.
So effectively they used the shielded Coulomb potential to calculate the charges as

overlapping Slater orbitals and subsequently used these values as point charges with a pure Coulomb potential in the MD step. Obviously this was done because in the commercial software package they used the correct electrostatic energy functional and its derivatives have not been impelemented, thus the software was only capable to use the point-charge interaction.
A hint to this inconsistency and an attempt to correct it was made recently in a later publication by Sefcik et al.[85], where the Morse Stretch parameters where optimized for a consistent potential energy function. Unfortunately α-quartz turned out to be unstable with this new parameter set in our simulations at ambient temperature. The crystal immediately transformed to β-quartz at temperatures well within the stability range of α-quartz. Because of this flawed behaviour of the new parameter set, which renders it useless for our simulations, we do not consider it in this work, but only use the inconsistent original potential, repectively the modification of it described in the next section.

1.3.5. Alternative Morse-Stretch fluc-Q Potential

Using different potentials for charge equilibration and force calculations makes it impossible to use the extended Lagrange method. Moreover, it is an unphysical treatment to treat these two potentials inconsistently. However, as the results presented in Ref. [20] seemed to be close to experimental data, it appeared to be useful to alter the present potential such that experimental data was reproduced and that charge equilibration and force calculations were based consistently on the same energy surface. To do this, an energy functional was constructed in a way that its derivatives with respect to position and charge should be as close as possible to the derivatives of the two different energy functionals used in ref. [20]. For this purpuse, both fluctuating charges and fixed charges were included in this potential. $J_{ij}(R_{ij})$ was used as Coulomb potential and a term $q_i q_j[J_{ij}(R_{ij}) - 1/R_{ij}]$, with q the average charge in α quartz, was added to the MS potential term. Using this modified potential energy within the extended Lagrangian formalism, the derivation with respect to the dynamical charges will only take notice of the $J_{ij}(R_{ij})$ part, but for the derivation with respect to the positions and therefore the ion motion the $1/R$ is dominant. This modification is of course purely heuristic, but it reproduces nicely the structure of different SiO_2 polymorphs, as shown for the $g_{\mathrm{SiO}}(R)$ in α-quartz in Fig. 1.1 and it reproduces the original data as well. Thus, while this procedure does not necessarily yield the optimum values for the interaction parameters, it seems as though it allows one to analyze at least qualitatively if not semi-quantitatively what benefits one can expect from a fluctuating charge model in SiO_2.

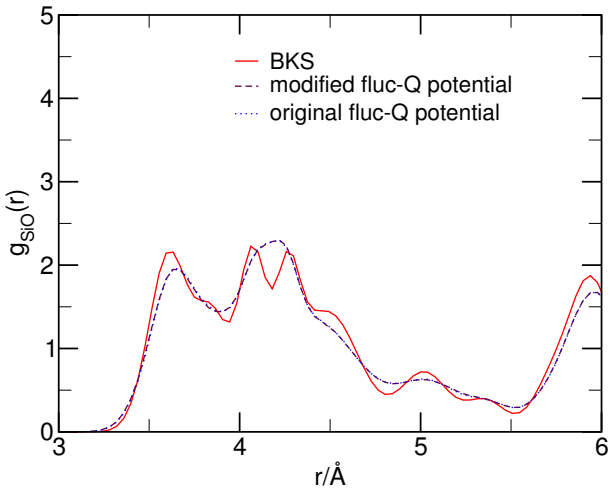


Figure 1.1.: Pair correlation function $g_{\mathrm{SiO}}(R)$ of α-quartz at 300 K. The curves for the new consistent fluctuating charge potential and the original unconsistent potential basically lie on top of each other. The nearest neighbor peak, which is located at 1.6 $\mathring{A}$, is not included in the graph.

1.3.6. Kinetic Parameters for Extended Lagrangian

In the extended Lagrangian approach, the equilibration of the effective charges reflects the movement of the electrons in the system. To adiabatically decouple the electronic movement from the ionic movement, it is therefore necessary to separate the relevant time scales. Therefore in addition to the usual relation $\Delta t \ll \tau_{\mathrm{ion}} \ll T_{\mathrm{Sim}}$, the new electronic timescale has to fit in: $\Delta t \ll \tau_{\mathrm{QEq}} \ll \tau_{\mathrm{ion}} \ll T_{\mathrm{Sim}}$, so that the charge equilibration time is well below the equilibration time of the ionic movement. This was achieved by choosing a fictious mass for the charges of 0.01 u. The damping constant in the Langevin thermostat was set by finding the value where the charge equilibration is critically damped, which is 2.0. With these numbers the equilibration time of the charges is about 10 fs, which means a frequency of 100 THz, well beyond the typical phonon frequencies in SiO_2. To cover these higher frequencies in the simulation, the time step was shortened by a factor of 10 to $\Delta t_{\mathrm{MD}} = 0.1$ fs in this type of simulation. As the authors of the fluctuating charge potential mention that they used the direct approach to calculate the charges every 25 to 100 timesteps, it is interesting to note that this corresponds to frequencies of 10 to 40 THz. This is within the phononic frequency range and leads to the situation that the charges cannot adapt to the instantaneous ionic configuration.

1.4. Fluctuating Dipole Model

In the fluctuating charge procedure, charge can only fluctuate from one atom to the other. This can bring about polarizations within the crystal, but it does not account for polarizations within the electron hull of an atom. The fluctuating dipole procedure in contrast directly assigns a dipole moment on each atom site. In case of the quartz system to be considered, only the oxygen atoms are assigned a dipole moment. This is because the silicon atoms have a rather stable nearest neighbor environment that consists of symmetrically arranged oxygen atoms in the corners of a tetrahedron. Therefore polarization on these atoms is unlikely to play a major role. The polarization of the oxygen atoms however is well established[60].
The force field for SiO_2 used in the following was parametrized by P. Tangney and S. Scandolo[90]. For the parameter fitting they used the *optimal potential routine* developed by Ercolessi[25] and in this form by Laio et al[57]. It is an iterative procedure that obeys the following algorithm: Beginning with the best potential available, the system is equilibrated with an MD simulation. Once equilibrated, a number of atomic configurations are generated. On these configurations, density functional theory calculations of total energy, forces and stress are performed and these are used to perform the parametrization. With the new parameters, the scheme will be repeated until the current parameter set fits the current and the previous ab initio data to the same degree. Finally the emerging parameter set is supposed to produce potential energy, forces and stress in agreement with the DFT method used for parametrization, but much faster as the ab initio method. This statement obviously only holds in terms of a certain accuracy and in the vicinity of the surrounding conditions that were employed in the parametrization procedure, like temperature and external pressure. When this temperature and pressure is exceeded significantly, the accuracy of the potential cannot be taken for granted but has to be questioned as it is the case with every model potential. One major advantage of this method is that it does not rely on any experimental input and thus the parametrization can take place in principle in any possible thermodynamic conditions, even where experimental data is unavailable.
In this work, we used the Fortran subroutine provided by Paul Tangney that calculates forces, stress tensor and potential energy for a given configuration. The parametrization was done by Tangney and Scandolo in the liquid phase at 3000 K, in order to find a parameter set which is not biased towards a particular crystal structure. In the following the algorithm used to solve for the dipoles will be outlined.
Calculation of the induced polarization is a many-body problem, where the induced dipoles all depend on each other. If α_i is the polarizability of site i, the induced dipole

$\underline{p}_i$ at position $\underline{R}_i$ is given by

$$\underline{p}_i = \alpha_i \underline{E}_i \tag{1.35}$$

$$\underline{E}_i = \underline{E}_i^0 + \sum_{j \neq i} \underline{\underline{T}}_{ij} \underline{p}_j \tag{1.36}$$

The polarization is proportional to the electric field, which in turn depends on all the dipoles via the dipole-dipole interaction tensor $\underline{\underline{T}}_{ij}$. Here $\underline{E}_i^0$ is the "fixed" part of the electric field due to charges and a possibly applied external field. To solve this pair of equations amounts to solving a set of $3N \times 3N$ linear equations. Thus the possibility to solve them directly or in an iterative way can be ruled out directly, as it would be far too computationally expensive. Therefore these equations have to be solved either with an extended Lagrangian scheme as done in [88, 106] or with a self consistent approach as in [94].
However, Wilson and Madden[106] found that this model for polarization had significant shortcomings. The ions tended to become over-polarized, and small anion-cation distances resulted in very large induction forces which overcame the short-range Pauli exclusion repulsion. This is because the model neglects the dipoles induced by short-range overlap effects, which should oppose the Coulomb-field induced dipoles. On the basis of electronic structure calculations[27], Wilson and Madden suggested an additional term to incorporate these dipoles into the present model. The induced dipole from eq. (1.35) will now be:

$$\underline{p}_i = \alpha_i \underline{E}_i + \sum_{j \neq i} \alpha_i g_{ij}(R_{ij}) \frac{q_j \underline{R}_{ij}}{R_{ij}^3} \tag{1.37}$$

where

$$g_{ij}(R_{ij}) = -\exp(-bR_{ij}) \sum_{k=0}^{4} \frac{(bR_{ij})^k}{k!}. \tag{1.38}$$

Putting everything together, the algorithm basically works according to the following principle:

1. Calculate the contribution from the charges to the energy, forces and stress
2. Calculate the short-range induced dipole moment on each ion
3. Formulate a guess for the electric field on each ion by extrapolation from the last three timesteps
4. Solve for the dipoles, until they are consistent with the resulting electric field up to a certain accuracy

5. Use the converged values to calculate the contribution from charge-dipole and dipole-dipole interaction to energy, forces and stress
6. Calculate the contribution of the short-range induction of dipole moments to the energy, forces and stress
7. Add up all contributions to the energy, forces and stress

The pair interaction between the ions used in the force field by Tangney and Scandolo is a point charge Coulomb interaction plus a pair potential of Morse-Stretch form to model the short-range interactions:

$$U_{ij} = \frac{q_i q_j}{R_{ij}} + D_{ij}[e^{\gamma_{ij}(1-r_{ij}/r_{ij}^0)} - 2e^{\gamma_{ij}/2(1-r_{ij}/r_{ij}^0)}]. \tag{1.39}$$

The parameter set used is shown in Table 1.5.

	D	γ	r_0
O-O	2.4748×10^{-4}	12.07092	7.17005
Si-O	1.9033×10^{-3}	11.1523	4.6371
Si-Si	-2.0846×10^{-3}	10.45517	5.75038

q_O	q_{Si}	α	b	c
-1.38257	2.76514	8.89378	2.02989	-1.50435

Table 1.5.: Fluc. Dipole Force Field parameters (atomic units)

1.5. Comparison of Performance

As far as the speed of calculations is concerned, the simplest potential, which is the two-body BKS potential, is obviously the fastest. The fluctuating-charge potential is slowed down in comparison to BKS by about a factor of 10 due to the shorter time step that is needed with the extended Lagrangian approach to equilibrate the charges and to decouple charge motion from the ionic motion. Using the direct calculation of the charges at every 25th to 100th time step as suggested by the original authors [20] leads to a similar slowing down.
The comparison of the performance of the fluctuating-dipole potential with the standard two-body potential is difficult as our BKS routine is highly optimized while the fluctuating-dipole routine uses new routines for every part of the calculation. As these

were not optimized at all, the comparison would be meaningless because of different ways to calculate the details as for instance the Ewald sum, especially its reciprocal part, or the neighbor lists. The original authors [90] claim to see a slowing down by a factor of 100. However in comparison to our BKS routine the slowing down is even more significant, which is why in many instances in the following work the system sizes for the fluctuating-dipole simulations will be smaller than for the other potentials. In general the fluctuating dipole can be treated comparable to the fluctuating charges, and thus in an optimized routine the slowing down should have a similar value, which means a factor of 10.

2. The α-β Quartz Transition

The transition from α- to β-quartz is a displacive phase transition, which occurs at a transition temperature of $T_{\text{tr}} = 846$ K. The atomic structure of the α and β polymorphs of quartz is known since 1925 [9] and was refined in the following decades by numerous experimental studies [39]. The structure of α-quartz is most easily envisioned as

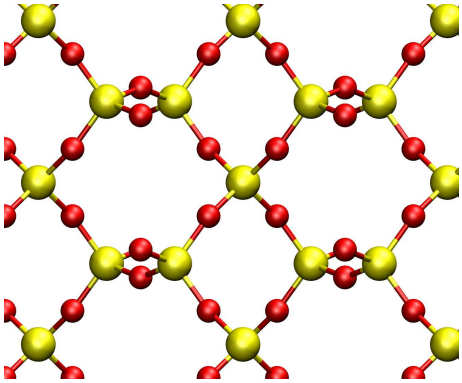


Figure 2.1.: Sketch of the average β-Quartz structure along the [100] direction

a distortion of the high-temperature β modification. When β-quartz is cooled below the transition temperature, the expanded β-quartz framework collapses to the denser α-quartz configuration. For the average structure, the contraction of the tetrahedral

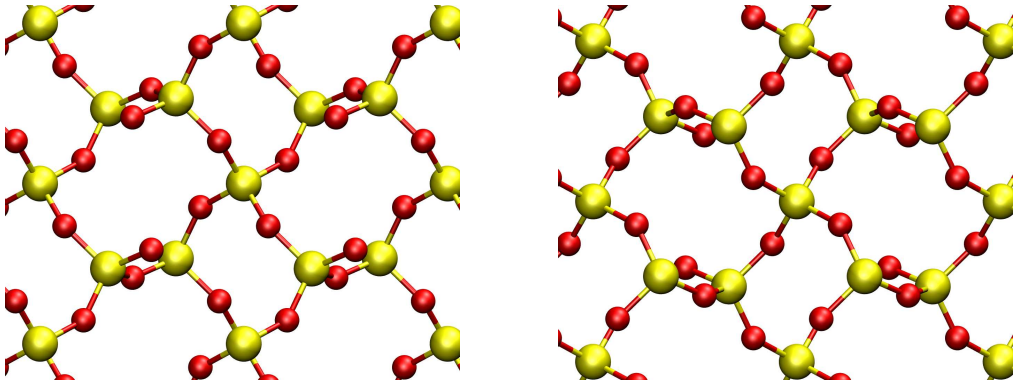


Figure 2.2.: The two orientations of α-Quartz

network may be described geometrically as the rotation of rigid tetrahedra around the [100] axes by an angle Θ. This angle can be used as on order parameter, which is zero in the β polymorph, whereas Θ is 16.3 for α-quartz at room temperature. Depending on the sign of the tilting angle, the transformation yields two distinct α_1 and α_2 twin orientations as shown in Fig. 2.2. These are called the Dauphiné twins of α-quartz.
However the real picture of the transition is far more complicated. Even though the $\alpha - \beta$ transition has been subject of many studies over the past 100 years [39], it is surprising that there is still no firm picture of what actually happens at the phase transition. This uncertainty is connected with the discussion about the real structure of β-quartz. While figure 2.1 shows the average atomic positions for the oxygen atoms, it turns out that these average position are rarely occupied. It is not completely resolved what the instantaneous structure looks like that leads to the symmetric average positions. There are basically two competing ideas of β-quartz.
One picture of the β-quartz structure is based on the idea that each oxygen atom is associated with not one but two potential minima. As a consequence the atom would oscillate between two positions that correspond to the Dauphiné twin configurations of α-quartz. This means that in the high symmetry phase the atoms are hopping between the positions corresponding to the different domains of the low symmetry phase, giving the structure shown in Fig. 2.1 on average. Upon cooling, the dynamic disorder in the high symmetry phase gets ordered when the atoms progressively occupy the positions corresponding to one of the orientations in the low symmetry phase. Therefore this picture of β-quartz would imply an order-disorder type transition. It has been supported by results of Neutron diffraction[107], NMR studies[87] and MD simulations[100].
The other picture supports an ordered structure of β-quartz. The idea is that the atoms in principle vibrate around mean positions given by the average structure, which change on cooling below T_{tr} through symmetry breaking displacements. This point of view explains the observed existence of soft modes in α- and β-quartz[3, 92, 13, 24] and the absence of symmetry forbidden phonons in the β phase of quartz[81], which should be detected if β-quartz consisted of clusters of α-quartz that persist over longer time scales. However it is clear that the oxygen atoms do not fluctuate harmonically around their average positions. This is supported by an X-ray study[49] that showed that the oxygen probability density functions deviate considerably from Gaussians. This outcome was not interpreted as disorder in the sense of clusters of microtwins, but rather as librational motion of the oxygen atoms around their averaged positions, which lie on the straight connection of two Si atoms.

2.1. Transition Temperature

In the following the transition temperature T_{tr} of the α-β transition will be determined for the different potentials. For the BKS potential this value was recently shown [66] to be 740 K with an uncertainty of about 5 K. For the fluctuating charge and the fluctuating dipole potential the transition temperature is not known and will be determined in this section.
In general for the calculation of the transition temperature in a computer simulation one is confronted with the problem, that due to the finite size of the simulation box all thermodynamic properties behave smoothly near the phase transition, and so does the order parameter. In order to determine the transition temperature T_{tr} nevertheless accurately, it is possible to use the fourth-order cumulant [8], which is defined in the case of a one-component order parameter as

$$g_4(N,T) = \frac{1}{2}\left(3 - \frac{\langle(\phi)^4\rangle_N}{\langle(\phi)^2\rangle_N^2}\right), \tag{2.1}$$

where $\langle\phi^k\rangle$ deontes the thermal average of the k'th moment of the order parameter for an N-particle system. It has been shown [104], that $g_4(N,T)$, aside from small correction terms, has a size-independent crossing point at a first-order phase transi-

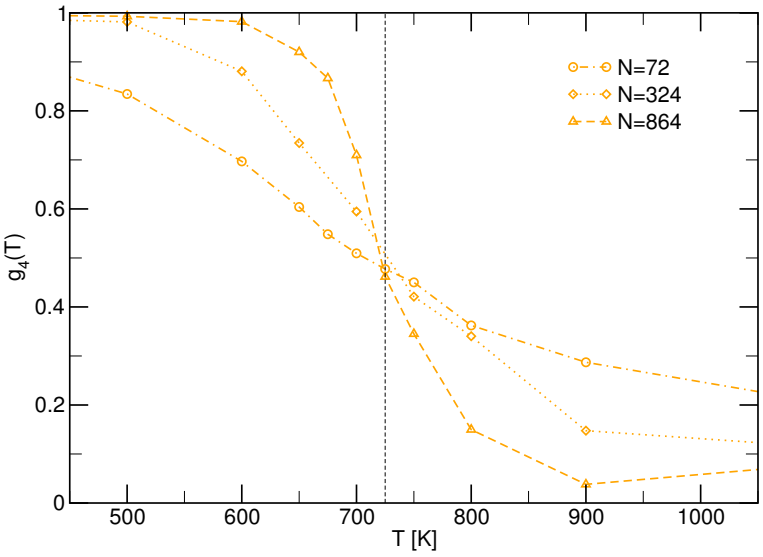


Figure 2.3.: Fourth-order cumulant g_4 as a function of T for the fluctuating dipole potential

tion. This is valid under the restriction that the geometry of the simulation cell remains similar when the particle number is increased. This condition can only be approximately satisfied in the case of a crystal, as it can only be expanded in multiples of the

unit cell. The unit cell used in the simulations is even larger than the primitive cell, as we fit the trigonal α-quartz phase into a rectangular simulation box. So only an approximately coinciding crossing point can be expected for the different system sizes. The system sizes chosen are $N = 72$, $N = 324$, and $N = 864$. As shown in Fig. 2.3, there is not a perfect crossing of the cumulants for the different system sizes, but it still allows one to locate the transition temperature at $T \approx 715$K with an uncertainty of about 5K. This value is comparable to the BKS transition temperature, but even lower and 120K too low compared to the experimental transition temperature.
To gain a more accurate picture of the transition it can be described within Landau

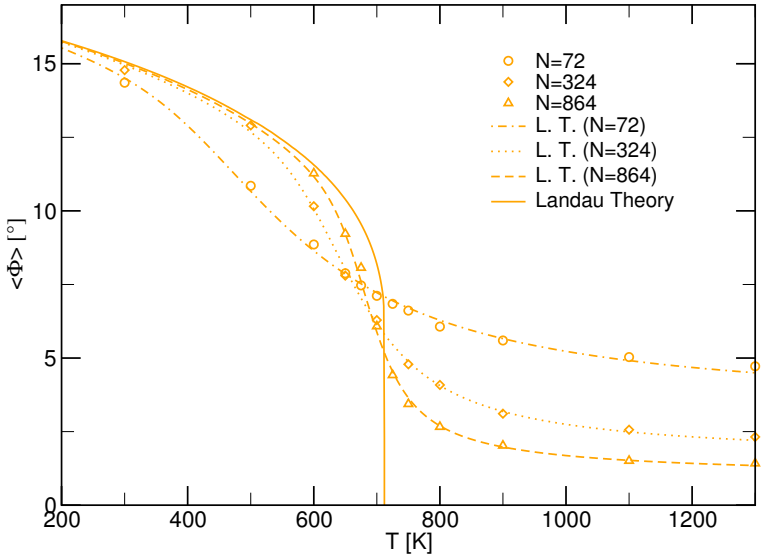


Figure 2.4.: Order parameter $\langle|\Phi|\rangle$ as a function of temperature. Shown are different system sizes simulated with the fluctuating dipole potential, fits obtained with Landau theory and finite-size Landau theory.

theory in the same way as it is done in Ref. [66]. It is well established that the macroscopic evolution of quartz at the transition point can be described by the standard Landau expansion for a first-order transition [13, 35, 4, 5]. Odd order terms in Φ are not permitted by symmetry and the expansion is written as

$$F(\Phi, T) = \frac{1}{2}a(T - T_c)\Phi^2 + \frac{1}{4}b\Phi^4 + \frac{1}{6}c\Phi^6. \tag{2.2}$$

F is the free energy per particle as a function of temperature T and order parameter Φ and a, b, c, and T_c are parameters. The parameter b should be negative to have a (weak) first order transition.
These parameters can be specified by fitting the absolute value of the order parameter $\langle|\Phi|\rangle$ as a function of temperature to the data obtained from simulation. In finite-size

Landau theory for a system with N particles the expectation value reads

$$\langle|\Phi|\rangle = \frac{\int_{-\infty}^{\infty} d\Phi |\Phi| \exp\{-\beta N F(\Phi, T)\}}{\int_{-\infty}^{\infty} d\Phi \exp\{-\beta N F(\Phi, T)\}}. \tag{2.3}$$

This function is fitted against the data from the $N = 324$ system and shown in Fig. 2.4. For the two other system sizes also the fit parameters obtained from $N = 324$ are shown. It can be seen the behaviour of $\langle|\Phi|\rangle$ is reasonably described by Landau theory. The fit parameters are $a = 7.41 \times 10^{-3}\,\mathrm{J/(mol \cdot K)}$, $b = -4.97 \times 10^{-3}\,\mathrm{J/mol}$, $c = 8.03 \times 10^{-5}\,\mathrm{J/mol}$, and $T_c = 704\,\mathrm{K}$. From Fig. 2.4 the transition temperature can be read as $T_{tr} = 712\,\mathrm{K}$. Similar to the experiment [13], b is found to be negative, which characterizes a first-order transition, and $T_{tr} - T_c = 8\,\mathrm{K}$. This is also in good agreement with different studies based on experimental data, which typically determined the value of $T_{tr} - T_c$ in the range of $4 - 13\,\mathrm{K}$ [3, 45, 35, 4, 22, 50]. In total the features of experimental studies are certainly reproduced, even though the transition temperature is about $130\,\mathrm{K}$ smaller.
The same analysis for the fluctuating-charge potential (Fig. 2.5) is done for systems

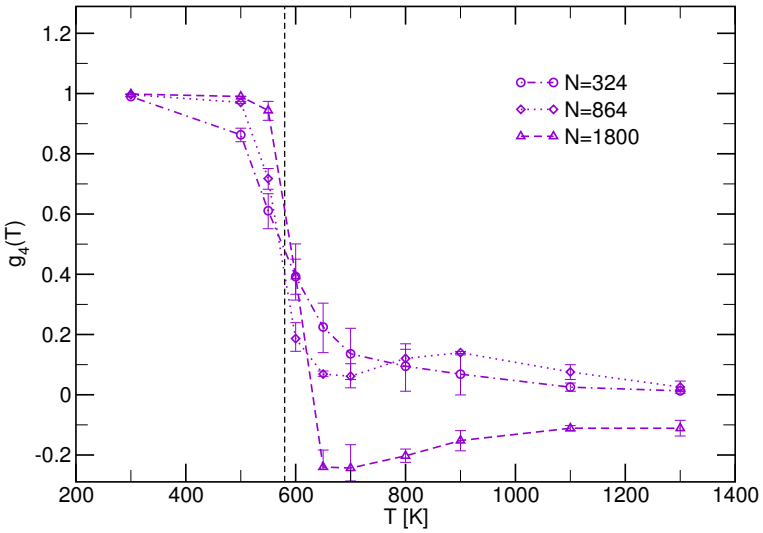


Figure 2.5.: Fourth-order cumulant g_4 as a function of T for the fluctuating charge potential.

sizes $N = 324$, $N = 864$, and $N = 1800$. The system sizes are chosen somewhat larger than before for the reasons mentioned in chapter 1.5. The non-monotonic behaviour of the cumulant, which was also observed in Ref. [104], does occur within the plotted error bars. The cumulant analysis yields a transition temperature of about $T = 550\mathrm{K}$. The parameters of the Landau expansion (eq. (2.2)) are fitted as described before to the data of the $N = 864$ system. The parameters obtained were used for all three system sizes shown in Fig. 2.6. The fit parameters are $a = 8.92 \times 10^{-3}\,\mathrm{J/(mol \cdot K)}$,

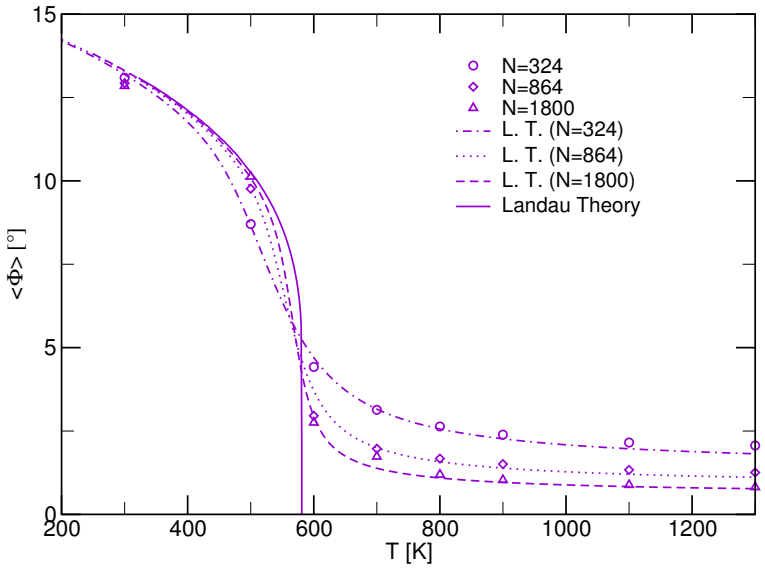


Figure 2.6.: Order parameter $\langle|\Phi|\rangle$ as a function of temperature. Shown are different system sizes simulated with the fluctuating charge potential, fits obtained with Landau theory and finite-size Landau theory.

$b = -4.30 \times 10^{-3}\,\mathrm{J/mol}$, $c = 1.02 \times 10^{-4}\,\mathrm{J/mol}$, and $T_c = 570\,\mathrm{K}$. Again, b is negative and the difference $T_{tr} - T_c$ is positive and about $10\,\mathrm{K}$, as we can find a transition temperature of $T_{tr} = 581\,\mathrm{K}$ in Fig. 2.6. Again the characterisation of the phase transition seen in experiment is obtained quite similar, but the discrepancy in the transition temperature of more than $250\,\mathrm{K}$ is tremendous.

2.2. Hysteresis

An interesting feature in the α-β transition was discovered in Ref. [66] that is linked to hysteresis effects in the transition. It was shown that one can distinguish between the two phases from the local structure, which offers the possibility to study hysteresis effects directly. This statement holds for our simulations, as it is shown for the case of the fluctuating-dipole potential in Fig. 2.7. The pair correlation function $g(r)$ does not change significantly with temperature above and below T_{tr}, but makes a sudden change at the transition. So one can observe the hysteresis effect in the $g(r)$ function. While the structure at $T = 725\,\mathrm{K}$ that was started with an α-quartz configuration equilibrated $25\,\mathrm{K}$ below, it remains stable in this phase. On the other hand it also remains stable in the β-quartz phase, when the initial configuration was equilibrated $25\,\mathrm{K}$ above. Of course in a simulation close to the transition temperature the system undergoes more changes between the two phase for a small box, only for a larger particle number the

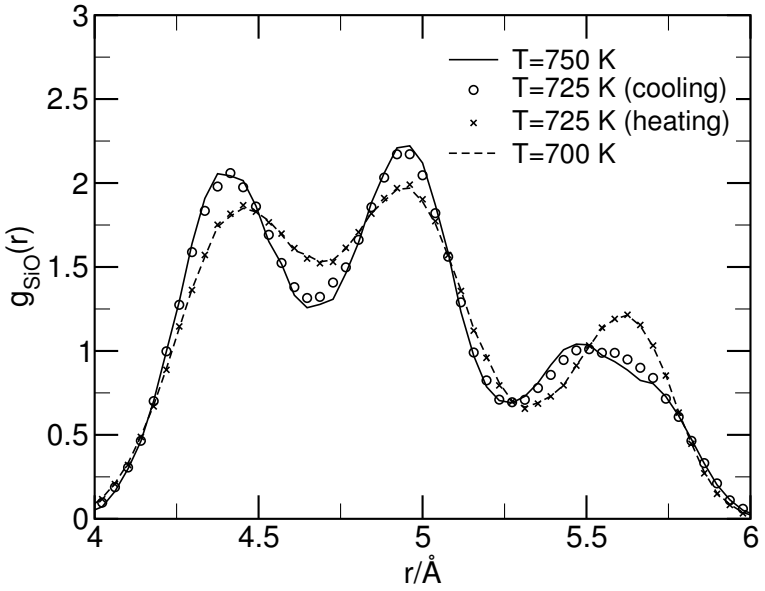


Figure 2.7.: Radial distribution function $g_{Si-O}(r)$ at various temperatures for system size N=864 with fluctuating-charge potential.

system stays in one phase.

2.3. Thermal Expansion and c/a Ratio

One check for the different models is the temperature dependence of the volume close to the transition. As can be seen in fig. 2.8 the BKS potential gives fairly good agreement with experiment, even though the volume is sligthly overestimated and the transition temperature differs from the experimental transition temperature as described in the previous section. The result of the fluc-Q potential is quite similar, even though the volume of both quartz phases is even bigger for this model. The best quantitative agreement with experiment can be seen for the fluc-μ potential. The rather smooth behaviour at the transition is due to the fact that this potential allows for the existence of both of the two phases in a quite broad temperature range around the transition.
However the volume expansion is not isotropic, but the jump at the transition was observed in experiment [13] to be of different magnitude for different spatial directions. This can be seen in the ratio of the lattice constants c and a, which shows a discontinuity at the transition. It was recently pointed out [66], that in a computer simulation using the BKS potential the behaviour of the c/a ratio differs qualitatively from the experimental result, as the simulation does not see any discontinuity or even any distinct feature in the c/a ratio at the transition. This behaviour is shown in figure 2.9. Shown

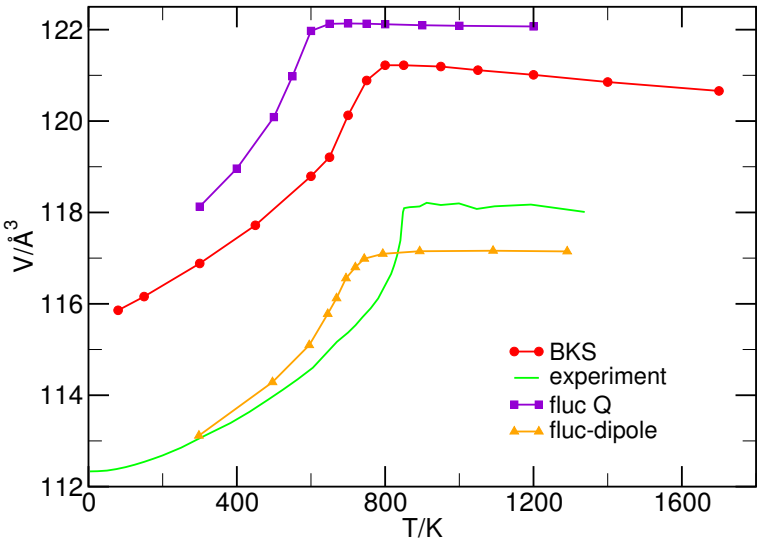


Figure 2.8.: Temperature dependence of the volume per unit cell at the α-β transition. All of the three different model potentials show qualitative agreement with experiment, while the quantitative agreement is best for the fluc-μ potential.

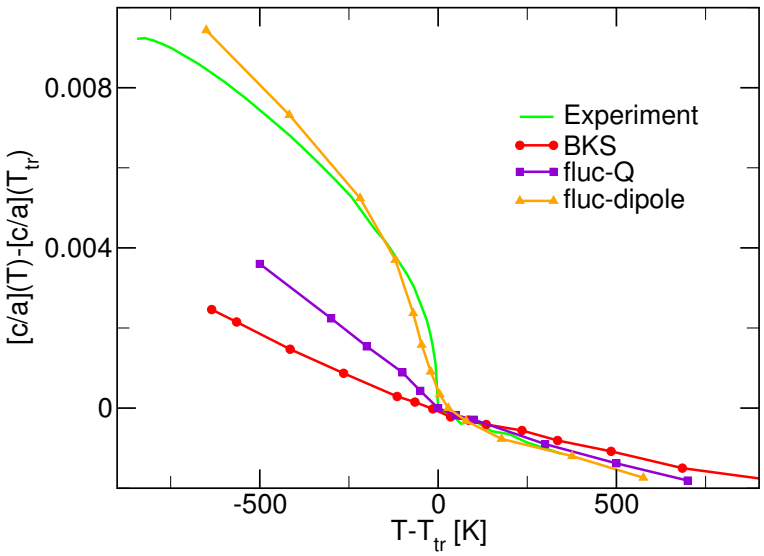


Figure 2.9.: Temperature dependence of the c/a ratio at the α-β transition. For easier comparison the curves are shifted to match at their respective transition temperature.

in this figure are also the curves for the fluc-Q and the fluc-μ potential. In the case of the fluc-Q potential one can see a change in the slope at the transition temperature. Thus there is at least an effect of the transition in the c/a ratio, but it is still far away from reproducing the height of the stroke that occurs in experiment. In contrast the fluc-μ approach matches the experimental course of the c/a ratio quite well even in

quantitative terms. The smooth behaviour at the transition is again owing to the broad range of phase coexistence as discussed above.

2.3.1. Influence of Specific Properties of the Potentials

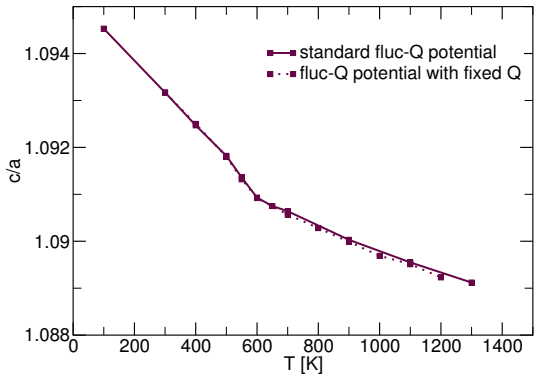


Figure 2.10.: Temperature dependence of the c/a ratio at the α-β transition for the fluc-Q potential with fluctuating and fixed charges.

A closer look at the origin of these differences can be done by investigating the influence of the fluctuating charges and the fluctuating dipoles on this behaviour. If we use the fluctuating charge potential with the charges fixed on the average charges in α-quartz, the behaviour of the c/a ratio is only marginally different, and only in the high temperature regime. The kink at T_{tr} is still of the same shape as with the fluctuating charges. In general there is not much difference between the average charges in α-quartz, which are about $< Q_{\text{Si}} >= 1.318e$ for Si, and in β-quartz, where the value is $< Q_{\text{Si}} >= 1.293e$. So it is justified to state that the fluctuations of the charges barely affect the simulation, at least the kink in the c/a ratio is certainly not a result of *fluctuating* charges but rather an effect of the pure two body forces in the potential.
The same analysis for the fluctuating dipole potential results in a completely different picture. The simulation crucically depends on the fluctuating dipoles. When the dipoles are fixed on the average values found in α-quartz, the crystal would not transform to the β phase even at temperatures as high as $1300K$. Similarly, with fixed average β quartz dipoles the crystal would stay in a β-quartz configuration even at $300K$. This can be seen in the temperature dependence of the order parameter $\langle|\Phi|\rangle$ as shown in fig. 2.11. While in the first case the volume (Fig. 2.12) matches the volume of the run that has the dipoles fluctuating at a temperature of $300K$, where the averaging was performed, the volume in the latter case is much higher than with a fluctuating dipole run. This is due to the fact that the average dipoles are not a reasonable quantity in the β-quartz phase, as they represent the dipoles that result from thermal averages

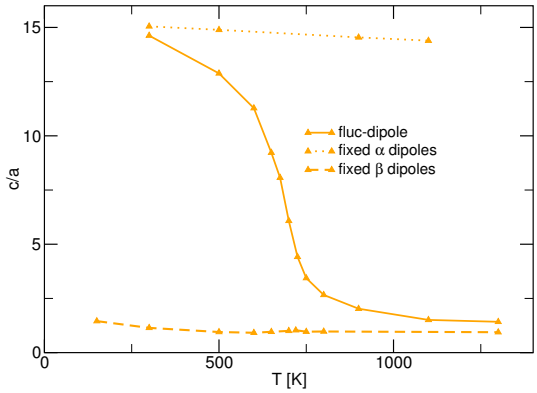


Figure 2.11.: Order parameter $\langle|\Phi|\rangle$ as a function of temperature for the unaltered fluc-μ potential and for the same potential with dipoles fixed on their average values in α-quartz or β-quartz

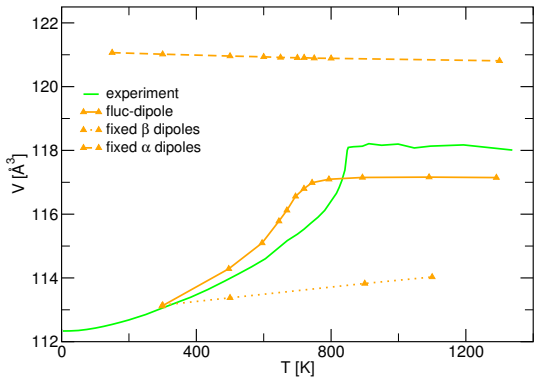


Figure 2.12.: Volume per unit cell of quartz as a function of temperature for the unaltered fluc-μ potential and for the same potential with dipoles fixed on their average values in α-quartz or β-quartz

over the structure. So for example, for symmetry reasons, the z component of the dipole must be zero in β-quartz but not in α-quartz. To constrain the dipoles to the values that they have in β-quartz makes the crystal tend more to the average structure with stretched bonds on the oxygen atoms, which explains the artificially increased volume that is observed when the constraint is applied in the simulations. Also the c/a ratio, shown in fig. 2.13, is quite independent of temperature in this setting and shows an unphysical value for the fixed β-quartz dipoles. To summmarize, one can say that the dipoles play an important part in the model and are responsible for a big part of the forces that act on the atoms. While these simulations show the effect that the dipoles have on the structure and the phase transformation, the mechanism how the c/a anomaly at the α-β transition comes about is not yet resolved.

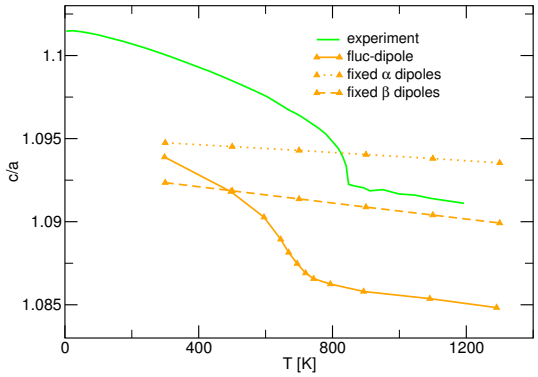


Figure 2.13.: c/a ratio of quartz as a function of temperature for the unaltered fluc-μ potential and for the same potential with dipoles fixed on their average values in α-quartz or β-quartz

2.4. Distortion of Tetrahedra

The reason for the behaviour of the c/a ratio in quartz was investigated by Grimm and Dorner [35]. They isolated three tendencies that go along with increasing temperature.

- increasing Si-O-Si angle
- decreasing mean Si-O distance
- contraction of SiO_4 groups in c direction

While it is possible to describe the change in the volume of quartz by rotation of rigid tetrahedra by leaving all nearest neighbor Si-O bond lengths constant, the behaviour of the c/a ratio cannot be reproduced by such a model. It was shown [86] by purely geometrical arguments that a value of $c/a > 1.0981$ can only be achieved in a quartz crystal if the tetrahedra are deformed. The model employing only tilting of rigid tetrahedral SiO_4 units only can not account for a lower value in this ratio, even though it can explain all volume effects. Given this argument and the experimentally known lattice constants a deformation of the SiO_4 tetrahedra must be present for temperatures lower than about 400 K. The shearing of the tetrahedra in the quartz phase is also well established experimentally [35, 49, 13]. To test if such a deformation is present in the simulations, one can look at the tetraeder angle $\vartheta_{\mathrm{OSiO}}$, which is shown in fig. 2.14. However the fluctuating dipole potential, which definitely shows the same c/a ratio behaviour as in experiment, does not exhibit a dramatic change in the overall average of the OSiO angle when going to small temperatures (see fig. 2.14) and no obvious difference compared to the BKS potential. On the other hand the average of all tetrahedron angles is not sensitive to shearing, as the different angles in a deformed tetrahedron are likely to cancel out on average. The second moment of the distribution

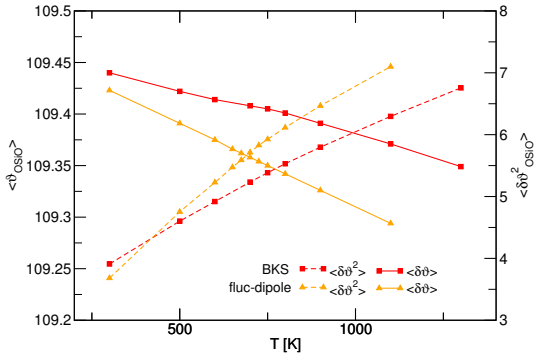


Figure 2.14.: Average O-Si-O bond angle and its second moment as a function of temperature

could give more information, but the value obtained with fluc-μ is comparable to that in BKS.
A more sensitive analysis of the angle distribution can be done with the help of the

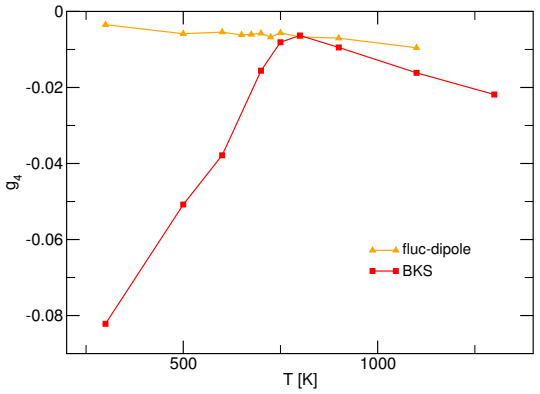


Figure 2.15.: Average fourth order cumulant of the O-Si-O bond angle as a function of temperature

fourth-order cumulant shown in fig. 2.15. As different mean values for the different O-Si-O bond angles would result in a superposition of Gauss distributions with different mean, the overall average should deviate from a plain gaussian. A measurement for the gaussian behaviour of a distribution is the fourth-order cumulant, defined as

$$g_4 = \frac{1}{2}\left(3 - \frac{\langle(\delta\vartheta)^4\rangle}{\langle(\delta\vartheta)^2\rangle^2}\right), \tag{2.4}$$

which is zero for a gaussian distribution. One can see that the angle distribution generated by the BKS potential deviates even more from gaussian behaviour than the fluc-dipole bond-angle distribution. This means that the tetrahedra in the α phase of the BKS potential are even more deformed than in the fluc-dipole potential. This can also be seen quite obvious by an exemplary look at the average angles making up the

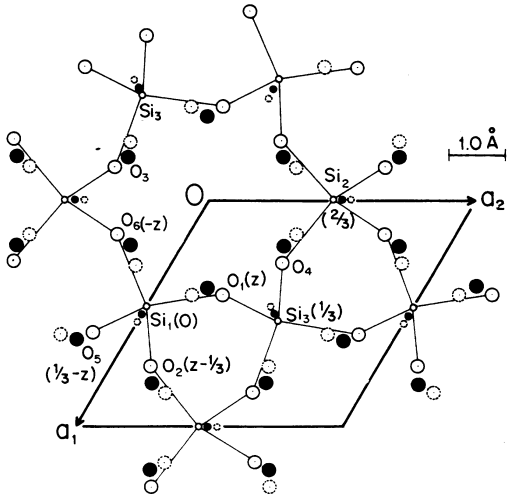


Figure 2.16.: [001] projection of the α_1, α_2 and β configuration. Heights along [001] are shown in parantheses. z is $1/6$ for β-quartz and slightly below or above for the two Dauphiné twins of α-quartz. Picture taken from Kihara.

tetrahedron around one specific silicon atom. In order to enumerate the atoms in a well-defined way, the definition by Kihara [49] is used, which is shown in fig. 2.16. The comparison of the angles around the Si_1 position for α-quartz at $298K$ is shown in the following table with experimental values by Kihara.

	Experiment	BKS	fluc-dipole
O_1-Si-O_2	110.5	107.5	110.1
O_1-Si-O_6	108.9	114.8	108.9
O_1-Si-O_5	108.8	108.2	109.1
O_2-Si-O_5	109.3	110.5	109.5

The table shows that the deviation between the ideal tetrahedral angle of 109.47 degrees and the calculated angles has the correct sign and amplitude for the fluctuating-dipole potential as compared with experiment, in contrast to the BKS result. One can see that the large deformation, that was also seen in the figures above, is actually way to large compared to experiment and moreover has the wrong sign. In three out of the four angles shown, the abberation from the tetraeder angle that is predicted by BKS goes into the wrong direction. The explanation follows from the geometrical argument given by Smith [86] when we look at the absolute value of the c/a ratio produced by BKS. It is shown in fig. 2.17, that the c/a ratio does not exceed the value of 1.0981 for the case of the fluctuating dipole potential, in contrast to the BKS potential, which is below this value only for high temperature in the β-quartz. So the BKS quartz has to be strongly deformed even in the β phase because of geometrical reasons, so it is clear

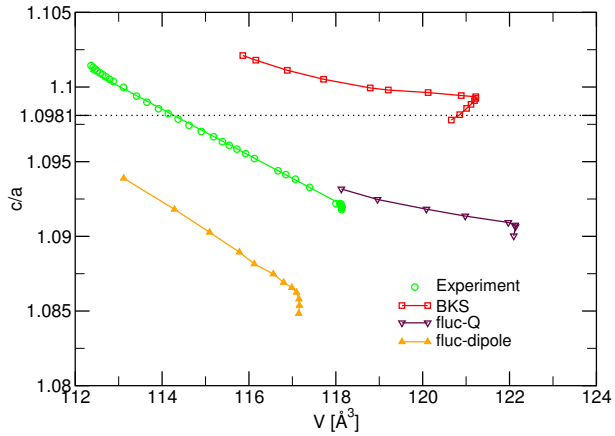
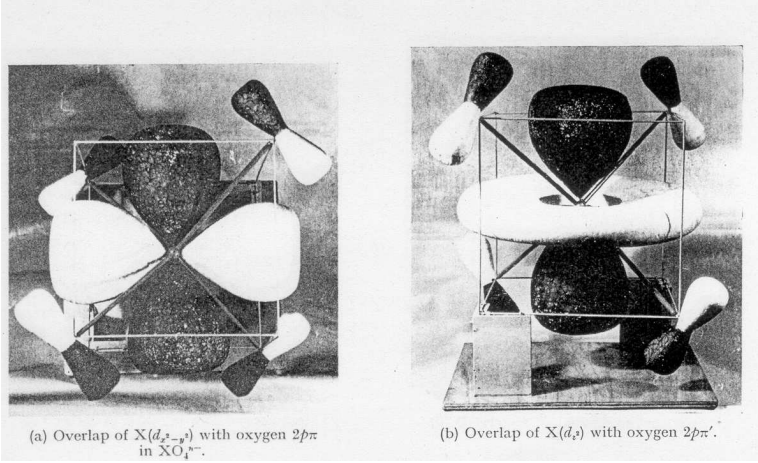


Figure 2.17.: c/a ratio in quartz as a function of the volume per unit cell.

that the subtle shear effect in the tetrahedra that would lead to the correct behaviour of the lattice constants is completely suppressed by a large deformation having the incorrect sign.

2.4.1. Influence of Atomic Bond Character

The origin of the distortion of the tetrahedra that leads to a contraction of the z axis in quartz, was related by Hill and Gibbs [44] to an increasing π bond order, i.e. an increasing fraction of π-bonded valence electrons. At ambient conditions, Hill and Gibbs have observed that the mean Si-O bond length is inversely proportional to the size of the Si-O-Si angle. Plots of R_{SiO} versus the function $-1/\cos(\text{Si-O-Si})$ for a wide variety of silicate structures are linear with the shorter bonds associated with the wider angles. On the basis of molecular orbital calculations [59, 34] a linear relation between π-bond order and both, the mean Si-O distance and $-1/\cos(\text{Si-O-Si})$, was proposed. This was based on the π-bonding model of Cruickshank [18], which is illustrated in Fig. 2.18. The two pictures show how the p-orbitals of the oxygen atoms can be combined with the d_{z^2} and $d_{x^2-y^2}$ orbitals of the Si atom to form π-bonding molecular orbitals (the colours correspond to the signs of the wave function). It was shown for SiO_4 type tetrahedra that the decreasing Si-O distance is accompanied by an increasing π-bond order. [18]. Furthermore the d-orbitals single out the z' axis and at least for the $d_{x^2-y^2}$ orbital it is obvious that an increasing π-bond order will tend to pull the oxygen atoms towards the $x'y'$ plane.
To demonstrate the validity of this model for quartz, Grimm and Dorner showed the correlation between the mean Si-O distance and the Si-O-Si angle as mentioned above. To compare their conclusions to the simulations one has to determine the Si-O bond



(a) Overlap of X($d_{x^2-y^2}$) with oxygen $2p\pi$ in XO_4^{n-}.

(b) Overlap of X(d_{z^2}) with oxygen $2p\pi'$.

Figure 2.18.: Models showing the formation of binding π-orbitals for XO_4 groups. Pictures taken from Cruickshank.

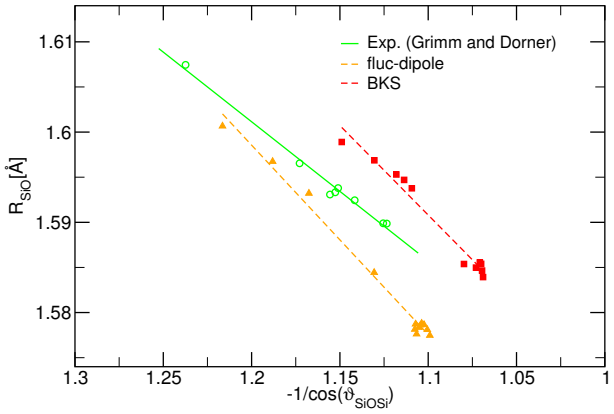


Figure 2.19.: Correlation between the mean Si-O distance R_{SiO} and the Si-O-Si angle θ_{SiOSi}

length in the average configuration rather than the average of the instantaneous bond lengths. The outcome in Fig. 2.19 shows clearly, that the angular dependence of the Si-O bond length is very similar to the experiment. The absolute numbers are comparable and the dependence is linear as predicted by the Hill-Gibbs relation. The figure shows that the effects of the π bonds on the Si-O distance and the Si-O-Si angle, which controls most of the volume expansion, are obviously included in the BKS potential fairly well as well. Only the third effect, the contraction of the tetrahedra in z direction that leads to the anomaly in the c/a ratio, is obviously not correctly modelled by the BKS potential as shown before, in contrast to the fluctuating dipole potential.
Keeping in mind the Hill and Gibbs discussion of π-bond order and the results presented above raises the question whether the effect of π-bonds has been parametrized implicitly as (fluctuating) dipoles in the potential by Tangney and Scandolo. It is shown

in Fig. 2.20, that the symmetries of the average dipoles in this potential very much re-

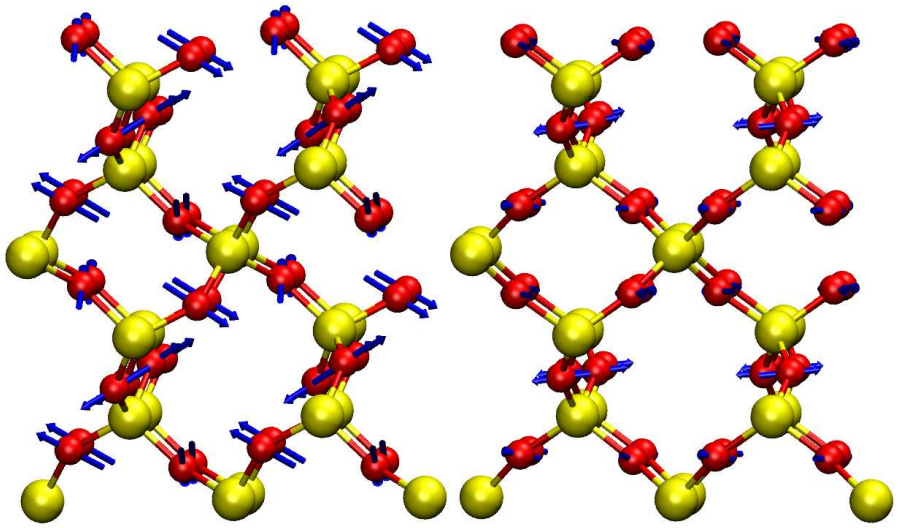


Figure 2.20.: Average dipoles in the α- (left) and the β-quartz phase (right). Projection along the [100] direction. Obvious is the symmetry forbidden z component (vertical) in β quartz as well as the dipole orientation orthogonal to the Si-O-Si connection line in α-quartz.

semble to the symmetries in the π bond model. Their orientation around a tetrahedron is the same as for the oxygen π orbitals and they single out the z direction. Of course this only constitutes an analogy as the quantum chemical interaction of the orbitals in Fig. 2.18 and the classical dipole interaction are of different nature. But it underlines the suspicion that the dipoles introduce the right effects into the system, but maybe do so by incorporating physics which is usually not to be described by classical dipoles alone. But it is not too surprising that this actually works, as already a simple classical two body potential like BKS can model the tetrahedra quite well, even though their formation is a quantum chemical effect of the sp^3 hybrid orbitals.

In a later chapter on electromechanical coefficients, it will be shown that the piezoelectrical response of quartz using the fluctuating charge potential reproduces available experimental data much better if the external electric field is coupled to the charges only but not to the dipoles themselves.

2.5. Elastic Constants

There are two reasons why the calculation of elastic constants is crucial to test whether the model describes the α-β transition well. The first is that the elastic constants are

very sensitive to small changes in the model potential energy surface and second they show distinct features at the α-β transition in quartz, as discussed recently by Carpenter et al. [13].
The calculation of the elastic constants in our simulations was done in the NpT ensemble using the fluctuations of the strain tensor $\underline{u}$ [72]. The strain tensor can be written in terms of the simulation box $\underline{\underline{h}}$ as

$$\underline{u} = \frac{1}{2}\left(\underline{\underline{h}}_0'^{-1}\underline{\underline{h}}'\underline{\underline{h}}\,\underline{\underline{h}}_0^{-1} - \underline{\underline{1}}\right) = \frac{1}{2}\left(\delta\underline{\underline{h}}\,\underline{\underline{h}}_0^{-1} + \underline{\underline{h}}_0'^{-1}\delta\underline{\underline{h}}'\right) + \mathcal{O}\left(\delta\underline{\underline{h}}^2\right) \tag{2.5}$$

Now the elastic constants can be evaluated with the help of the fluctuations of this quantity:

$$C_{\alpha\beta\gamma\delta} = \frac{V}{k_B T}\left\langle \delta u_{\alpha\beta}\delta u_{\gamma\delta}\right\rangle^{-1} \tag{2.6}$$

As the $\underline{\underline{h}}$ matrix is symmetric in the Parrinello Rahman method, derived tensors like the strain tensor and the elastic constants will also be symmetric and can be written in Voigt notation (see chapter 1.1.1). This reduces the 4th rank tensor of dimension 3 to a square matrix of dimension 6. In a crystal the number of independent coefficients reduces even more, related to the crystal symmetry class. For α-quartz, which is a trigonal holoaxial system, the elastic constants have the following symmetry:

$$\underline{\underline{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} & 0 & 0 \\ C_{12} & C_{11} & C_{13} & -C_{14} & 0 & 0 \\ C_{13} & C_{13} & C_{33} & 0 & 0 & 0 \\ C_{14} & -C_{14} & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{44} & C_{14} \\ 0 & 0 & 0 & 0 & C_{14} & \frac{1}{2}(C_{11}-C_{12}) \end{pmatrix} \tag{2.7}$$

With ascending symmetry the number of independent elastic constants decreases. Thus when going to the β-quartz configuration, which is a tetragonal holoaxial system, the crystals gains symmetry and therefore loses independent entries in the matrix, in this case C_{14} will vanish. Note that C_{14} can only be non-zero for systems that lack inversion symmetry, as in tensor notation C_{14} becomes C_{1123} in which the indices 2 and 3 only occur once. This is a reflection of quartz lacking inversion symmetry, see Fig. 2.20. Consequently, the sign of C_{14} depends on the chirality in quartz, i.e., whether we have left-handed or right-handed quartz. The elastic constants tensor of β-quartz looks like

$$\underline{\underline{C}} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & 0 & 0 & 0 \\ C_{12} & C_{11} & C_{13} & 0 & 0 & 0 \\ C_{13} & C_{13} & C_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & C_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & C_{44} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(C_{11}-C_{12}) \end{pmatrix} \tag{2.8}$$

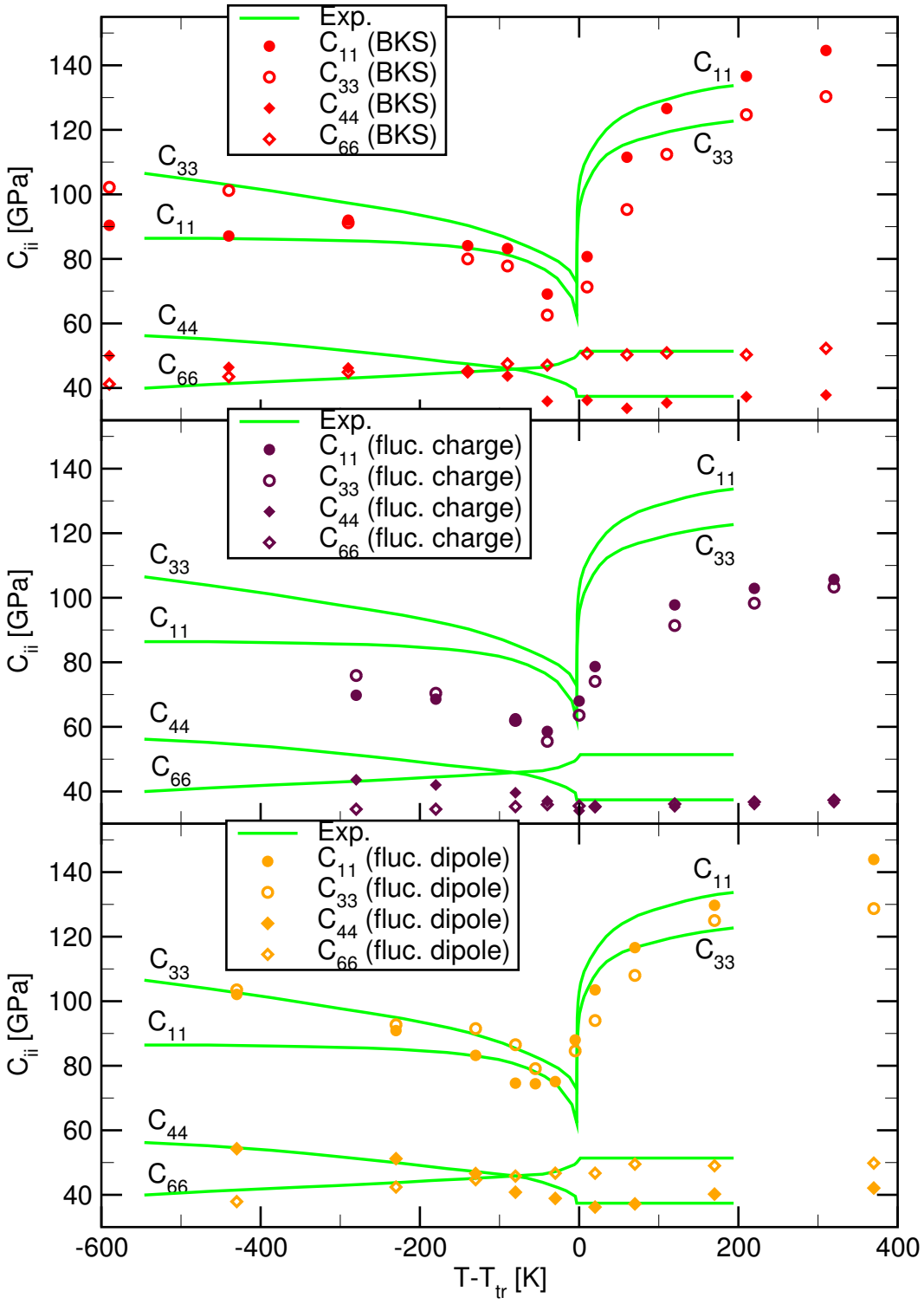


Figure 2.21.: Elastic Constants

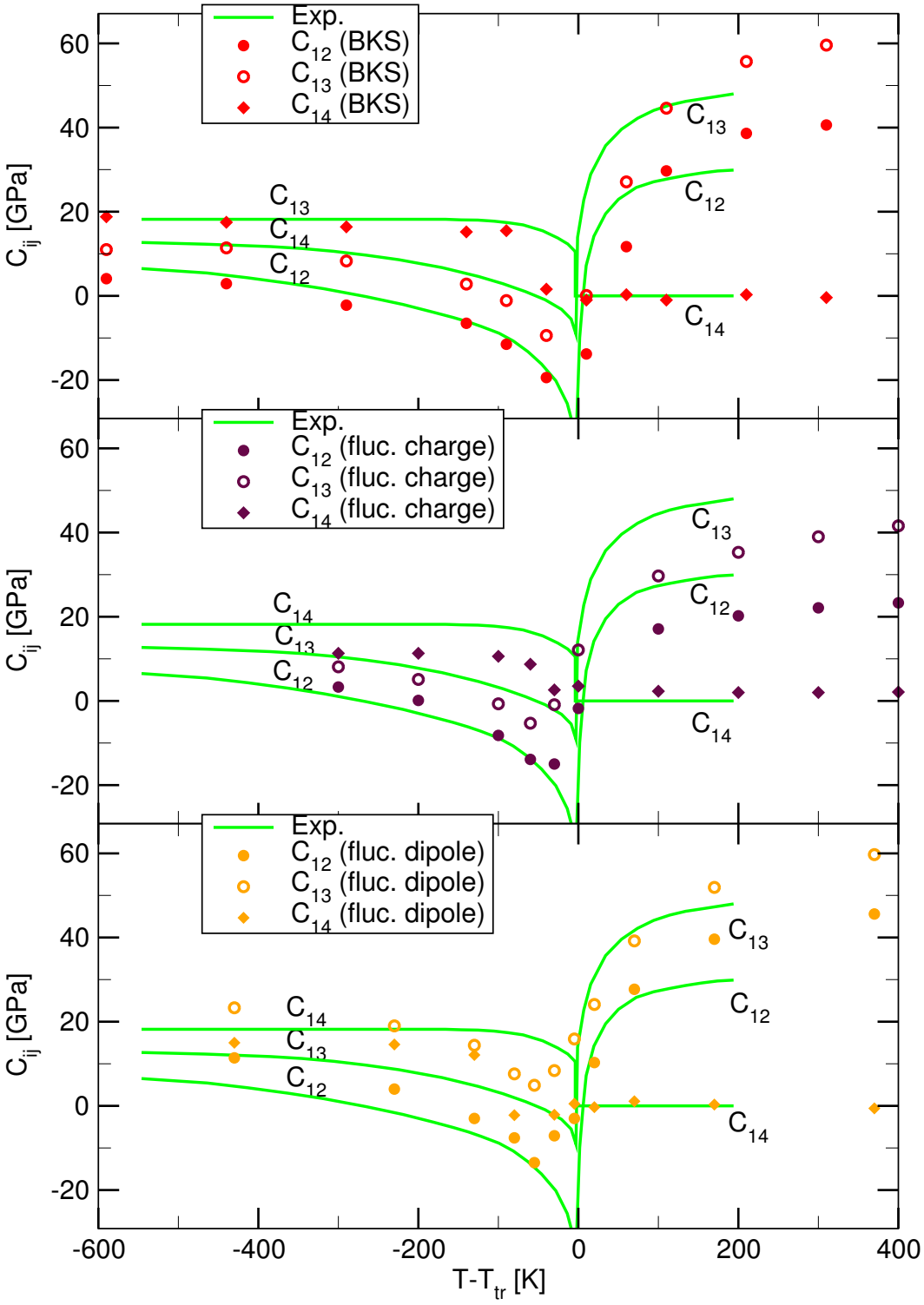


Figure 2.22.: Elastic Constants

The simulation results for the elastic constants are shown in Fig. 2.21 for the relevant diagonal elements $C_{11}, C_{33}, C_{44}, C_{66}$ and in Fig. 2.22 for the relevant non-diagonal elements C_{12}, C_{13}, C_{14}. The curves are shifted in temperature to match the respective transition temperatures of the models.
All models show the general behaviour that there is a marked softening in C_{11}, C_{12}, C_{13}, and C_{33} as $T \rightarrow T_{tr}$. Only the C_{44} and C_{66} do not vary so much with temperature and not at all in the stability field of β-quartz, while C_{14} is quite constant in α-quartz and vanishes in the β-quartz phase due to symmetry reasons.
As already shown in Ref. [66], BKS shows good qualitative agreement with experiment, also for the strong temperature dependence near T_{tr}. Also most experimental features are well reflected, e.g. C_{11} and C_{33} cross between high and low temperatures similar to C_{44} and C_{66} (both shown in Fig. 2.21), while C_{12} and C_{13} do not cross. This good agreement is not too surprising considering the way in which the BKS parameters where determined: ab initio data along with elastic moduli were used to construct the potential energy surface.
In comparison to BKS the agreement of the fluctuating charge potential with experiment is not too convincing. While the qualitative trends are reproduced quite well, almost every value is significantly lower than in experiment.
The fluctuating dipole potential is in better agreement, but does not improve on the BKS behaviour. While the agreement in α-quartz is better than BKS for C_{44}, it is less good for C_{11}. However the values for β-quartz seem to have similar deviations for the experimental values. Close to the transition temperature the softening is generally more pronounced in all simulations due the finite system size, but at higher temperatures BKS and the fluctuating dipole potential agree on higher values for C_{12} and C_{13} than the experiment.

2.6. Phonon Density of States

A quantity of interest in computer simulations of amorphous as well as crystalline systems is the vibrational density of states (DOS). In general it is a measurement for the lattice dynamics, as it shows the distribution of vibrational states over frequency. This means $g(\nu)d\nu$ give the number of eigenstates in the frequency range ν and $\nu + d\nu$. While it can also be measured experimentally by neutron scattering, the simulation gives insight to where the microscopic origin of the distribution comes from. For example the partial distribution functions are often shown, which take into account only the contributions from a certain atom type, which is a quantity not possible to be measured experimentally in general. However it was shown recently [7] for the example of BKS in the simulation of amorphous silica, that the density of states is not well reproduced by this model. Even though the structure of vitreous SiO_2 was shown to change

only weakly by a subsequent first-principles calculation, the DOS was strongly modified by using an ab initio treatment of the forces, and that this lead to a much better agreement with experimental results. Moreover, in a large frequency range the nature of the excitations as determined from the effective potential differed significantly from the one determined from the ab initio force. This calls into question the explanatory power of an analysis of the nature of the vibrational excitations determined from the BKS force field. On the other hand it gives rise to the question if other effective force fields can do better, as ab-initio calculations suffer from their heavy computational cost which restricts this type of calculations to the study of very small systems with rather low statistical accuracy.
The definition of the vibrational density of states reads

$$g(\nu) = \frac{1}{3N} \sum_{i=1}^{3N} \delta(\nu - \nu_i), \tag{2.9}$$

with N being the particle number and $\nu_i, i = 1, \ldots, 3N$ the eigenfrequencies of the dynamical matrix $\underline{\underline{D}}$. A less laborious way is the calculation in harmonical approximation, which only requires to calculate the Fourier transform of the velocity auto correlation function:

$$g(\nu) = \frac{1}{Nk_BT} \sum_j m_j \int_{-\infty}^{\infty} dt \left\langle \underline{v}_j(t).\underline{v}_j(0) \right\rangle e^{i2\pi\nu t}. \tag{2.10}$$

This was calcluated in the simulations by averaging the velocity auto correlation function over time intervals of 2048 steps, which corresponds to real time intervals of 2 ps, allowing for a resolution of $1/2\text{ps} = 0.5$ THz in the vDOS.
The most obvious disagreement between the lattice dynamics of a BKS simulation and an ab initio simulation can already be seen directly in the vibrational DOS in Fig. 2.23. The figure shows a comparison of α quartz at 300 K with ab initio data from Roma et al. [79], as no experimental measurement of the DOS in quartz could be found. The figure shows that the inter tetrahedral motions, the so called rigid unit modes which account for the lower frequency range, show a significant discrepancy, as it is already known for amorphous systems. In the higher frequency range the double peak structure, which is available in both of the curves, comes about due to intratetrahedral stretching vibrations [32, 73]. The four oxygen atoms in an SiO_4 tetrahedron are moving at the higher frequency ("breathing mode") in phase in relation to the central Si atom, at the lower frequency two O-Atoms oscillate in anti-phase to each other. However the frequency of these oscillations seems to be shifted from 32.1 THz for the low frequency peak, according to neutron scattering in vitreous silica [75], to around 30 THz in α quartz, according to the ab initio data at hand. The lower peak of the BKS potential, which coincides with the experimental frequency quite well, is however shifted rather to a higher frequency of about 33 THz in α quartz.

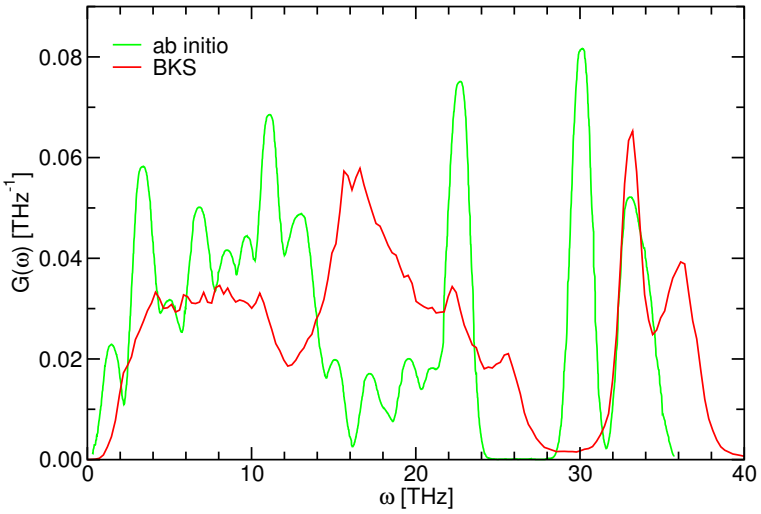


Figure 2.23.: Phonon Density of States in α-quartz at 300K. Comparison of BKS and ab initio data.

The fluctuating charge model shows a slightly better agreement with the ab initio data

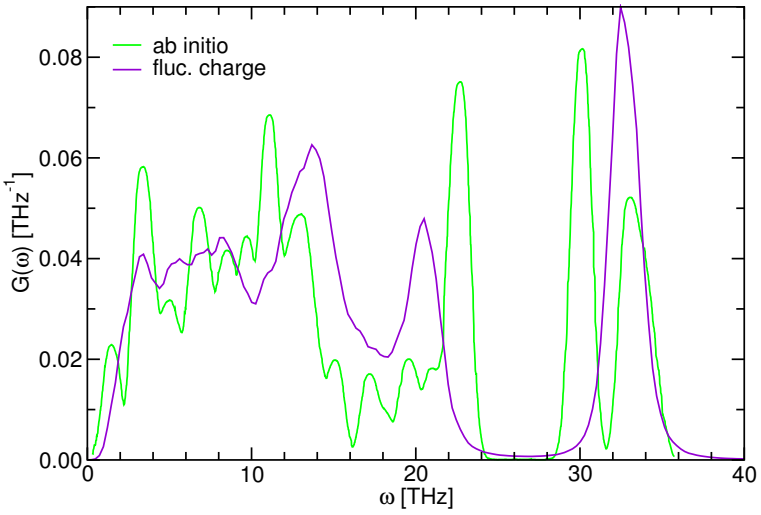


Figure 2.24.: Phonon Density of States in α-quartz at 300K. Comparison of the fluctuating chare potential and ab initio data.

where the lower frequencies are concerned. The two peaks at 11 THz and 22 THz at least exist close by and the structure is approximately maintained. The two peaks of the intra tetrahedral modes collapse on just a single peak, but exist at roughly the right frequency.

The best overall agreement for both the rigid unit modes and the double peak is clearly shown by the fluctuating dipole potential. This is not too surprising as it was parametrized to match ab initio forces and stresses, but also the BKS potential was

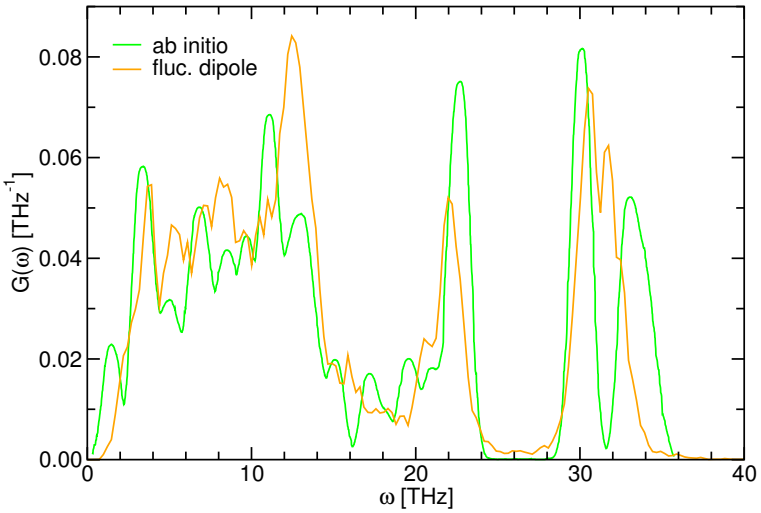


Figure 2.25.: Phonon Density of States in α-quartz at 300K. Comparison of the fluctuating dipole potential and ab initio data.

parametrized to quartz data, even though the DOS looks better for vitreous silica. The fluctuating dipole potential is almost congruent with the ab initio result in this case, only the tetrahedral breathing mode is slightly shifted in frequency. This convincing agreement suggests a good credibility for the lattice dynamics in this force field.

3. Pressure-Driven Transitions in α-Quartz

The high-pressure behaviour of α-quartz is not yet fully understood neither on the theoretical and nor on the experimental side. The majority of polymorphs that are observed at ambient to moderate pressures, such as cristobalite and quartz as well as other polymorphs including tridymites and coesite, are built up of SiO_4 tetrahedra [39]. At higher pressures dense forms containing SiO_6 octahedra are observed such as stishovite above 9 GPa, which has a rutile type structure. Because of the relatively strong Si-O bonding in silica, there are high kinetic barriers associated with the transitions to stable phases containing octahedrally coordinated silica at high pressure.
As the pressure is increased, the four-coordinated structures become increasingly unfavorable and eventually the polymorph become unstable. New phases are formed that can be either crystalline or amorphous. The newly formed phases can often be reached relatively easily by local rearrangement of the atoms, however these phases are often only thermodynamically metastable [52, 53, 105, 97, 99]. The transition path way from quartz to stishovite, which is the proper high-temperature polymorph at elevated pressure, would be highly reconstructive and is thus kinetically suppressed. Because of slow kinetics, the large number of possibilities to form six-coordinated structures from four-coordinated structures, the metastability, but also experimental difficulties such as low x-ray scattering factors, the phase behaviour of high pressure quartz is still not completely resolved.
The high-pressure behaviour of α-quartz is particularly complex [52]. The stable phases of SiO_2 in the order of increasing pressure are quartz, coesite (from about 3 GPa), and stishovite (from about 7 GPa), as shown in the phase diagram in Fig. 0.1. However, at room temperature the transformations are kinetically inhibited. Consequently, quartz can be kept metastable up to 21 GPa. Initially a pressure induced amorphization was observed in α-quartz [40], but soon it was discovered, that some of these new materials formed at high pressure were found to exhibit unusual properties for amorphous solids such as elastic anisotropy [64] and memory effects [55]. The experimental observations at this point are somewhat complex, but it is clear that around 21 GPa a transformation to a quartz II phase of unknown structure occurs. The amorphous phase develops out of this new phase and grows with pressure until a fully amorphous structure is reached above 30 GPa. Kingma et al. [52] report that samples recovered from above 30 GPa appeared to be entirely amorphous, while those

recovered from below this pressure contained crystalline regions. This explains the unexpected anisotropic properties of the under X-ray apparently amorphous phase. A different result was obtained by Haines et al. [38], who saw a quartz II structure only as an intermediate phase from 19 to 26 GPa, which reverted for higher pressures to a monoclinic $P2_1/c$ phase.

3.1. Quartz II Transition in Simulations

For molecular dynamics simulations employing the BKS potential it was shown that a phase transition occurs, when α-quartz is compressed to 22 GPa [95, 96]. This transition turned out to be not reversible [83]. It is crucial for the transition to occur that

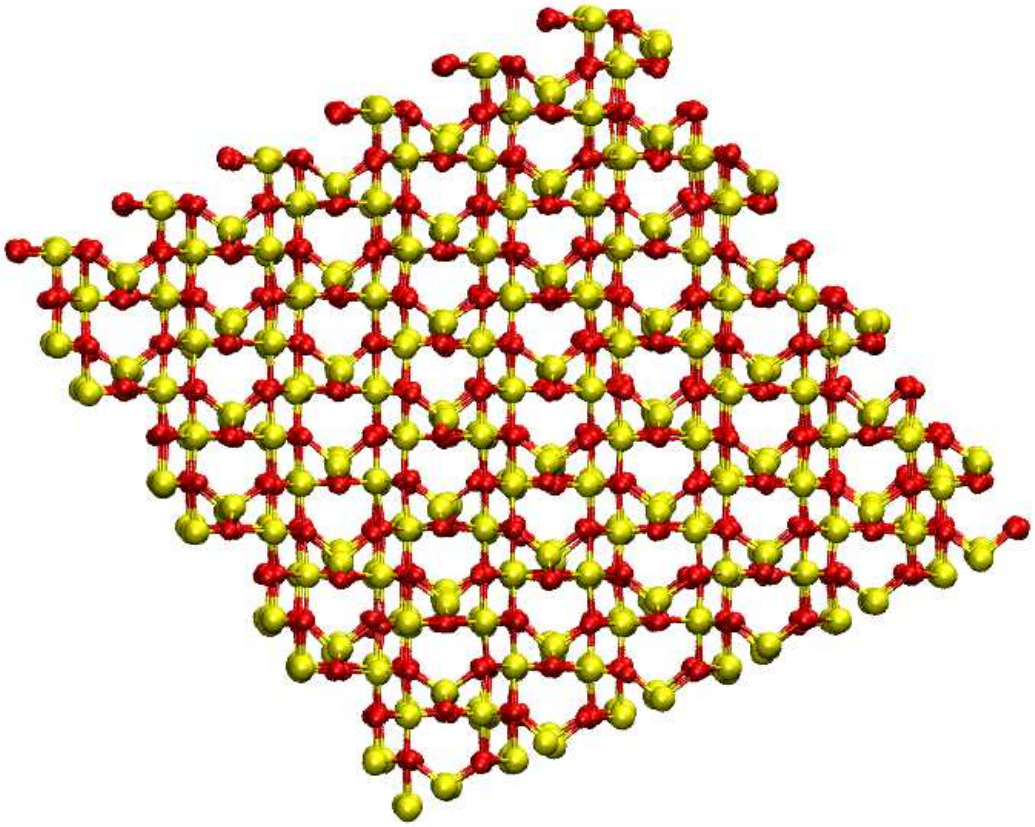


Figure 3.1.: Snapshot of a Quartz II configuration with 2160 atoms. The configuration is rotated to align the crystal axis horizontally, but it can still be seen that the box is sheared rather than orthorombic.

a constant pressure regime with a fluctuating box geometry is used, as the new phase can only be established from the α-quartz configuration when the box is allowed to shear, which is shown in Fig. 3.1. The new phase is perfectly crystalline, which is obvious from the structure snapshot, but it is subtle to achieve. In order to end up in this crystalline phase and not in an amorphous phase, the fictious mass of the box has to be sufficiently small, and the particle number has to be large. For smaller particle number the transition would occur at a pressure of about 28 to 30 GPa. This is because at pressures such high α quartz is far beyond its stability range anyway and so it depends on various conditions if the metastability holds or if the crystal converts to a different phase. However the pressure of 22 GPa is the smallest that could be found and is certainly very comparable to experimental findings, which makes it a likely candidate for the unknown Quartz II structure.
From the pair correlation function or actually by viewing a snapshot of the system it can be seen that only a third of the silicon atoms is four coordinated, but two thirds

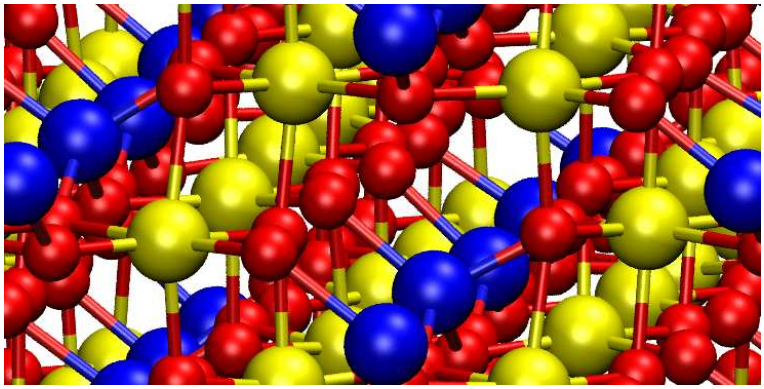


Figure 3.2.: Silicon Coordination in Quartz II. Four coordinated silicon is printed in blue while the six coordinated silicon atoms are shown in yellow.

of silicon atoms have six neighbours, as it is common for the crystalline high pressure phases of SiO_2. This can be seen in Fig. 3.2, where in a snapshot of the configuration the silicon atoms are coloured according to their coordination number. The four coordinated Si atoms are in the centres of SiO_4 tetrahedra, while the six coordinated atoms are surrounded by oxygen atoms in the corners of an octahedron.

3.1.1. Transition Path Ways

The density changes in a compression/decompression cycle are shown in the equation of state in Fig. 3.3. Starting with the α-quartz configuration at zero pressure, the pressure is increased up to 21 GPa, where the system collapses from the α-quartz configuration into the quartz II phase. The transition is discontinous in the volume and

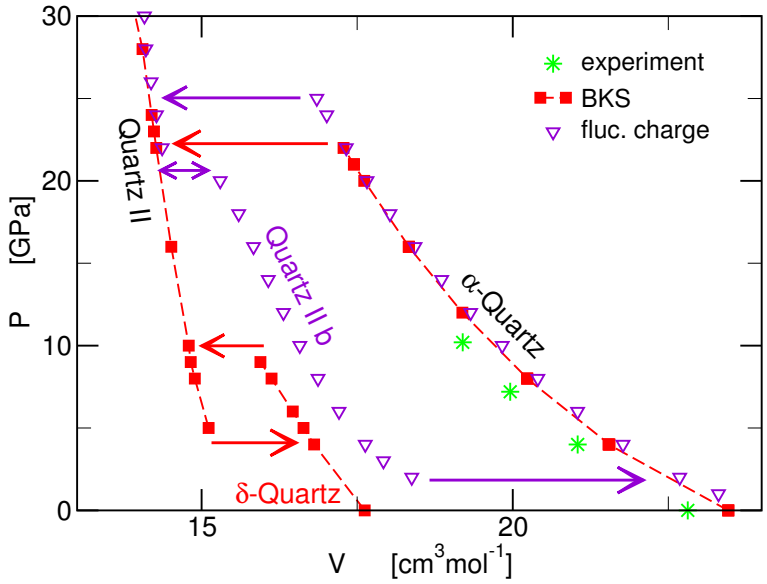


Figure 3.3.: Equation of State for α-quartz under pressure for BKS and fluctuating charge potential.

irreversible: in the simulation the new phase remains stable upon further compression and even on decompression below the transition pressure. Even at small pressures the cyrstal does not revert to the α-quartz structure, but transforms into a new δ-quartz phase. This phase transition is reversible with a hysteresis, but is also non-displacive. The δ-quartz phase is stable within the BKS potential even at ambient pressure and would not revert to α-quartz.
For the fluctuating charge potential the equation of state for different quartz phases is also shown in Fig. 3.3. The phase transition occurs at a slightly higher pressure of 25 GPa, and the volume of both the α quartz and the quartz II phase agree very well with the BKS results. Nevertheles, the quartz II phase does not remain stable on decompression, but the system undergoes a reversible phase transition at 21 GPa to a structurally similar phase named quartz II b in the figure. This phase has a smaller density than the BKS quartz II phase at the same pressure and reverts to the α quartz phase at 4 GPa. The δ-quartz phase of BKS is not produced. Moreover the δ-quartz phase is not even stable for the fluctuating charge potential but quickly transforms to α quartz.
These findings for the fluctuating charge potential are also valid for the fluctuating dipole potential. As shown in Fig. 3.4, the variation of the volume with pressure is identical to the fluctuating charge potential with respect to its qualitative features. The transition temperature to quartz II is 27 GPa now, and it has to be noted that the transition only occurs for small systems. For larger systems the α-quartz configuration is stabilized for even much higher pressures. The transition to quartz II b occurs also at 21 GPa and the next transition to α-quartz will take place not until 2 GPa. Also in this potential the BKS δ-quartz phase is not stable and thus seems to remain an artificial

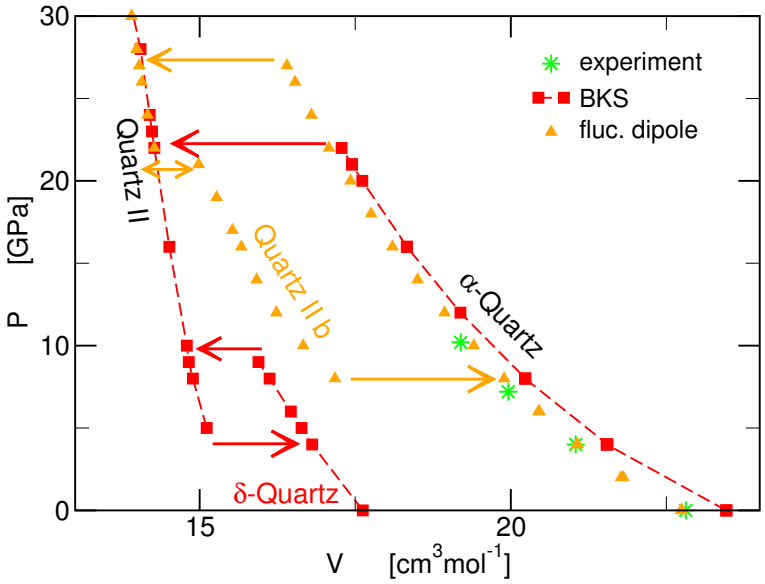


Figure 3.4.: Equation of State for α-quartz under pressure for BKS and fluctuating dipole potential.

polymorph of the BKS potential.

3.2. Quartz II b

The quartz II b phase, which does not form in a BKS simulation out of quartz II, emerges from quartz II phase with decreasing pressure at around 20 GPa for both the fluctuating dipole and the fluctuating charge potential. The connectivity of the atoms remains the same as in quartz II during the transition and the structure change is not visible in a snapshot nor in the scattering spectra, but the change in the pair correlation function is obvious as shown exemplary in the Si-O pair correlation for the fluc. dipole potential in Fig. 3.5. From 23 to 20 GPa the cyrstal is in the quartz II phase and the pair correlation function changes only gradually. At the transition to the quartz II b phase at 19 GPa the pair correlation function exhibits a discontinuous jump, and is changing only slightly on further decompression down to 17 GPa within the quartz II b phase. The phase transition is reversible when the pressure is increased again, while there is a hysteresis effect that implies a higher transition pressure of about 2 GPa on recompression. These findings indicate a (quasi) displacive crystalline-crystalline first-order transition.

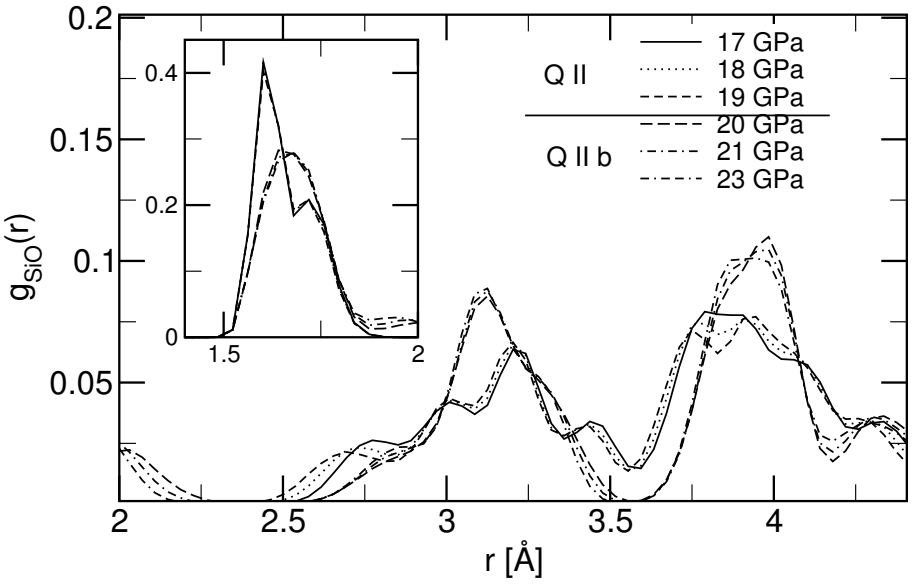


Figure 3.5.: Si-O Pair correlation for the fluctuating dipole potential for different pressures around the Quartz II - Quartz II b transition. The pressure was decreased subsequently starting from quartz II.

3.3. X-Ray Diffraction Spectra

The similar behaviour of the three model potentials at the quartz II transition and the apparent similarity of the crystalline structure emerging from these simulations gives the simulation results a certain credibility for predictions towards the experiment. In experiments the structure of a crystal can be characterized by its diffraction pattern, such as the energy dispersive x-ray diffraction spectra measured for quartz II by Kingma et al. [52]. To measure these data in a simulation one has to calculate the structure factor

$$S(q) = \left| \sum_{j,k} f_j f_k \exp\{i\underline{q}(\underline{R_j} - \underline{R_k})\} \right|^2 = \left| \sum_j f_j \exp\{i\underline{q}\underline{R_j}\} \right|^2 \tag{3.1}$$

with the appropriate atomic form factors f_i. These are tabulated as a function of $\sin(\Theta)/\lambda$ in Ref. [61], pp. 202, 203. A diffraction angle of 5° and a Gaussian broadening with a 0.5 eV FWHM were used to generate the spectra. The wave vector is related to $\sin(\Theta)/\lambda$ via the Bragg condition $\lambda = 2d\sin\Theta$ as $q = 4\pi sin(\Theta)/\lambda$.
The reciprocal lattice vectors have to fit in the simulation box: $\underline{q} = 2\pi\underline{\underline{h}}^{-1}(n_x, n_y, n_z)$, which simplifies eq. (3.1) to

$$S(\underline{q}) = \left| \sum_j f_j \exp\{2\pi i(n_x, n_y, n_z)\underline{r_j}\} \right|^2 . \tag{3.2}$$

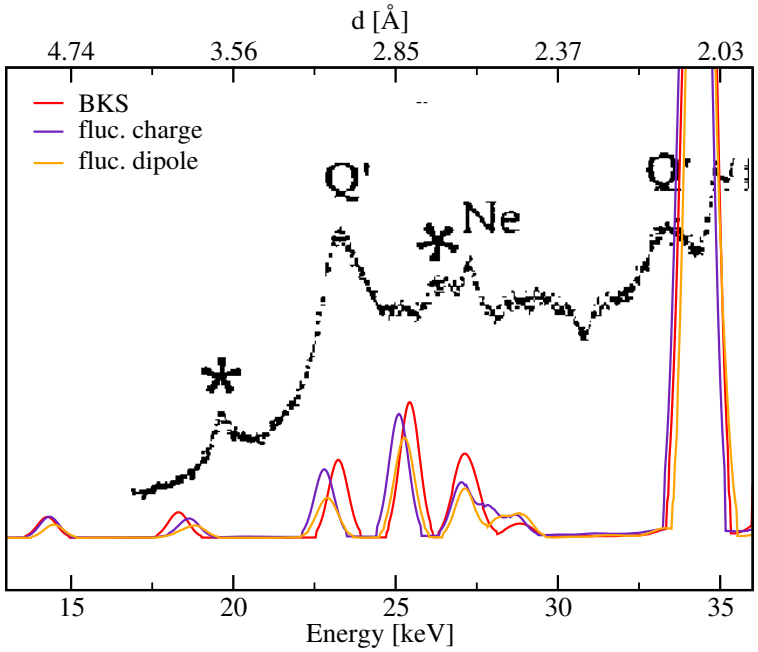


Figure 3.6.: Energy dispersive x-ray diffraction patterns of Quartz II at 27 GPa. The experimental x-ray diffraction spectrum shown in black is taken from Kingma et al. [52]. The Q' symbols represent shifted quartzlike peaks while the * depict new nonquartz peaks. Ne denotes diffraction from neon, which is used as pressure medium.

The diffraction spectra for the different potentials are compared to the data by Kingma et al. in Fig. 3.6. The three potentials are in good agreement with each other concerning the structure of quartz II and can be related quite reasonably to the features of the experimental spectrum.
An overview comparison for the spectra of the three potentials is given in Fig. 3.7, where one can see the course of the peaks in the α quartz phase and the change to the quartz II peaks at the different transition temperatures. The position of the peaks can be reexpressed as a Bragg distance between diffraction planes. The Bragg distances as a function of pressure can then be compared to the data given by Kingma et al. In Fig. 3.8 the interplanar spacings are shown in comparison with the experimental data. From this figure and fig. 3.6 it can be summarized that the features of the experimental spectrum agree quite well with the simulations. The apparent absence of the interplanar spacing around 3.6Å in all simulations might be explained with the fact that the peaks at 3.8Å in the simulation consists of two very close peaks which are not resolved in the plot. On the other hand the apparent absence of the 2.8Å Bragg distance in experiment might arise out of a misinterpretation of the experimentally complicated energy range around 27 keV. In this energy range one peak arises from Neon that was used as pressure medium in experiment, which might cover the additional interplanar spacing

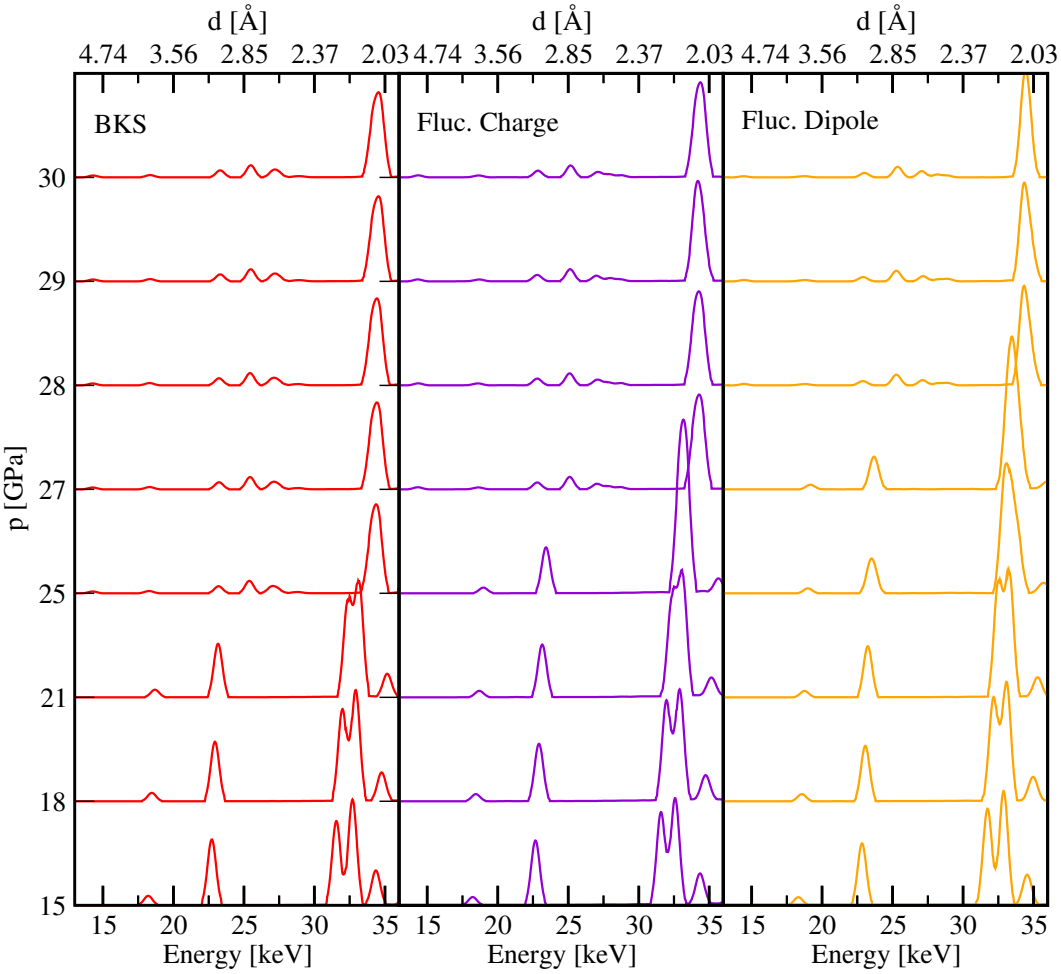


Figure 3.7.: Comparison of X-Ray spectra at various pressures for the BKS potential (left), the fluc. charge potential (middle), and the fluc. dipole potential (right)

which is predicted by all of the simulations. In total it seems that all the three model potentials support the structure seen by Kingma. Still these results are questionable as the models were parametrized for ambient conditions and are not verified to reproduce the real behaviour under such extreme pressures, but their stunningly similar findings adds to the credibility of the results.

3.4. Other High-Pressure Phases

Wentzcovitch et al. [105] found a high pressure phase with symmetry $P3_2$ in simulations at 40 GPa. These simulation were done with an ab-initio MD approach with

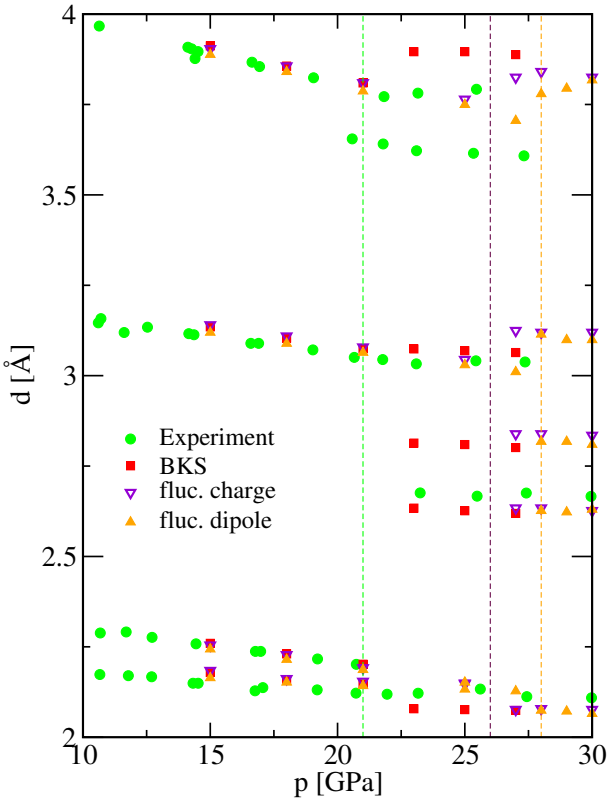


Figure 3.8.: Interplanar spacing for the compression of α-quartz in experiment and simulation. Lines represent the different transition pressures. Experimental data by Kingma et al. [52].

variable cell shape and a box of 27 particles. This high-pressure phase phase did not emerge in our simulations from α-quartz, however when using the crystal structure reported by Wentzcovitch as initial configuration, it proved to be stable under high pressures between 30 and 40 GPa with the fluctuating dipole potential and from 40 GPa down to 15 GPa with the BKS potential. In the fluctuating charge approach the phase is only stable at around 30 GPa.

At lower pressures, and even down to zero pressure, structures are forming in all potentials that are not entirely crystalline and not related to any known quartz phases. The density of the crystal at 40 GPa can be calculated from the lattice constants given in Ref. [105] to be 4.60 g/cm^3, while the BKS potenial estimates a density of 4.53 g/cm^3 and the fluctuating dipole potential finds 4.56 g/cm^3. However the polymorph proposed by Wentzcovitch et al. seems to be a valid option for a high-pressure phase of silica in the BKS and fluctuating dipole potential as well, however it is not nearly as stable as the Quartz II phase in the three potentials under consideration.

4. Silica Polymorphs Other Than Quartz

4.1. Cristobalite

Quartz is not the only polymorph that is (meta) stable at ambient conditions. Cristobalite, a cubic polymorph, and tridymite, which has hexagonal symmetry, are found to be stable as well during accessible experimental time scales [47, 23]. A model potential that is used to mimic the behaviour of silica should be able to reflect the experimentally observed stabilities. In some cases, the observation of stability in a simulation might be fortuitous as it could be the consequence of periodic boundary conditions and/or the number of atoms used. For example, the phase into which the system would like to convert is not compatible with the number of atoms used in the simulation or it cannot be accessed without a major reconstruction of the cell shape. This issue will be discussed in more detail in this section with a focus on cristobalite that shows subtle effects depending on the choice of the periodic boundary conditions. For the simulation of a crystalline system, the crystal structure has to be placed inside the box in a way that the periodic images fit to each other at the box ends. For the initial configuration of a simulation the atom positions are chosen to be identical with experimental crystallographic data such that the atoms were set onto ideal lattice positions and are equilibrated. For many systems and also for the case of cristobalite there are different ways to arrange such an initial configuration.
The most natural configuration is according to the usual description of cristobalite as a diamond structure for the silicon atoms with connecting oxygen atoms between each neighboring pair of silicon atoms. This would place the [100] crystal direction parallel to the z axis of the simulation box, and the box geometry would be cubic.
On the other hand there is a nice ABCABC layering of sheets of hexagonal rings in the cristobalite structure, which is also a very common representation as it leads to tridymite when the layers are stacked in an ABAB order. This layering is seen along the [111] direction in the crystal, and accordingly would place this direction parallel to the z axis of the simulation box when the layers are stacked in the xy plane.
However both geometries are completed by the periodic boundary conditions to form an infinite cristobalite crystal, and therefore they are equivalent regarding the static structure. On the other hand the connectivity of the atoms at the boundary is different and therefore the two geometries are not equivalent regarding phonons. When a lattice

vibration leaves a periodic image it would reenter the box at a different position for both setups.
This difference becomes obvious in the simulation of β-cristobalite. When relaxing

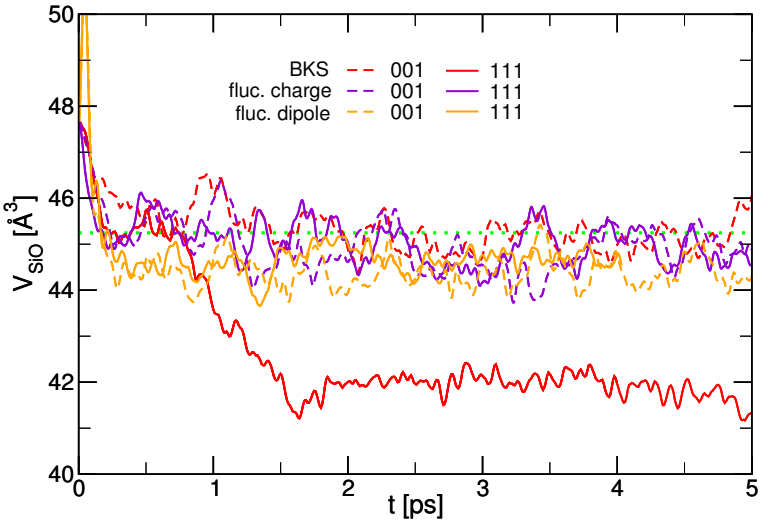


Figure 4.1.: Time evolution per SiO_2 unit V_{SiO_2} for the two geometries with three different potentials.

the crystal at $T = 1273$ K and zero external pressure, where cristobalite is considered to be thermodynamically stable [39], the volume of the simulation quickly stabilizes on a value close to the experimental value of $V_{SiO_2} = 45.25$ Å^3. In contrast with the second configuration, which had the [111] direction parallel to the z axis, the observed voulme is, if at all, metastable only for a short period of time, before densification of the system takes place. After 2 ps the system is stable in a phase which is much denser than cristobalite. It was tested that this state remains stable for more than a nanosecond, which is not shown in the picture.
Obviously in contrast to this both the fluctuating charge and the fluctuating dipole potential do not exhibit such a behaviour, but remain stable in the cristobalite phase regardless of the initial condition.
Apparently the cristobalite phase is not universally stable in the BKS potential, but only if certain phonons are prohibited by special periodic boundary conditions.

4.2. Tridymite

Another high temperature polymorph of quartz is tridymite, which was accepted to be a stable phase of silica as late as in the 1960's. There is a broad range of structural modifications upon heating, while the structures increase in complexity with decreas-

ing temperature. The ideal high-temperature tridymite, the HP-tridymite, is stable over 650 K. Its structure consists of an AB layering in c direction of the same tetrahedral sheets that also form the cristobalite phase when stacked in a tripled ABC repeat along [111].
We found that the HP-tridymite structure is not stable in BKS at 1000 K, which is well

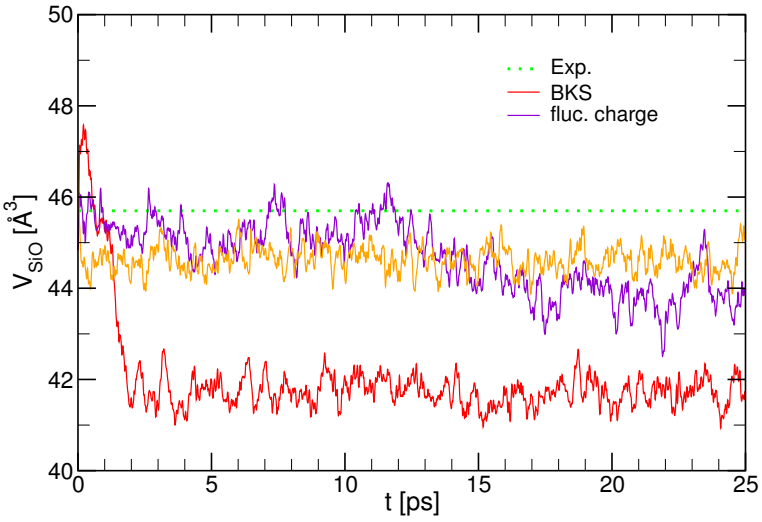
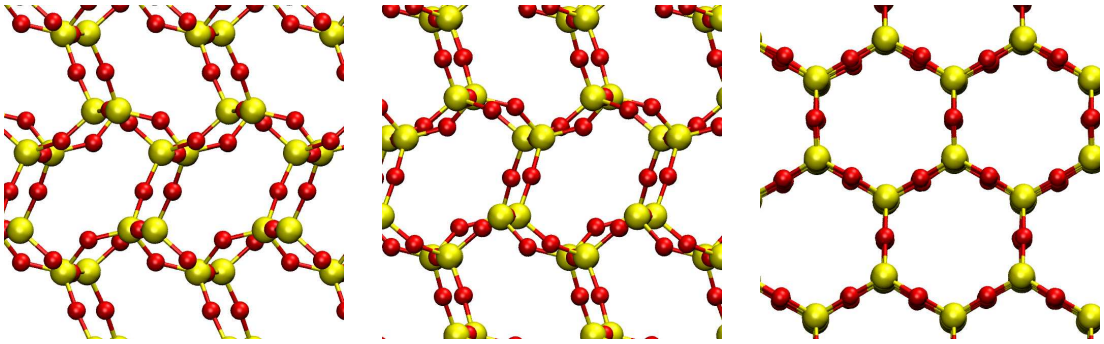


Figure 4.2.: Time evolution per SiO_2 unit V_{SiO_2} of tridymite for the three different potentials at T=1000 K.

in the experimental stability range. Surprisingly, also the fluctuating charge potential, in which cristobalite turned out to be stable, suggested in contradiction to experiment that the HP-tridymite configuration is unstable. Shown in Fig. 4.2 is the time evolution of the volume in a simulation run. While the crystal simulated with the BKS potentials collapses into a denser phase about as quickly as the cristobalite collapses, with



a) BKS **b)** Fluc. Charge **c)** Fluc. Dipole

Figure 4.3.: Snapshot of the different phases that emerged after 20 ps at 1000 K. In all cases the initial structure was HP-tridymite. Only the fluctuating dipole potential remained in the initial phase, which is stable experimentally.

the fluctuating charge potential the HP-tridymite is metastable for about 10 ps in the simulation, only then it collapses slowly afterwards. The fluctuating dipole potential is the only one that predicts that the HP-tridymite phase is stable.
In Fig. 4.3 this is shown in terms of the averaged configuration. One can see that the collapsed structure of BKS and the fluctuating charge potential looks quite similar. The densification is mainly achieved by a contraction in the a direction (vertical). The framework collapse induces distortions of hexagonal rings to oval configurations. Additionally the tetrahedrons are tilted relative to the stacking direction c and thus adjacent sheets are displaced along b. While the later is known to occur in the tridymite phases at intermediate temperatures, these phases all have a superstructure of characteristically alternating oval and ditrigonal configurations. The occurence of only oval rings in the present simulations may be caused by the finite box size, which does not allow the superstructure to emerge.

4.3. Stishovite

The quartz, cristobalite and tridymite phases studied in this work were composed out of corner-sharing SiO_4 tetrahedra, which is the usual configuration in the low pressure phases. Stishovite in contrast is a polymorph that often serves as a prototype phase having octahedrally coordinated silicon. It is often assumed [20, 90] that with the fluctuating charge approach the ability of the charges to adapt to the configuration is especially helpful for the transferability between differently coordinated structures.

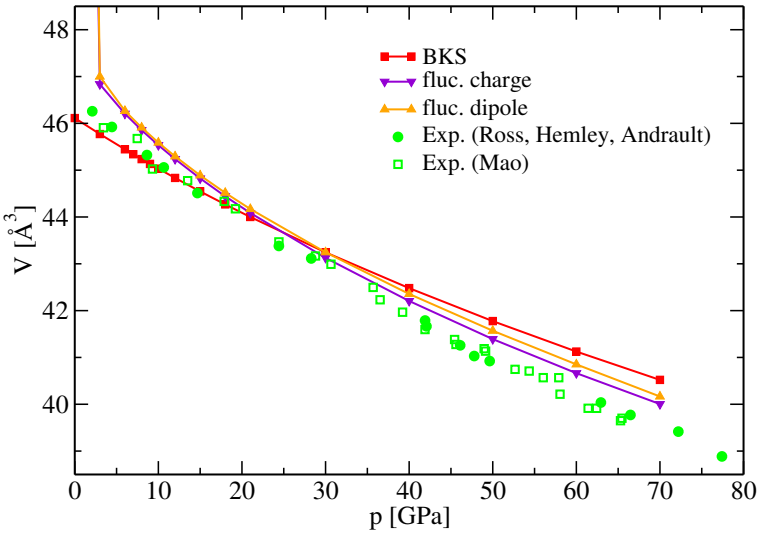


Figure 4.4.: Stishovite Equation of State. Solid circles are the data of Ross et al. [80], Hemley et al. [43], and Andrault et al. [2]; open squares are the data of Mao et al. [62]

Also it is very likely that the degree of ionicity changes with pressure. Therefore, one might expect that the fluctuating charge potential comes closest to the available experimental data out of the three models.
Stishovite is usually stable at high pressures, above 8 GPa at room temperature, but it was also discovered to be metastable at ambient pressures. It was first synthesized in the laboratory [89] and later was discovered in association with coesite [14]. Stishovite has the rutile structure, which consists of infinite chains of edge-shared SiO_6 octahedra parallel to the c-axis. Each oxygen atom is coordinated by three silicon atoms [43].
The equation of state for stishovite under pressure is shown in Fig. 4.4. The agreement in the low pressure range is relatively good for the BKS potential, while the fluctuating dipole and the fluctuating charge potential both do not see a stable stishovite phase at zero pressure. When increasing the pressure, these two potentials constantly underestimate the density of stishovite, while the volume of the BKS simulation deviates more for higher pressures.
One feature of stishovite under pressure is a different compressibility in the a and c direction, which is shown in Fig. 4.5. This plot shows again a very good agreement of

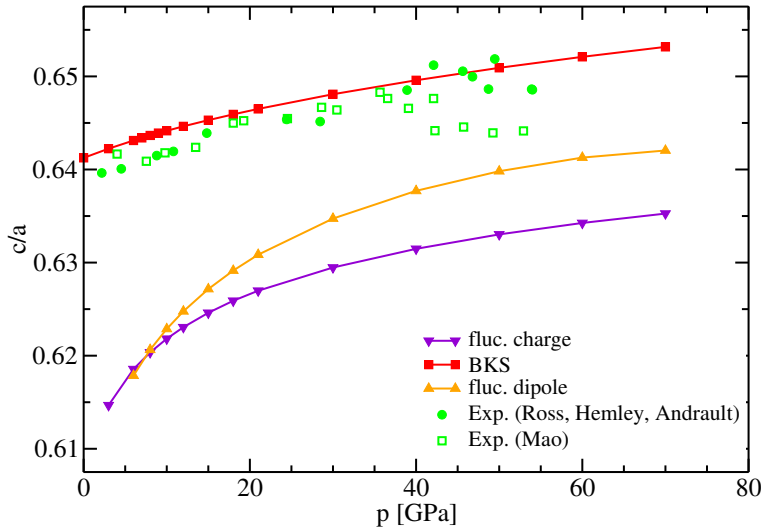


Figure 4.5.: c/a ratio in stishovite against pressure. Experimental data as in Fig. 4.4. Please note that the phase transition discussed in the next section does not have much influence on the c/a ratio and the volume.

the BKS data with the experimental results. The effect of the different compressibility in the fluc. charge and fluc. dipole potential is rather overestimated, while the fluctuating dipole potential again improves in the quantitative prediction for higher pressures. The result of the fluctuating charge potential shows no improvement over neither of the two other potentials.

4.3.1. High-Pressure Behaviour

Shortly after the discovery of stishovite, a major question has evolved around possible transformations to a denser structure at high pressures. By crystal chemical arguments [67], first-principle calculations [42, 15], and experimentally [41, 98, 51, 12] a pressure-induced phase transition at 50 GPa was confirmed. At the transition the SiO_6 octahedra are only slightly tilted, whith the result that the lattice constants a and b, which are identical in the low pressure rutile-type, tetragonal phase, are of different size in the $CaCl_2$-type orthorombic stishovite phase. This tilting has the same symmetry as the stishovite B_{1g} vibrational mode, which was identified as the pressure-induced soft mode that drives the transition [67]. By means of a Landau expansion Carpenter et al. [12] found a classical second-order character. The transition has almost no impact on the volume or the c/a ratio, which is shown in the paper by Carpenter.
However molecular dynamics models were not able to reproduce the tranistion pressure but only found the transition at pressures in the megabar region [42, 97, 56].
As a comparison between molecular dynamics and experiment the elastic constants can be analysed. Carpenter et al. [12] used a Landau expansion to generate expressions for the elastic constants of stishovite with pressure. Fig. 4.6 shows the relevant

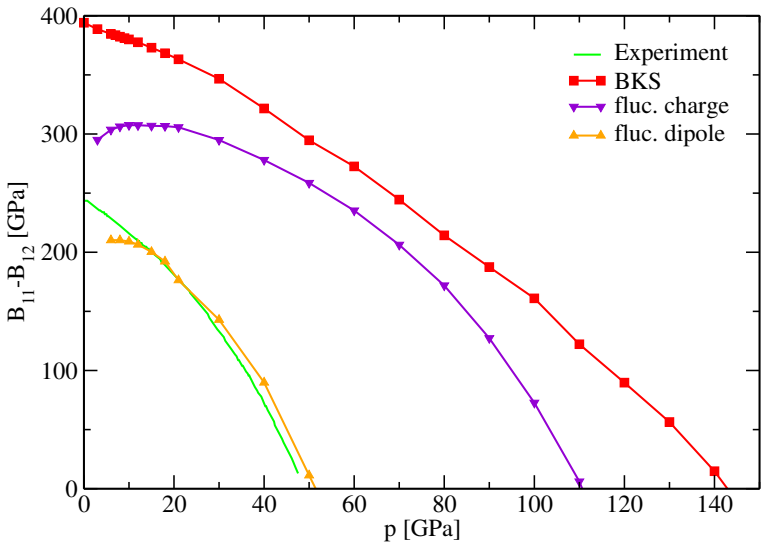


Figure 4.6.: Birch coefficients B_{11}-B_{12} as a function of pressure. Simulation data is compared to the result of a Landau expansion by Carpenter et al. [12]

shear modulus B_{11}-B_{12}, which softens on approach of the transition, as the B_{1g} mode directly contributes to it. The Birch coefficents B_{ij} have to be considered for a stability analysis rather than the elastic constants C_{ij}, when the system is observed under external pressure. In the case of the two coefficents needed here, they are related to each other as $B_{11} = C_{11} - p$ and $B_{12} = C_{12} + p$.

One can see that the BKS potential and the fluctuating dipole potential show a softening towards pressures of over 100 GPa, which is in agreement to the molecular dynamics simulations cited above, and far away of the 50 GPa measured experimentally. In contrast the fluctuating dipole potential predicts the transition in almost perfect agreement with the analysis by Carpenter at 50 GPa.
This prediction can be verified by a plot of the lattice constants a and b, which coincide in a tetragonal lattice and differ in an orthorombic lattice. In Fig. 4.7 the course of

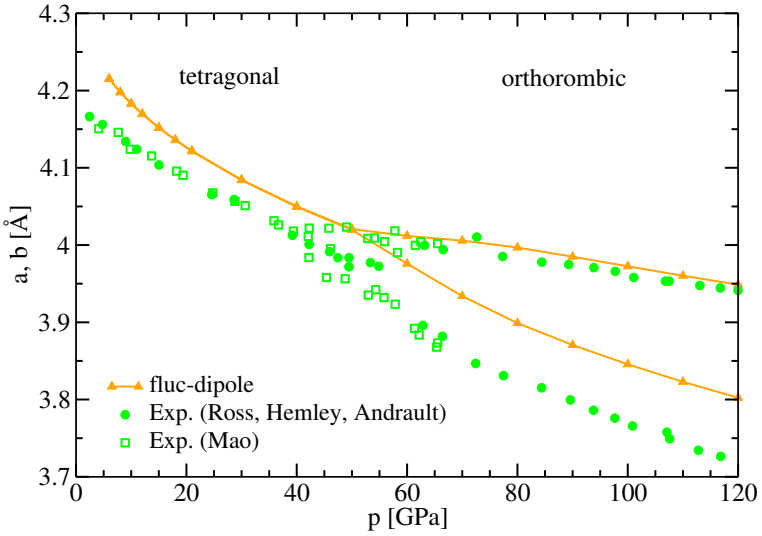


Figure 4.7.: Lattice constants a and b against pressure. Experimental data as in Fig. 4.4.

the lattice constants with pressure is compared to experimental data. The transition is clearly visible at a pressure of around 50 GPa in both the fluctuating-dipole simulation and experiment. The length of the lattice constant is overestimated constantly in the simulation, which was already obvious in the volume plotted in Fig. 4.4.
However the agreement with experiment in this phase transition is far better than in the other model potentials. This is rather surprising, as it was parametrized for tetrahedrally coordinated silicon only.

5. Electromechanical and Dielectric Properties

Dielectric and electromechanical properties of materials play an important role in many technological applications, as for instance piezoelectric resonators which are used as frequency generators in computers. Furthermore it is interesting to check how the results of the fluctuating charge and dipole potential compare to the standard two-body force field, as these new approaches claim to account for electronic properties and therefore should offer a better basis for the investigation of dielectric properties.
Due to these reasons it is desirable to have methods at hand that allow one to compute dielectric, piezoelectric and related tensors in atomistic simulations. Dielectric properties are more challenging to calculate than thermal and mechanical properties, because the definition of polarization in periodically repeated cells is less obvious than the formulation of thermo-mechanical variables. The reason is that an electrical dipole, and hence the polarization, lives on the surface of a sample. One of the consequences for computer simulations is that the value of the dipole depends upon where we chose the 'central image' of the periodically repeated cell to be. Conversely, internal energy, lattice parameters, strain, etc. are independent of the central image's position. As outlined further below, the ambiguity in defining the polarization becomes particularly delicate even in classical molecular dynamics simulations when one treats a system containing free charges at finite temperatures and constant stress, which implies fluctuating box shapes.
Despite the mentioned ambiguity in the definition of polarization in periodic cells, it has been shown that there is a bulk polarization that is intrinsic to the crystal and only depends on variations in charge density induced by atomic displacements. It is therefore independent of the surfaces and thus of the choice of the central image [63].
In the following chapter different methods are established for the calculation of dielectric and piezoelectric properties. These are in particular the dielectric tensor, defined as

$$\epsilon_{\alpha\beta} = \frac{1}{\epsilon_0}\frac{\partial P_\alpha}{\partial E_\beta}, \tag{5.1}$$

and the piezoelectric coefficients. The latter represent the mechanical reaction of the system to an applied field, and can therefore be defined either in terms of the strain or

the stress. In the following the focus will be on the piezoelectric strain coefficients:

$$d_{\alpha\beta,\gamma} = \frac{\partial u_{\alpha\beta}}{\partial E_\gamma}. \tag{5.2}$$

Another quantity that will be used in the following derivation is the piezoelectric stress coefficients at constant strain:

$$\gamma_{\alpha\beta,\gamma} = \frac{\partial \sigma_{\alpha\beta}}{\partial D_\gamma}. \tag{5.3}$$

Two different approaches for the calculation of these quantities will be employed. The first is the direct calculation of the slope of the polarization and the strain respectively against an electric field applied on the simulation box in different spatial directions. The second method is an evaluation of fluctuation relations within a usual simulation run in zero external field.

5.1. Theory and Methods

In order to develop the fluctuation relations, we will introduce different generalized vectors and matrices. In the following, a vector $\underline{v}$ denotes a column of numbers, while a transposed vector $\underline{v}^T$ represents a row of numbers. This means that the scalar product of two vectors u and v will be written as $\underline{u}^T\underline{v}$ and $\underline{u}\underline{v}^T$ is a matrix of rank two. Within Voigt notation, all matrices except for the piezoelectric tensors are square matrices. The piecoelectric coefficients are represented by a 6×3 matrix in Voigt notation. As before, spatial coordinates are written in capital letters to denote real space, while lowercase letters represent reduced space:

$$\underline{R}_i = \underline{\underline{h}}\,\underline{r}_i.$$

Additionally a vector $\underline{s}_i$ is introduced, which represents the reduced coordinates of the central periodic image of the particle i. When the particle moves over the image boundaries, the coordinate vector $\underline{r}_i$ behaves smoothly while we have to add or subtract unity to $\underline{s}_i$. This means that the components of $\underline{s}_i$ always lie within 0 and 1, whereas $\underline{r}_i$ can exceed these boundaries. This will prove to be a relevant property as the polarization must not depend on the boundary conditions.

5.1.1. Linear Response

Starting point is the isothermal partition function $Z_{h,D}(N, \beta)$ at fixed dielectric displacement $\underline{D}$ and constant box geometry as defined by a symmetric matrix $\underline{\underline{h}}$. Accord-

ing to the regular rules of statistical mechanics, $Z_{h,D}(N,\beta)$ reads

$$Z_{h,D}(N,\beta) = \int d\Gamma \delta(\underline{\underline{h}}(\Gamma) - \underline{\underline{h}}) \delta(\underline{D}(\Gamma) - \underline{D}) \exp\left(-\beta\Phi(\Gamma)\right). \tag{5.4}$$

Here $\int d\Gamma$ is an integral over phase space, the $\delta(\cdot)$'s are the delta functions singling out proper geometry and dieelectric displacement, and $\Phi(\Gamma)$ is the potential energy of the system as a function of phase space, this means as a function of $\underline{\underline{h}}$ and $\{\underline{s}\}$. From the partition function, the free energy $\mathcal{F}$ is defined via the equation

$$\mathcal{F}_{h,D}(N,\beta) = -k_B T \ln Z_{h,D}(N,\beta). \tag{5.5}$$

As outlined in more detail below, these equations allow one to connect phenomenogical materials parameters with ensemble averages over phase space. With the assumption of ergodicity the integration over phase space is carried out by averaging along the trajectory of the molecular dynamics simulation.
In the theory of elasticity it is helpful to express the partition function and free energy as a function of strain rather than $\underline{\underline{h}}$. Unlike the box geometry, the definition of the strain does not need the reference geometry $\underline{\underline{h}}_0$ of the system, which is typically chosen to reflect the expectation value of $\underline{\underline{h}}$ at given temperature and stress, as an additional information. As used before in the calculation of elastic constants, the Lagrangian strain with respect to $\underline{\underline{h}}_0$ can be written as

$$u_{\alpha\beta} = \frac{1}{2}\left\{ \left(\underline{\underline{h}}_0^{-1}\right)_{\alpha\gamma} h_{\gamma\delta}\ h_{\delta\epsilon} \left(\underline{\underline{h}}_0^{-1}\right)_{\epsilon\beta} - \delta_{\alpha\beta} \right\}, \tag{5.6}$$

where tensor notation was used instead of Voigt notation. In Eq. (5.6), $\delta_{\alpha\beta}$ denotes the Kronecker symbol. Since the relation between $\underline{\underline{h}}$ and $\underline{u}$ is well defined for a given reference $\underline{\underline{h}}_0$, it is possible to calculate the free energy as a function of $\tilde{u}$ in terms of a Taylor series expansion in $\tilde{u}$ away from a phase transition point, i.e.,

$$\mathcal{F}_{u,D} = \mathcal{F}_{h_0,D} + \frac{\partial \mathcal{F}_{h,D}}{\partial \underline{\underline{h}}} \frac{\partial \underline{\underline{h}}}{\partial \underline{u}} \underline{u} + \ldots \tag{5.7}$$

The new free energy depends on both, strains $\underline{u}$ and displacement $\underline{D}$. In the following, for convenience of calculations these variables will be grouped together into a generalized strain $\tilde{u} = (u_1, \cdots, u_6, D_1, \cdots, D_3)^t$, with $(u_1, \cdots, u_6)$ being the strain tensor in Voigt notation. The generalized stresses $\tilde{\sigma}$ that are the conjugate thermodynamic variables to $\tilde{u}$ can be calculated as $\tilde{\sigma} = (1/V)\,\partial\mathcal{F}(N,\beta)/\partial\tilde{u}$. Away from a phase transition, it is possible to expand $\mathcal{F}_{\tilde{u}}$ into a power series around a reference strain $\tilde{u}_0$ (which one can usually set to zero if $\tilde{u}_0$ at the reference geometry), thus

$$V_0\tilde{\sigma} = \frac{\partial \mathcal{F}_{h_0,D}(N,\beta)}{\partial \tilde{u}} + \frac{\partial^2 \mathcal{F}_{h,D}(N,\beta)}{\partial \tilde{u}^t\ \partial \tilde{u}} \delta\tilde{u} + \ldots, \tag{5.8}$$

where the derivatives are evaluated at $\tilde{u}_0$, the relative generalized strain is $\delta\tilde{u} = \tilde{u} - \tilde{u}_0$, and V_0 denotes the volume of the system at the reference geometry. The (expectation value of the) generalized stress at $\delta\tilde{u} = 0$ will be called $\tilde{\sigma}_0$ and $\delta\tilde{\sigma} = \tilde{\sigma} - \tilde{\sigma}_0$ denotes an (average) stress variation. One can then interpret the second-order derivative of $\mathcal{F}_{\tilde{u}}$ with respect to $\tilde{u}$ as generalized elastic constants $\underline{\underline{\tilde{C}}}$, that connect $\delta\tilde{u}$ and $\delta\tilde{\sigma}$ via $\delta\tilde{\sigma} = \underline{\underline{\tilde{C}}}\,\delta\tilde{u}$, or if we represent $\tilde{u}$ and $\tilde{\sigma}$ explicitly

$$\begin{pmatrix} \delta\underline{\sigma} \\ \delta\underline{E} \end{pmatrix} = \begin{bmatrix} \underline{\underline{C}}_{\underline{D}} & \underline{\underline{\gamma}}^t \\ \underline{\underline{\gamma}} & \left(\epsilon_0\, \underline{\underline{\epsilon}}_{\mathrm{r},\underline{u}}\right)^{-1} \end{bmatrix} \begin{pmatrix} \delta\underline{u} \\ \delta\underline{D} \end{pmatrix}. \tag{5.9}$$

Here, $\underline{\underline{C}}_{\underline{D}}$ denotes 'real' elastic constants at constant dielectric displacement, $\underline{\underline{\gamma}}_{\underline{u}}$ are the piezoelectrical stress coefficients at constant strain, and $\underline{\underline{\epsilon}}_{\mathrm{r},\underline{u}}$ is the (isothermal) dielectric tensor at constant strain. Eq. (5.9) is a linear equation that connects $\delta\tilde{u}$ with $\delta\tilde{\sigma}$. It is therefore possible to chose up to nine linearly independent thermodynamic variables, i.e., we can 'fix' the $\delta\tilde{\sigma}$ and 'measure' the response $\delta\tilde{u}$. This can be done by multiplying both sides of Eq. (5.9) with the inverse of $\underline{\underline{\tilde{C}}}$, which can be written as:

$$\begin{pmatrix} \delta\underline{u} \\ \delta\underline{D} \end{pmatrix} = \begin{bmatrix} \underline{\underline{C}}_{\underline{E}}^{-1} & \underline{\underline{d}}^t \\ \underline{\underline{d}} & \epsilon_0\, \underline{\underline{\epsilon}}_{\mathrm{r},\underline{\sigma}} \end{bmatrix} \begin{pmatrix} \delta\underline{\sigma} \\ \delta\underline{E} \end{pmatrix}. \tag{5.10}$$

Here $\underline{\underline{C}}_{\underline{E}}$ have to be interpreted as elastic constants at constant (external) electric field and $\underline{\underline{d}}$ are the piezoelectrical strain coefficients. Eq. (5.9) uses the more natural representation of the linear-response coefficients than Eq. (5.10), as stress and (external) electrical field are typically the experimentally controlled parameters, while strain and polarization and thus dielectric displacement are the measured responses.

Within linear-response theory, $\delta\tilde{u}$ and $\delta\tilde{\sigma}$ are conjugate to each other. In the constant-$\tilde{u}$ ensemble, we can therefore attribute an excess free energy $\Delta\mathcal{F}$ due to the fluctuations in $\tilde{u}$, which is given by

$$\Delta\mathcal{F} = \frac{V_0}{2}\tilde{u}^t\underline{\underline{\tilde{C}}}\,\tilde{u}. \tag{5.11}$$

Thermal fluctuations of $\tilde{u}$ at fixed $\tilde{\sigma}$ will thus be related to $\underline{\underline{\tilde{C}}}$, which for harmonic approximations, results in

$$\left\langle \begin{bmatrix} \delta\underline{u}\delta\underline{u}^t & \delta\underline{u}\delta\underline{D}^t \\ \delta\underline{D}\delta\underline{u}^t & \delta\underline{D}\delta\underline{D}^t \end{bmatrix} \right\rangle = \frac{k_B T}{V_0} \begin{bmatrix} \underline{\underline{C}}_{\underline{E}}^{-1} & \underline{\underline{d}}^t \\ \underline{\underline{d}} & \epsilon_0\, \underline{\underline{\epsilon}}_{\mathrm{r},\underline{\sigma}} \end{bmatrix}. \tag{5.12}$$

In this section, we have measured the free energy density by dividing the actual value of $\mathcal{F}$ by the volume V_0 of the reference strain. This convention, which is typically used

in the theory of elasticity. is also beneficial when evaluating Eq. (5.12): The fluctuation of the electric displacement can be rewritten as fluctuation of the polarization $\underline{P}$. Introducing the dielectric susceptibility, $\underline{\underline{\chi}} = \underline{\underline{\epsilon}}_r - \underline{\underline{1}}$, we can rewrite Eq. (5.10) as:

$$\begin{pmatrix} \underline{\delta u} \\ \underline{\delta P} \end{pmatrix} = \begin{bmatrix} \underline{\underline{C}}_{\underline{E}}^{-1} & \underline{\underline{d}}^t \\ \underline{\underline{d}} & \epsilon_0 \underline{\underline{\chi}}_{\underline{\sigma}} \end{bmatrix} \begin{pmatrix} \underline{\delta\sigma} \\ \underline{\delta E} \end{pmatrix}. \tag{5.13}$$

and conclude with the same procedure the flucuation relation

$$\left\langle \begin{bmatrix} \delta\underline{u}\delta\underline{u}^t & \delta\underline{u}\delta\underline{P}^t \\ \delta\underline{P}\delta\underline{u}^t & \delta\underline{P}\delta\underline{P}^t \end{bmatrix} \right\rangle = \frac{k_B T}{V_0} \begin{bmatrix} \underline{\underline{C}}_{\underline{E}}^{-1} & \underline{\underline{d}}^t \\ \underline{\underline{d}} & \epsilon_0 \underline{\underline{\chi}}_{\underline{\sigma}} \end{bmatrix}. \tag{5.14}$$

Relations such as Eqs. (5.8) and (5.14) can be exploited in atomistic calculations to determine the susceptibilities of interest. However, it will first be necessary to state the proper estimators for the polarization fluctuation.

5.1.2. Ambiguity of the dipole and its fluctuation

To understand the difficulties connected with the definition of bulk polarization and its fluctuations in an infinitely replicated system it is helpful to consider Fig. 5.1. Part (a) of Fig. 5.1 shows that the definition of the dipole moment μ in the cell (which leads to the polarization P through division by the volume of the simulation box), depends upon the choice of the boundary of the central cell. By displacing the boundary, none of the interatomic distances is altered, yet the dipole changes its value. As long as the shape (here simply the length) of the cell is maintained, this ambiguity causes no complications for the response functions, because the only relevant quantity is the difference between polarizations. A polarization change is independent of the choice for the boundary - provided the 'true' trajectory of the atoms is measured, i.e., when a charged particle moves across a boundary the scaled coordinate for the calculation of the dipole is not increased or decreased by unity. Thus evaluating fluctuations related to dipoles (from which the constant-strain dielectric and piezoelectric constants could be calculated), do not depend on the choice for the position of the central image.
When the cell fluctuates, even a polarization difference will depend upon where we chose the central image to be. This is illustrated in part (b) of Fig. 5.1, where in one case, the reduced coordinates of a charged particle is constrained to zero (bottom) and in the other case it is set constant to one half (top). Both times, we assume the presence of an opposite background charge of constant density ensuring charge neutrality. In the upper part of Fig. 5.1 (b), the fluctuation of the dipole vanishes exactly if we keep the reduced positions of the charge Q fixed in the center of mass. Therefore, the fluctuation of the dipole will be zero in the upper half. In the bottom part, we would

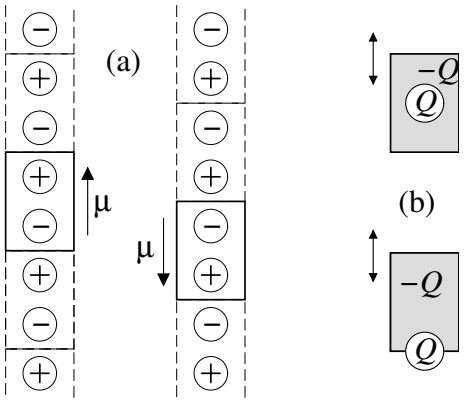


Figure 5.1.: Illustration of the ambiguity for (a) the definition of a dipole μ and for (b) the fluctuation of the dipole. The boxes framed by a solid line present the position of the central image, while boxes framed with a broken line are periodic images. The lower left corner of the central image is defined to the origin of the coordinate system. In part (b), a positive background neutralizes the point charge Q and the arrows with the two arrow heads represent fluctuation of the box geometry.

attribute a dipole to the cell, which would fluctuate with changing box size. Thus the dipole fluctuations are affected by the choice of the location for the central image. The above mentioned difficulties do not arise if one deals with neutral molecules whose constituents have charges that add up to zero. In that case it is easy to remove the ambiguity due to surface effects or periodic-boundary conditions. One would only have to suppress any minimum image convention within the molecules, i.e., for the calculation of the dipole, the $\underline{r}_i$ have been to be defined such that two covalently bonded atoms are separated by the proper distance. In principle, it would be possible to define similar molecules in an ideal crystalline network such as quartz, i.e., by evaluating the dipole over entities that consist of a central silicon atom and its four oxygen neighbors, where the charges of the O atoms would only count half, in order to avoid double counting. However, such 'tricks' cannot be generally applied, for instance, if any type of disorder or even impurities are present in the system, or if a non-rigid charge model is employed.

5.1.3. Fluctuation estimators for dipoles

The problem of defining bulk dipoles can be overcome by considering the small wave length limit of the dipole's spatial Fourier transform $\tilde{\mu}$. If the box cell is fluctuating, it is appropriate to work with scaled reciprocal vectors $\underline{k} = 2\pi(m_x, m_y, m_z)$ where the m_α are integer numbers that are related to the true reciprocal vectors through $\underline{K} = \underline{k}\,\underline{\underline{h}}^{-1}$. Since the box is fluctuating, it is more meaningful to work with reduced coordinates in

both real and reciprocal space. Let us first consider the Fourier transform of the charge distribution $\rho(\underline{\tilde{k}})$

$$\tilde{\rho}(\underline{k}) = \sum_i Q_i e^{i\underline{k}\,\underline{s}_i}. \tag{5.15}$$

It remains unchanged if we add or subtract unity from a particular $\underline{s}_i$ or if the box geometry is rescaled. Yet, the formal derivative $\tilde{\rho}(\underline{k})$ with respect to $\underline{k}$ (which one cannot do in practice as the k_α's are discrete) does not show the same invariance. The derivative of $\tilde{\rho}(\underline{k})$ evaluated at $\underline{k} = 0$ corresponds to the contribution of the reduced dipole due to the point charges, which is defined as

$$\underline{\mu}_{\mathrm{red}} = \sum_i Q_i \underline{r}_i + \underline{\underline{h}}^{-1} \underline{\mu}_i, \tag{5.16}$$

If, however, one defines a reference configuration, which can be an initial configuration at time $t = 0$ that does not have to be equilibrated, then the *difference* between a reduced dipole $\mu_{\mathrm{red}}(t)$ evaluated at time t and the reference dipole $\mu_{\mathrm{red}}(0)$ is invariant against the transformations discussed in Fig. 5.1, as long as the transformation is performed on the configuration of interest and the reference configuration. For the final evaluation of the dipole, the value $\mu_{\mathrm{red}}(0)$ has to be multiplied with the expectation value of the box shape. Thus our estimator becomes

$$\mu_{\mathrm{estimator}} = \langle \underline{\underline{h}} \rangle \; \mu_{\mathrm{red}} \tag{5.17}$$

and consequently that for the polarization $\underline{P}_{\mathrm{estimator}} = \mu_{\mathrm{estimator}} / \langle V \rangle$. These estimators do not enable one to calculate absolute dipoles or polarizations but only relative quantities, i.e., those to be used in Eq. (5.14) for the calculation of fluctuation relations. If absolute values for the polarization were needed, the dipole moment of the initial configuration should vanish. This could be accomplished by chosing a configuration with vanishing dipole moment due to an underlying symmetry.

5.1.4. Direct estimators with noise reduction

An alternative method to evaluate piezoelectric coefficients is to apply an external electric field $\underline{E}$ to the system. The appropriate force that acts on the reduced coordinates related to a point charge Q_i would read $\underline{\underline{h}}^{-1} \underline{E} Q_i$. It is yet not meaningful to simply add a term $-\sum_i Q_i \underline{E}^t \underline{R}_i$ to the Hamiltonian. Such a perturbation would require to have a force act onto the shape of the simulation cell and this force would depend on whether a particle is counted within the central image or within a periodic image. We thus suppress the force from the external field onto the $\underline{\underline{h}}$ matrix, also because it would be absent in a system without charges and dipoles.

If the shape of the system is known for $\underline{E} = 0$ and the initial configuration is equilibrated, then one may switch on the field adiabatically and monitor the box shape variation from which the strain can be calculated. This procedure will give correct results, in particular if one averages over an appropriate number of independent, initial configurations. However, this algorithm may produce large stochastic scatter in particular at elevated temperatures.

The noise can be reduced and the necessity of having to know the average structure can be omitted if one performs a reference simulation in which the electric field remains switched off. The reference simulation should be based on the same initial condition and an identical initialization of the thermostat. Any instantaneous configuration at finite field can be compared directly to the instantaneous configuration at zero field. Within a few molecular dynamics steps, it is then possible to obtain quite reasonable estimates for the piezoelectrical and other dielectric coefficients, even if the system is rather anharmonic and contains slow modes, as is the case in quartz close to the $\alpha - \beta$ transition. Fig. 5.2 confirms this expectation. In Fig. 5.2, we increased the

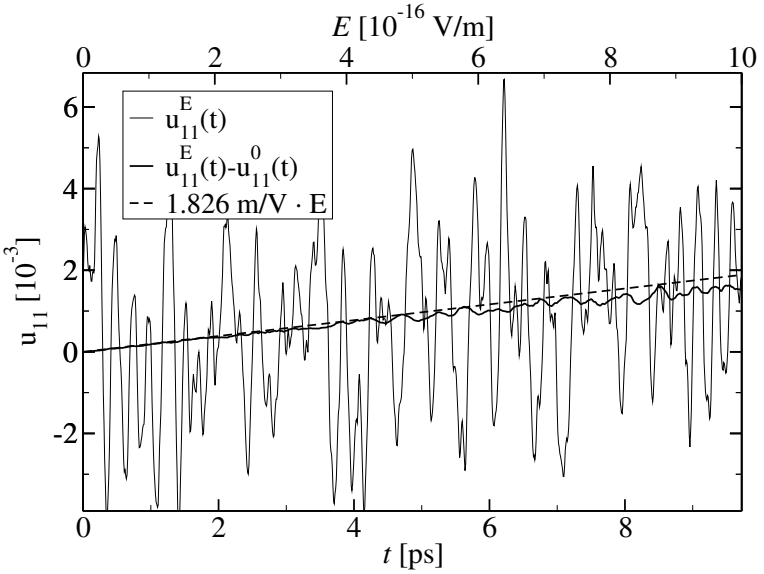


Figure 5.2.: Response of the strain u_{11} as a function of time (below) or electrical field (above) for an $N = 1080$ particle system (α-quartz) at temperature $T = 300$ K. The long dashed line is the result for the piezo-electrical coefficient based on evaluating the fluctuation relation in Eq. (5.14).

external field slowly with time and also chose box inertia sufficiently small to have the cell shape adapt quickly to the new field. The box geometry was thermostatted with a Langevin thermostat to decrease the (stochastic) correlation time. Excellent estimates for the piezoelectrical coefficients can be obtained already after one hundred MD steps by evaluating the slope $\partial u_{11}/\partial E_x$. In this way, four simulations have to be run to

determine all generalized elastic constants; three in which the electric field is parallel to one of the three coordinate axis and one in which the electric field remains zero. As said above it is necessary that in all four runs the same initial conditions (one large sample in thermal equilibrium or an average over some small equilibrated samples) must be used as an input into the simulation and that all runs require the same random number sequence.
For the various silica polymorphs investigated here, we obtain a statistical accuracy of better than 5% for γ if we first equilibrate the material for about 10,000 MD steps to produce an equilibrated start configuration. Note that even though quartz is crystalline, it does takes relatively long to equilibrate, because it has an unusually large phonon density of states at small frequencies. The start configuration is then used for four independent runs, namely one without electric field plus three runs with electric field in one of the three spatial dimensions. Each of these runs is about 2,500 MD steps long and produces a graph similar to that shown in Fig. 5.2. This length of the simulation run is by far long enough to obtain a small error in the linear regression, while in a longer run the abberations in the four trajectories get too large. Also non linear effects would begin to dominate, at least for the set up and the slope of electric field against time used above. Thus, the required numerical effort to obtain the data is relatively small.

Using Nosé Hoover thermostats or related (less sophisticated) rescaling methods do probably not lead to similarly reliable results. In particular for large systems and/or small temperatures, the motion of the simulation cell will be quite harmonic and thus the crucial exchange of (kinetic) energy between the simulation cell and the internal degrees of freedom will be inefficient. This inefficiency will prevent the cell from quickly finding its new preferred shape or from fluctuating around it.

5.2. Temperature Dependence of d_{11} in Quartz

The temperature dependence of piezoelectrical constants is of technological relevance because one is often interested in having pressure sensors and pressure transducers at varying temperatures. One of the few disadvantages of α-quartz is the relatively low transition temperature to β-quartz. Due to the increase in symmetry from the trigonal holoaxial α-quartz structure to the tetragonal holoaxial β-quartz structure, the piezoelectricity decreases such that the d_{11} component becomes symmetry-forbidden. This means that quartz cannot be used as an effective piezoelectrical material at high temperature. Fig. 5.3 shows the temperature dependence of the piezoelectrical strain coefficient d_{11}. One can see that both potentials underestimate the value of d_{11}, the fluctuating dipole potential underestimates its value in the low-temperature phase by

more than 50%. The discrepancy between experiment and the fluctuating dipole

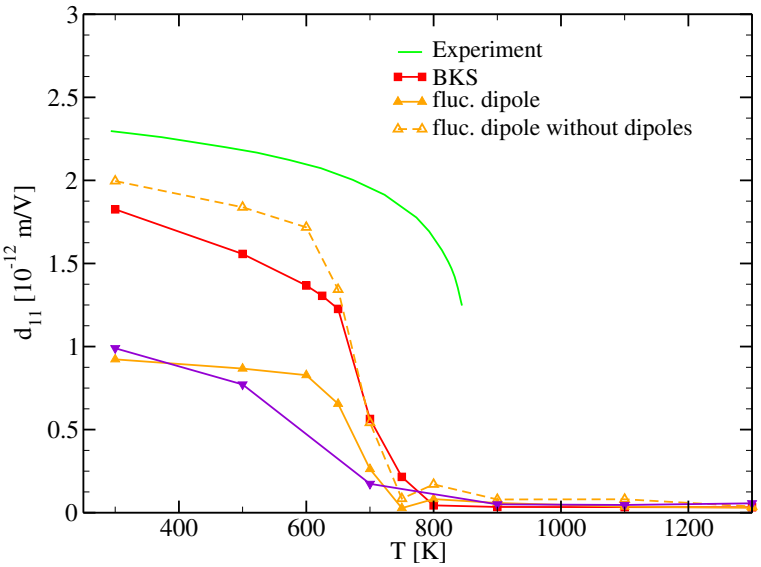


Figure 5.3.: Comparison of the temperature dependence of the piezoelectrical strain coefficient d_{11} between experiment and simulation. Different potentials were used, the BKS and the fluctuating dipole potential. In the fluc. potential, two set of runs were performed, one in which the electric field was coupled to the dipoles, and one in which this coupling was suppressed for the reasons outlined in the text. Experimental data was taken from Cook and Weissler [17]

potential is suprising having in mind the otherwise good performance of this potential. As already mentioned in chapter 2.4.1, it is possible that Tangney and Scandolo parametrized quantum-chemical effects into the dipoles, so that their dielectric properties should possibly not be treated as such. Re-running the simulations without coupling the dipoles to the electric field, so that only the bare charges coupled to $\underline{E}$, showed that the discrepancy between the fluctuating dipole potential and the experiment is significantly reduced and quantitative agreement is almost achieved.
Of course in this comparison one has to take into account that the transition temperature is about 100 K underestimated by this potential.
The high temperature behaviour of quartz was recently studied by Haines et al. [37], who observed a loss of piezoelectric properties in α-quartz still below the transition temperature. It had been known for many decades that the piezoelectric coefficient d_{11} of quartz decreases gradually above room temperature [10, 17, 6] as shown in Fig. 5.3. Haines et al. measured the quality factor of piezoelectric resonators and found that it decreases significantly already at 550 K and reaches values characteristic of a poor resonator above 750 K, which is well below the transition temperature. The time-averaged structure of α quartz does not provide an explanation for this behaviour, as the order parameter decreases only gradually in this temperature range. Therefore

Haines et al. suspected that this deterioration of the piezoelectric response arises from local instantaneous disorder. This can be characterized by the tilt angle Φ. In Fig. 5.4, the distribution of the tilt angles in the crystal is shown for different temperatures.

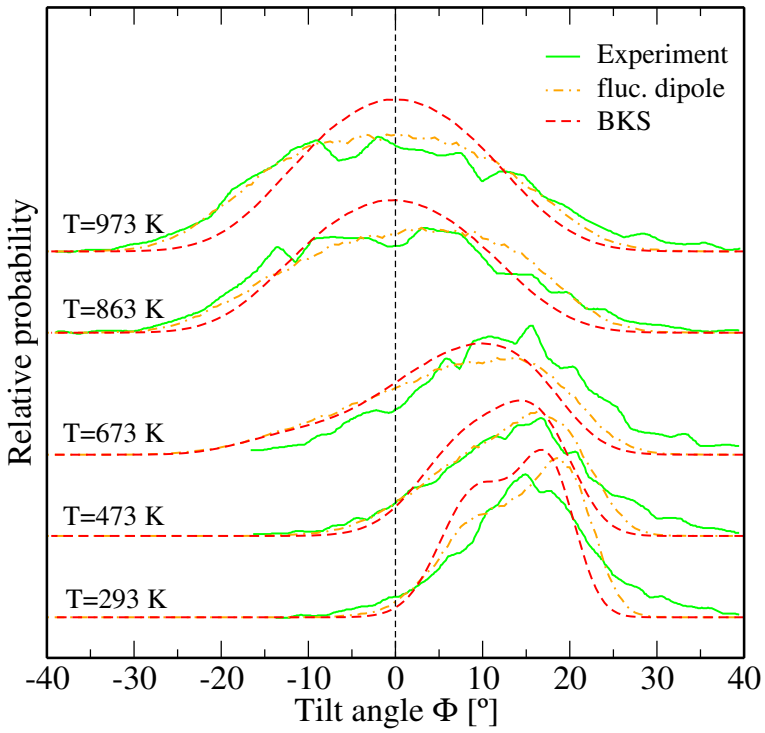


Figure 5.4.: Distribution functions of the tilt angle Φ in quartz as a function of temperature. Note that at temperatures of 863 K and 973 K, the crystal is in the β-quartz form. Experimental data taken from Haines et al. [37].

The comparison of the simulation data with experiment shows that this explanation also holds in computer simulations. While the average tilt angle hardly changes for the three temperatures in the α-quartz stability range, the distribution of tilt angles is significantly broadened. The shift towards smaller tilt angles in the simulation as compared to experiment at $T = 673$ K arises from the lower transition temperature in both potentials. This increasing disorder leads to the early decline of the quality factor, while quantities that can be correlated to the static average value, like the piezoelectric coupling constant, remain constant towards higher temperatures.
In the α-quartz phase, where this phenomenon occurs, both potentials are in good agreement with experiment, which is reflected in Fig. 5.3 in the quite similar decrease of d_{11} with temperature. However it is interesting to note that the tilt angle distribution in β-quartz is obviously more accurate with the fluctuating dipole potential, which means that the disorder in the β-quartz phase is modelled closer to reality.

5.3. Pressure Dependence of d_{11} in α-Quartz

In Fig. 5.5, we show the pressure dependence of quartz with the same approaches as in Fig. 5.3. While probably the most accurate data is obtained with the fluctuating dipole approach by suppressing the coupling between $\underline{\mu}$ and $\underline{E}$, the "full" fluc.-dipole potential and the BKS potential are included for comparison. The fluc.-dipole potential with

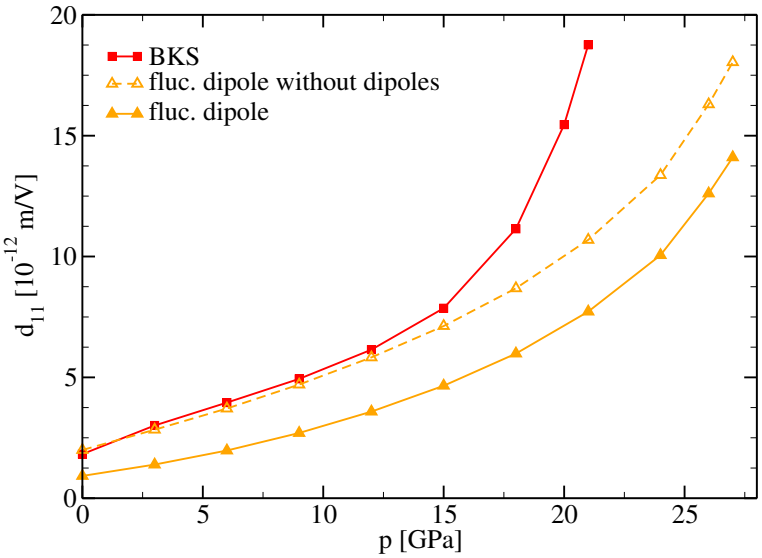


Figure 5.5.: Comparison of the pressure dependence of the piezoelectrical strain coefficient d_{11} between experiment and simulation. Different potentials were used, the BKS and the fluctuating dipole potential. In the fluc. potential, two set of runs were performed, one in which the electric field was coupled to the dipoles, and one in which this couplig was suppressed for the reasons outlined in the text.

suppressed $\underline{\mu}$-$\underline{E}$ coupling shows a surprisingly strong similarity to the BKS potential up to 12 GPa. Above this pressure, the results for the piezoelectrical coefficients are starting to differ. Keeping in mind that the transition pressure for the $\alpha \rightarrow II$ transition is 21 GPa in BKS compared to 28 GPa in TS, it is obvious that the results start to differ on approach to the transition pressure. However the absolute values compared at the respective transition temperature are again rather comparable. The increase in the piezoelectrical activity originates in the softening of internal modes in α-quartz, in which the oxygen atoms are displaced with respect to the oppositely charged silicon atoms.

5.4. Bond-Angle Dependence of the Dipole Moment

In chapter 5.2 above the observation was made, that the predictions of piezoelectric properties approve, when the dipoles are not fully treated as dipoles in the calculation, but when the dipoles do not couple to the electric field and do not contribute to the polarization of the material. It was already suspected before in the discussion of the tetraeder distortion in chapter 2.4.1 that the value of the dipole moments calculated in the potential by Tangney and Scandolo modelled effects caused by the quantum chemical bonds. It was shown that the distortion of the SiO_4 tetraehedron, which is modelled extremely accurately by this potential, was related by Grimm and Dorner [35] to the π-bonds between Si and O atoms. Additionally the symmetry of the π orbitals resembles closely to the symmetry of the dipoles.
We want to investigate this possibility by having a closer look on the relation between the dipole moment and its nearest surrounding. In Fig. 5.6 the distribution of the dipole moments with respect to the Si-O-Si bond angle around the respective oxygen atom is

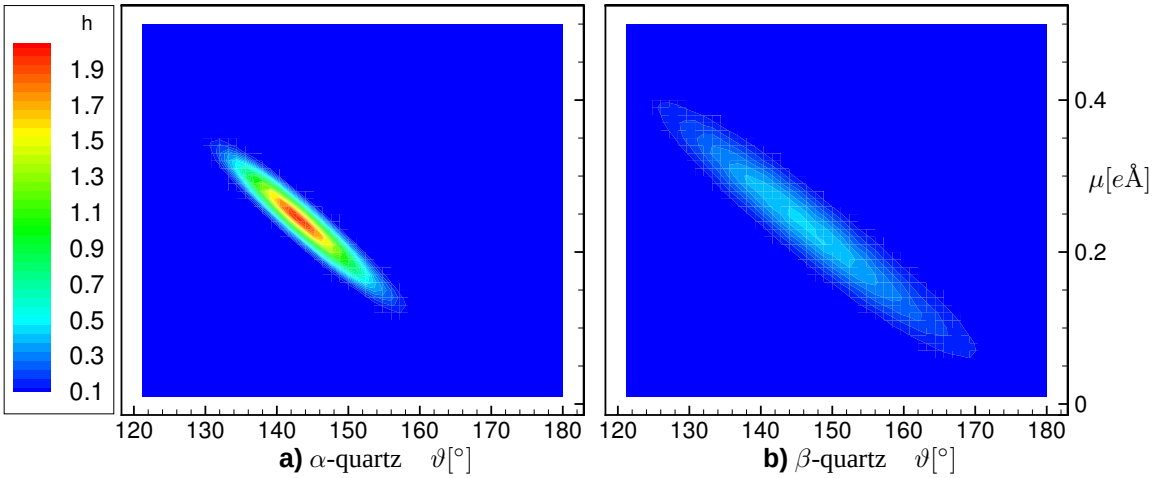


Figure 5.6.: Distribution of the absolute value of the dipole moment μ against the Si-O-Si bond angle ϑ on the adjacent oxygen atom. The probability density h is plotted in units of probability per degree and eÅ.

shown. One can see that the dipoles depend almost linearly of the oxygen bond angle ϑ. There is some statistical spread, which increases with higher temperature. But apparently the absolute value of the dipoles depends mainly on the bond angle. This means that they depend largely on local effects of the bonding to the closest neighbors. To measure the orientation of the dipole moments, we use the labels introduced in Fig. 5.7. The orientation angle φ is the angle between the direction of the dipole moment and the direction given by the bisecting line of the Si-O-Si angle ϑ.

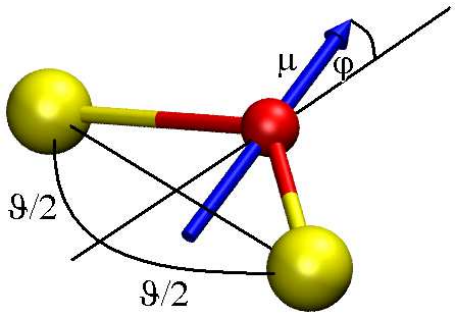


Figure 5.7.: Sketch of bond angle ϑ, dipole orientation angle φ and absolute value of dipole μ, as used in Fig. 5.6 and 5.6.

One can see in the distribution of the orientation angle in Fig. 5.8 that the dipole moment points most probable in the direction of the bisecting line, which is within the layer defined by the oxygen and the two neighboring silicon atoms and represents the most symmetric orientation at this point. In the β-quartz phase the orientation might deviate more than 10° with a certain probability. But in these cases the dipoles have a small absolute value, as shown in Fig. 5.8, and therefore they have only little influence on the forces.

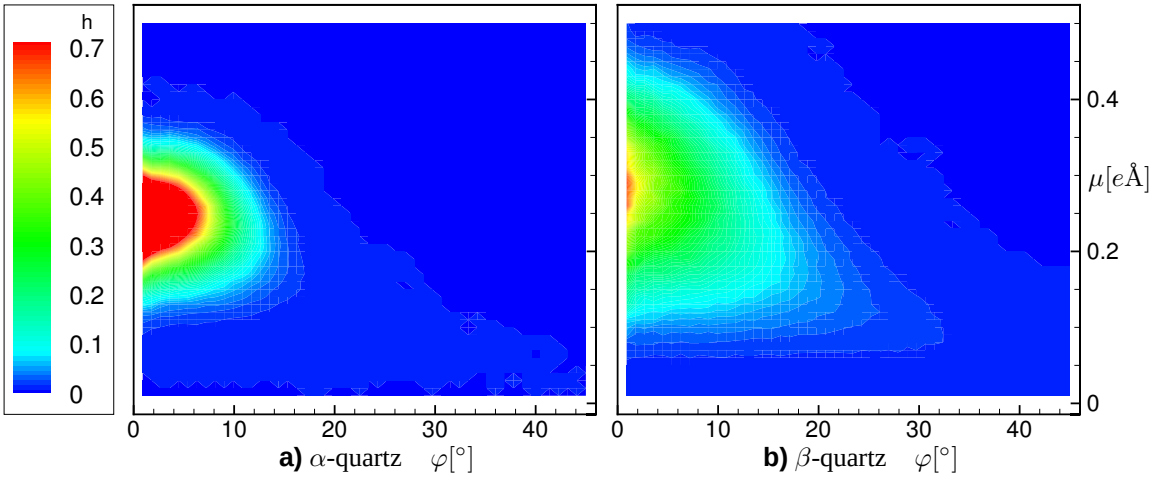


Figure 5.8.: Distribution of the absolute value of the dipole moment μ against the deviation angle φ. The deviation angle is measured as explained in the text and shown in Fig. 5.7. The probability density h is plotted in units of probability per degree and eÅ.

As the orientation of the four dipoles sitting on the oxygen atoms around one silicon atom are pairwise opposed (see Fig. 2.20), one can conclude that the contribution of distant dipoles on the force acting on an atom dissappear faster than the usual charge-dipole interaction.
Summing up these observations it is likely that there was some contribution of the quantum chemistry to the forces, stresses and energy parametrized into the dipoles.

6. Conclusion and Outlook

In the present work, different model potentials were used for the simulation of silica polymorphs, in order to study the influence of different enhancements of the standard two-body potential. The potentials used for comparison were the frequently used BKS two body pair potential [101], a model that had the atomic charges dynamically calculated from the spatial configuration [20] and a third that employs fixed charges but allows for inducible dipole moments on the oxygen atoms [90]. In the case of the fluctuating charge potential some modifications had to be implemented as the original parametrization was flawed.

For the testing of the different potentials we concentrated on crystalline phases, as in disordered structures effects are more likely to average out and therefore in crystals more independent susceptibilities can be observed. An example are the piezoelectric constants or the elastic constant C_{14}, which are forbidden for symmetry reasons in the glass phase, but are present in certain crystalline phases. The observables taken into account are predominantly of static nature and sometimes dynamic.

We analysed several cases in which the BKS potential showed quantitatively or qualitatively incorrect results. So the α-β quartz transition is located at a transition temperature that is 100 K smaller than in experiment, and the density of both the α and the β quartz phase are underestimated by about 4%, but the global picture with respect to the volume change, the weak first-order nature of the transition and the hysteresis effects is certainly well reproduced. However one important detail, the ratio of the lattice constants c and a, shows a qualitative difference to the experimental behaviour at the transition temperature. This discrepancy turned out to be originated in a distortion of the SiO_4 tetrahedra that deviates strongly in direction and size from the true distortion. Another problematic quantity is the phonon density of states, which only reproduces basic aspects of the lattice dynamics, namely the intra-tedrahedral modes, but even in these modes the frequency differs from ab-initio data. The low frequency range cannot be brought into agreement with ab-initio data at all.
In all of these difficulties, the allowance for fluctuating charges could not be of significant help. While the aberration in the transition temperature and the density was even stronger in the same direction, the predictions for the c/a ratio and the phonon density of states basically shows the same shortcomings as seen in the BKS potential.

The introduction of additional inducible dipole moments on the oxygen atoms was a far more successful supplement to the two-body forces. Even though the transition temperature deviates slightly more from experiment than in the BKS potential, this remained the only drawback in the results of the fluctuating dipole potential. For the estimation of the density the agreement with experiment is very satisfactory, and most important the qualitative failures of the BKS potentials in the vibrational density of states and the c/a ratio do not occur with the fluctuating dipoles. It could be shown that the distortion of the tetrahedra, which causes the behaviour of the c/a ratio, is in perfect agreement with experiment. The resemblance of the dipole orientation to the symmetry of the π-bonds, which were hold responsible for the tetrahedra deformation in a work by Grimm and Dorner [35], gives rise to the suspicion that some of the quantum chemistry in the atomic bonding was incorporated into the dipole interaction during the parametrization of this potential.
The elastic constants were in generally good agreement for the BKS potential, and were equally good reproduced by the fluctuating dipole potential. Only the fluctuating charge potential showed strong deviations. The good results of BKS in this respect are not surprising given that the BKS parameters were adjusted to the experimental elastic constants of α-quartz.

To test the new potentials at extreme conditions, the post-quartz phases under high pressure were studied. Even though there is a rich variety of different high pressure polymorphs and different suggestions for possible high-pressure transition path ways from experiment and theory, it is remarkable that all three potentials showed a transition in a pressure range of 21 to 27 GPa to the same crystalline structure, which is supposedly the structure of the still unclear quartz II phase. However the δ-quartz phase seen by the BKS potential on decompression to ambient pressures is not reproduced by any of the two new potentials. Therefore the formation of this phase seems to be an artifact of the BKS potential. In contrast, the two new potentials do see the quartz II phase revert to α-quartz, while at intermediate pressures a phase emerges that is very similar to the quartz II phase at higher pressures.

As the new potential approaches incorporate electrostatic properties, namely fluctuating charges and fluctuating dipole moments, the investigation of dielectric properties in chapter 5 should highlight the question if these are real electrostatic variables and thus improve the prediction for the dielectric properties of the system. As there was no appropriate method for the calculation of piezoelectric properties available in literature, we developed a method to measure these by fluctuation relations within the simulation run. In order to achieve this the definition of the polarisation in an infinitely replicated system had to be clarified, as the definition is ambigious due to surface effects and to box fluctuations, and the fluctuation relation was established by linear response theory.

To test this method the direct impact of an external field was measured, where the noise was reduced by subtracting the observables measured in a zero field run with identical start configuration and random number generator.
We found that the BKS potential gives a quite reasonable estimate for the piezoelectric constants of α-quartz against temperature. The fluctuating charge potential was again slightly more away from experiment than BKS. For the fluctuating dipole potential the observation was made that the result depends on the question whether the dipoles are taken seriously, this means if the dipoles are coupled to the electric field and contribute to the polarization in the material, or if they are ignored with respect to electrostatic quantities. The results clearly improved in the later case, again giving rise to the suspicion, that the dipoles that are induced on the oxygen atoms are not pure electric dipole moments. This question was further investigated by an analysis of the relation between the dipole moment and its direct surrounding. It turned out that the strength of the dipole depends linearly on the adjacent oxygen bond angle and is most likely in the direction of the bisecting line of this angle.

Putting these observations together, we can compare the potentials in the following way:

	BKS	Fl-Q	Fl-μ
DOS in α-quartz	-	0	+
Elastic constants in α and β quartz	+	0	+
c/a anomaly at the α-β quartz transition	-	-/0	+
Equation of state in α and β-quarty	0	-	+
Stability of cristobalite and tridymite	-	0	+
Piezoelectric coefficients	0	0	(+)
Pressure induced transition	+	+	+
Stishovite c/a and elasticity	0	-	0

The results shown above make it clear that the fluctuating charge approach is not of use for the simulation of silica. Although it is very likely that the effective charges, which represent the degree of ionicity, change with pressure and for different polymorphs, there is no evidence to suggest that this happens dynamically at a given pressure or temperature in the real system. In the simulations it was shown that keeping the charges fixed on their average values does not influence the results of the simulations. The fluctuation in the charges is quite small, and one can even see that the original authors update the charges on a time scale which is far beyond the time scales of ionic motion. In an equilibrated system the charges can therefore not react on the present spatial configuration. The overall conclusion is that the dynamic adaption of the charges within an equilibrated system does not influence the ionic motion notably. However this approach could be probably more useful for systems where the ionicity

changes dynamically, as for instance at a surface.

In contrast, the effects contained in the fluctuating dipoles have a significant influence on the system. Their contribution is sufficiently large that the α-β transition is suppressed when the dipoles are constrained to their average values in the respective phases. In summary, the results achieved with this potential turn out to be in very close agreement with experiment with the exception of six-coordinated silicon at small pressures. It is the only potential that accurately models the tetrahedral distortion and the lattice dynamics in quartz.
Possibly the quantum effects that are apparently parametrized in terms of dipoles could be expressed in a newly parametrized three body term and thus reduce the computational expense.

All properties of four-coordinated silica could be described very precisely with the fluctuating dipole potential, even though the potential was not explicitly tested nor optimized for these experimental results, but only parametrized to (microscopic) ab-initio data. The parametrization was done in the predominantly tetrahedrally coordinated liquid phase and therefore slightly biased in favor of a tetrahedral crystal structure, which explains the weakness in the six coordinated stishovite phase at low pressure. However it is even more surprising that it is the only model that predicts the experimentally measured transition pressure in stishovite. The overall success clearly approves the functional form of this model potential as well as the parametrization procedure, and gives rise to the hope that this procedure can find similarily good potentials for other materials as well.

A. Calculations

A.1. Calculations for Fluc-Q Potential

A.1.1. Direct Solution of Charge Equilibration Equations

Equation (1.24) and its following explanations only take into account a nonperiodic system. Regarding a system with periodic boundary conditions, the Coulomb interaction has to be calculated by making use of the Ewald summation 1.1.5. With summation over all periodic images, eq. (1.23) reads

$$\chi_i = \chi_i^0 + J_{ii}^0 Q_i + \sum_j{}' J_{ij} Q_j \tag{A.1}$$

The notation $\sum_j{}'$ meaning the sum over all periodic pictures and $i \neq j$ in the original picture can be rewritten as

$$\begin{aligned} \sum_j{}' J_{ij} Q_j &= \sum_j{}' \left(\frac{1}{R_{ij}} - \left(\frac{1}{R_{ij}} - J_{ij} \right) \right) Q_j & \text{(A.2)} \\ &= \sum_j{}' \frac{1}{R_{ij}} Q_j - \sum_{j \neq i} \left(\frac{1}{R_{ij}} - J_{ij} \right) Q_j, & \text{(A.3)} \end{aligned}$$

because $J_{ij} \rightarrow 1/R_{ij}$ for $R_{ij} \rightarrow 0$ and therefore the difference is short range. The remaining pure Coulomb term can be evaluated using standard Ewald summation:

$$\sum_j{}' \frac{1}{R_{ij}} Q_j = \sum_{j \neq i} \frac{1}{R_{ij}} \text{erfc}(\sqrt{\alpha} R_{ij}) Q_j - \frac{2\alpha}{\sqrt{\pi}} Q_i + \sum_j C_{ij} Q_j \tag{A.4}$$

$$\text{with} \qquad C_{ij} = \frac{4\pi}{V} \sum_{\underline{k} \neq 0} \frac{\exp(-\frac{k^2}{4\alpha})}{k^2} \exp(i \underline{k} \underline{R_i}) \exp(-i \underline{k} \underline{R_j}) \tag{A.5}$$

Inserted back to eq. (A.1) this gives

$$\chi_i = \chi_i^0 + A_{ij} Q_i + \sum_{j \neq i} B_{ij} Q_j \tag{A.6}$$

$$\text{with} \qquad A_i = J^0_{ii} - \frac{2\alpha}{\sqrt{\pi}} + C_{ii} \tag{A.7}$$

$$\text{and} \qquad B_{ij} = \frac{1}{R_{ij}}\text{erfc}(\sqrt{\alpha}R_{ij}) - \left(\frac{1}{R_{ij}} - J_{ij}\right) + C_{ij} \tag{A.8}$$

Now the condition $\chi_i = \chi_1$ reads

$$-A_1Q_1 + A_iQ_i + \sum_{j\neq i} B_{ij}Q_j - \sum_{j\neq 1} B_{1j}Q_j = -(\chi^0_i - \chi^0_1) \tag{A.9}$$

Together with the charge neutrality constraint this leads to the set of equations

$$\underline{\underline{C}} \cdot \underline{Q} = -\underline{D}, \tag{A.10}$$

where

$$D_i = \chi^0_i - \chi^0_1 \tag{A.11}$$

and

$$\underline{\underline{C}} = \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ -A_1 + B_{21} & A_2 - B_{12} & B_{23} - B_{13} & \cdots & B_{2N} - B_{1N} \\ -A_1 + B_{31} & B_{32} - B_{12} & A_3 - B_{13} & \cdots & B_{3N} - B_{1N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -A_1 + B_{N1} & B_{N2} - B_{12} & B_{N3} - B_{13} & \cdots & A_N - B_{1N} \end{pmatrix} \tag{A.12}$$

with A_i and B_{ij} as above.

A.1.2. Two Body Slater Integral

The integration over the overlap of two slater orbitals with two different centers can be sketched as in Figure A.1. The value of the integral depends on the distance r between the centers a and b. The integral to be evaluated (see eq. 1.32) looks in this nomenclature like

$$J(r) = \iint \phi^2_i(r_{i1}) \frac{1}{r_{12}} \phi^2_j(r_{j2}) d\underline{r}_{i1} d\underline{r}_{j2} \tag{A.13}$$

$$= \int \phi^2_i(r_{i1}) U_j(\underline{r}_{i1}) d\underline{r}_{i1} \tag{A.14}$$

with

$$U_j(\underline{r}_{i1}) = \int \frac{1}{r_{12}} \phi^2_j(r_{j2}) d\underline{r}_{j2} \tag{A.15}$$

$$= A^2_j \int \frac{1}{r_{12}} r^{2n_j-2}_{j2} \, e^{-2\zeta_j r_{j2}} d\underline{r}_{j2} \tag{A.16}$$

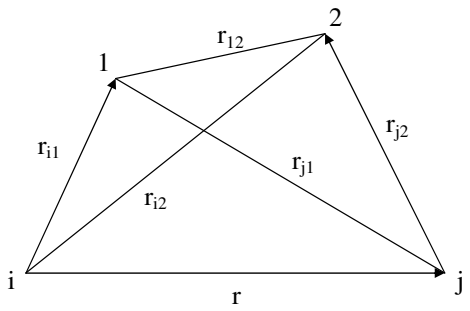


Figure A.1.: The i and j points are the centers of the orbitals, the r_{i1} and r_{j2} are the integration variables.

The A_j is the normalization factor to ensure $\int \phi^2 dV = 1$ and can be calculated as

$$A = \sqrt{\frac{1}{4\pi}\frac{(2\zeta)^{2n+1}}{(2n)!}} \tag{A.17}$$

Solving at first the $U_j(\underline{r}_{i1})$ integral, one has to expand $1/r_{12}$ in spherical harmonics:

$$\frac{1}{r_{12}} = \sum_{\lambda=0}^{\infty} \frac{4\pi}{2\lambda+1}\frac{r_<}{r_>^{\lambda}+1} \sum_{\kappa=-\lambda}^{+\lambda} Y_\lambda^\kappa(\theta_1,\varphi_1)^*\, Y_\lambda^\kappa(\theta_2,\varphi_2) \tag{A.18}$$

Here $r_<$ is the smaller and $r_>$ is the bigger of r_{j1} and r_{j2}. In the angular integration only $\lambda = 0$ delivers a non-vanishing contribution. So the radial integration is left over:

$$\begin{aligned} U_j(\underline{r}_{i1}) &= A_j^2 4\pi \int_0^\infty \frac{1}{r_>}\, r_{j2}^{2n_j-2}\, e^{-2\zeta_j r_{j2}} dr_{j2} && \text{(A.19)} \\ &= A_j^2 4\pi \left\{ \frac{1}{r_{j1}} \int_0^{r_{j1}} r_{j2}^{2n_j}\, e^{-2\zeta_j r_{j2}} dr_{j2} + \int_{r_{j1}}^\infty r_{j2}^{2n_j-1}\, e^{-2\zeta_j r_{j2}} dr_{j2} \right\} && \text{(A.20)} \\ &= A_j^2 4\pi r_{j1}^{2n_j} \left\{ \int_0^1 y^{2n_j}\, e^{-2xy} dy + \int_1^\infty y^{2n_j-1}\, e^{-2xy} dy \right\} && \text{(A.21)} \end{aligned}$$

In the last line, the substitution $y = r_{j2}/r_{j1}$ and $x = r_{j1}/\zeta_j$ was applied. After evaluation of the two integrals, substitution of A_j from eq. (A.17) and further simplification, one obtains

$$U_j(\underline{r}_{i1}) = \frac{1}{r_{j1}} - e^{-2\zeta_j r_{j1}} \sum_{\nu=1}^{2n_j} \frac{\nu (2\zeta_j)^{2n_j-\nu}}{(2n_j-\nu)!2n_j} r_{j1}^{2n_j-\nu-1} \tag{A.22}$$

Substitution back into eq. (A.13) yields

$$J(r) = A_i^2 \left\{ \int r_{i1}^{2n_1-2} \frac{1}{r_{j1}} e^{-2\zeta_1 r_{i1}} d\underline{r}_{i1} - \sum_{\nu=1}^{2n_j} \frac{\nu (2\zeta_j)^{2n_j-\nu}}{(2n_j-\nu)!\, 2n_j} \int r_{i1}^{2n_1-2} r_{j1}^{2n_j-\nu-1} e^{-2\zeta_1 r_{i1}} e^{-2\zeta_j r_{j1}} d\underline{r}_{i1} \right\}. \tag{A.23}$$

Due to their symmetry these two integrals can be evaluated best in elliptical coordinates (ξ, η, φ). The transformation is $r_{i1} = (r/2)(\xi + \eta)$, $r_{j1} = (r/2)(\xi - \eta)$ and $d\underline{r}_{i1} = (r/2)^3(\xi^2 - \eta^2) d\xi d\eta d\varphi$. After some simplifications, the final result for the overlap integral now reads

$$\begin{aligned} J(r) = \frac{(2\zeta_i)^{2n_i+1}}{2\,(2n_i)!} \Bigg\{ & \left(\frac{r}{2}\right)^{2n_i} \sum_{\nu=0}^{2n_i-1} \binom{2n_i-1}{\nu} A_{2n_i-1-\nu}(\zeta_i r) B_\nu(\zeta_i r) \\ & - \sum_{\nu=1}^{2n_j} \frac{\nu (2\zeta_j)^{2n_j-\nu}}{(2n_j-\nu)!\, 2n_j} \left(\frac{r}{2}\right)^{2n_i+2n_j-\nu} \sum_{k=0}^{2n_i-1} \sum_{l=0}^{2n_j-\nu} \binom{2n_i-1}{k} \binom{2n_j-\nu}{l} \\ & \times (-1)^l A_{2(n_i+n_j)-k-\nu-l-1}(r(\zeta_i+\zeta_j)) B_{k+l}(r(\zeta_i-\zeta_j)) \Bigg\} \end{aligned} \tag{A.24}$$

with $$A_k(x) = \int_1^\infty \xi^k e^{-x\xi} d\xi, \qquad \text{and} \qquad B_k(x) = \int_{-1}^{+1} \eta^k e^{-x\eta} d\eta \tag{A.25}$$

The auxiliary integral $A_k(x)$ can be evaluated by

$$A_k(x) = e^{-x} \frac{k!}{x^{k+1}} \sum_{\nu=0}^{k} \frac{x^\nu}{\nu!}, \tag{A.26}$$

whereas a similar representation of $B_k(x)$ turned out to be numerically unstable. Therefore $B_k(x)$ is evaluated by the following formula:

$$B_k(x) = \sum_{\nu=0}^{\infty} \frac{1 - (-1)^{k+\nu+1}}{\nu!(k+\nu+1)} (-x)^\nu \tag{A.27}$$

B. Programs

B.1. Molecular Dynamics Code

This is the main simulation code. Input parameters is handed over in a file called pimd.inp, the start configuration in a file `conf.sta`. If no start configuration is present, this file has to exist anyway with only the number 0 as content.
The program consists of the file `pimd.f`, and the file `pimd.com.f` with declarations of common variables. The force routine for the fluctuating dipole potential is not printed but can be obtained from Paul Tangney. The interface for this routine with the conversion of units is within the subroutine scandolo_potential.
The following routines from "Numerical Recipes in Fortran" [74] need to be included in the code: ludcmp, lubksb, erfcc, four1. All variables needed by this force routine are in atomic units.
The units used in the main program are as follows:

$$[l] = 1\text{Å} \quad [E] = 1k_BK \quad [t] = 1\hbar/(k_BK)$$
$$[m] = 1m_p \quad [Q] = 1e \quad [p] = 1k_BK/\text{Å}^3$$

The output files generated have the following content:

`fort.60`	Different observables with time, see code for columns.
`fort.61`	Order parameter with time
`pimd.g_r`	Pair correlations functions
`pimd.bon`	$g(r)$ for Si-O at nearest neighbor distance, bond angle distributions
`fort.62`	Average, minimum and maximum charge with time
`conf.end`	End configuration
`conf.xmo`	End configuration in xyz format
`confavg.xmo`	Averaged configuration in xyz format
`conf.pol`	End dipole configuration
`confavg.pol`	Averaged dipole configuration
`pimd.out`	Averaged observables and input configuration

pimd.f

```
      program pimd
      implicit none
      include "pimd.com.f"
      double precision avg_charge(n_type), max_charge(n_type),
     &     min_charge(n_type)
      integer i_test
      double precision qeqrec2(n_part,n_part)
      common /qeqrectest/ qeqrec2
      double precision pmix
      common /pmisch/ pmix
      double precision erstes_v, zweites_v
      double precision qeqrec21(n_part)

      double precision v_qabhaengig_etest
      common /extlagetest/ v_qabhaengig_etest

      integer count, c1, c2, crate, cmax
      real performance

      n_run = 1
      do i_run = 1, n_run
      write(*,*) "i_run",i_run
      call para_init
      if (i_run.eq.1) call conf_init
      call obse_init

      call check_skin
      if (f_skin.eq.1) then
         call binning
      end if

      if ((f_chargeadjust.eq.3).or.(f_chargeadjust.eq.4)) then
         call matrixinversion
      end if

      r_real_avg = 0.

      do i_time = 1, n_relax + n_obser

       r_time = r_time + dt

!---   propagate coordinates and its derivatives one time unit
       call predict
       if (f_chargeadjust.eq.3) call extdlagrange_predictor

!---   check whether neighbor lists need to be updated
       if (f_potential.ne.1) then
        call check_skin
        if (f_skin.eq.1) then
         call binning
        end if
       end if

!---   transform predicted coordinates into real space
       call trans_real

!---   ensure harmonic boundary conditions
       if (f_potential.ne.1) call pb_condit

!---   calculate interaction
       if (f_potential.eq.1) then
        call harm_osc
       else if (f_potential.eq.2) then
        call lennard_jones
       else if (f_potential.eq.3) then
        call bks_potential
       else if (f_potential.eq.4) then
        call bks_potential
       else if (f_potential.eq.5) then
        call goddard_potential
       else if (f_potential.eq.6) then
        call goddard_potential
       else if (f_potential.eq.7) then
        call goddard_potential
       else if (f_potential.eq.8) then
        call vashishta_potential
       else if (f_potential.eq.9) then
        call scandolo_potential
       end if

       if (f_e_applied.gt.0) call applied_field

!---   calculate new charges
       if (f_chargeadjust.eq.1) then
!        leave charges unchanged
       else if (f_chargeadjust.eq.3) then
!        do nothing (included in goddard_potential)
       else if (f_chargeadjust.eq.4) then
          if (i_time.le.n_relax) then
             if (mod(i_time,25).eq.0) call matrixinversion
          else
             if (mod(i_time,100).eq.0) call matrixinversion
          end if
       end if


       if (i_time.gt.n_relax) then
          call observation
       end if

!---   transform interactions into reciprocal space
       call trans_recip
       if (f_potential.ne.1) call box_chain

!---   add random forces
       call thermostat

!---   correct trajectories
       call correct
       if (f_chargeadjust.eq.3) call extdlagrange_corrector

       r_real_avg = r_real_avg + r_real

       if (mod(i_time,n_safe).eq.0) call conf_save
      end do

      call obser_out

      end do

      end program
```

```
! ! ! ! ! ! ! ! ! ! ! !

      subroutine para_init
      implicit none
      include "pimd.com.f"

      integer n_order_max
      parameter (n_order_max=5)
      double precision pred_coef(0:n_order_max-1,0:n_order_max),
     &     corr_coef(0:n_order_max)
      double precision slatercutoff, slaterijr
      double precision slaterint, slaterint_d

      integer jseed
      common /cjseed/ jseed

      double precision r_1,r_2,r_6,exp_pot,v_coulomb,pot_loc
      double precision d_dumm1, d_dumm2

      integer msidummy1, msidummy2
      logical exist
      double precision potcheck(n_type,n_type)

! *************** f_potential 1
      if (f_potential.eq.1) then

      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
       jseed = iseed

       dt = 0.035d0
       tk = 0.275
       mass(0,1) = 1.d0
       mass(0,2) = mass(0,1)

       if (n_order.eq.2) then
!----   check energy conservation
        friction(1) = 0.
        fric_box = 0.
        dt = dt/2
       else
        friction(1) = 0.01 / dt
        fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
      else     ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"
       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box
      end if   ! i_dummy
      close(10)
      else     ! i_run
       tk = tk - 0.25
      end if   ! i_run

! *************** f_potential 2
      else if (f_potential.eq.2) then


      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
       jseed = iseed

       for_sca_mfa_cor = 0.

!----    Lennard Jones potential; LJ units
!       mass(0,1) = 100.d0
!       epsil(1,1) = 1.d0
!       sigma(1,1) = 1.d0
!----    LJ potentials; my units \hbar=1,k_b=1,A=1,e=1
!----    Argon
        mass(0,1) = 40*mass_amu
        epsil(1,1) = 1.67e-21*joule
        sigma(1,1) = 3.405
!----    Helium: Aziz potential more accurate
!       mass(0,1) = 4*mass_amu
!       epsil(1,1) = 10.22
!       sigma(1,1) = 2.556
!----    Neon
!       mass(0,1) = 18*mass_amu
!       epsil(1,1) = 0.494e-21*joule
!       sigma(1,1) = 2.750d0
!----    symmetrize masses and Lennard Jones coefficients
        mass(0,2) = mass(0,1)
        epsil(2,2) = epsil(1,1)
        epsil(1,2) = sqrt(epsil(1,1)*epsil(2,2))
        epsil(2,1) = epsil(1,2)
        sigma(2,2) = sigma(1,1)
        sigma(1,2) = (sigma(1,1)+sigma(2,2))/2
        sigma(2,1) = sigma(1,2)
!----    chose appropriate cutoff and skin
```

```
         do i_type = 1,n_type
         do j_type = 1,n_type
          r_cutoff(i_type,j_type) = 1.7*sigma(i_type,j_type)
!         r_cutoff(i_type,j_type) = 3.00*sigma(i_type,j_type)
          r_skin(i_type,j_type) = 0.18*r_cutoff(i_type,j_type)
         end do
         end do
!----    chose appropriate start parameters
         tk = 0.4*epsil(1,1)
         dt = 0.0125*sqrt(sigma(1,1)**2*mass(0,1)/epsil(1,1))
         f_pressure = 1
         press = .001*epsil(1,1)/sigma(1,1)**3

       if (n_order.eq.2) then
!----   check energy conservation
        friction(1) = 0.
        fric_box = 0.
        dt = dt/2
       else
        friction(1) = 0.01 / dt
        fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12

       mass_box = mass_box / 2

      else  ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"

       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box

        read(10,200) char_dummy
        read(10,201) press
        read(10,210) f_pressure

        read(10,200) char_dummy

         do i_type = 1,n_type
          read(10,202) (epsil(i_type,j_type),j_type=1,n_type)
         end do
         read(10,200) char_dummy
         do i_type = 1,n_type
          read(10,202) (sigma(i_type,j_type),j_type=1,n_type)
         end do

        read(10,200) char_dummy
```

```
         do i_type = 1,n_type
          read(10,202) (r_cutoff(i_type,j_type),j_type=1,n_type)
         end do

        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
        end do

       end if
       close(10)
       else
        tk = tk - 0.25
       end if

       if (i_run.eq.1) then
!---   absorb factor 4 into epsilon and initialize sigma_2
       do i_type = 1,n_type
       do j_type = 1,n_type
        sigma_2(i_type,j_type) = sigma(i_type,j_type)**2
        epsil(i_type,j_type) = 4*epsil(i_type,j_type)
       end do
       end do
      end if


       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
        r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
     &  r_skin(i_type,j_type)
        r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
       end do
       end do

!---   define shifts in energy

        do i_type = 1,n_type
        do j_type = 1,n_type
         r_dummy = (sigma_2(j_type,i_type)/r_cutoff_2(j_type,i_type))**3
         e_shift(j_type,i_type) = r_dummy*(r_dummy-1.)
        end do
        end do

! ************** f_potential 3
      else if (f_potential.eq.3) then

      if (f_chargeadjust.ne.1) then
         stop "BKS potential does not work with variable charge"
      end if

      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
```

```
      jseed = iseed

       if (f_chargeadjust.ne.1) then
         qeqelectroneg(1) = 4.168*11604.5049  ! eV -> kB K
         qeqelectroneg(2) = 8.741*11604.5049
         qeqhardness(1) = 6.974*11604.5049
         qeqhardness(2) = 13.364*11604.5049
       end if

        n_relax = 30
        n_obser = 30
        n_safe  = 30
!----   silicium
        mass(0,1) = 28.1*mass_amu
!----   oxygen
        mass(0,2) = 16.0*mass_amu
        epsil(1,1) = 0.*ev
        epsil(2,2) = 1388.773*ev
        epsil(1,2) = 18003.7572*ev
        epsil(2,1) = epsil(1,2)
        sigma(1,1) = 0.
        sigma(2,2) = 2.76
        sigma(1,2) = 4.87318
        sigma(2,1) = sigma(1,2)
        c_vandwaals(1,1) = 0.
        c_vandwaals(2,2) = 175.0*ev
        c_vandwaals(1,2) = 133.5381*ev
        c_vandwaals(2,1) = c_vandwaals(1,2)
        alpha = 0.3  * (1002./n_part)**(1./6)
        alpha = 0.3
        v_cut = 0.01*elec_prefac*(1.2*2.4)/1.6/sqrt(1.*n_part)
        f_opt_rec = 0
        do i_type = 1,n_type
        do j_type = 1,n_type
         r_skin(i_type,j_type) = 0.45
        end do
        end do
        tk = 3600./n_trot
! ----   time step = 1 fs
        dt = 1.3091e-4
        f_pressure = 1
        press = .0

      if (n_order.eq.2) then
!----   check energy conservation
       friction(1) = 0.
       fric_box = 0.
       dt = dt/2
      else
       friction(1) = 0.01 / dt
       fric_box = 0.01 / dt
      end if
      friction(2) = friction(1)
!---   define inertia for box deformation
      mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
       mass_box = mass_box / 8
     else    ! i_dummy
      if (i_dummy.ne.f_potential)
    &     stop "f_potential and pimd.inp mismatch"
      read(10,210) i_dummy, char_dummy
      if (i_dummy.ne.f_chargeadjust)
    &     stop "f_chargeadjust and pimd.inp mismatch"

      read(10,210) n_relax
      read(10,210) n_obser
      read(10,210) n_safe
      read(10,210) iseed
      read(10,200) char_dummy
      read(10,201) dt
      read(10,201) tk
      read(10,200) char_dummy
      read(10,202) (mass(0,i_type),i_type=1,n_type)
      read(10,202) (friction(i_type),i_type=1,n_type)
      read(10,202) mass_box, fric_box

       read(10,200) char_dummy
       read(10,201) press
       read(10,210) f_pressure
       read(10,210) f_opt_rec
       read(10,200) char_dummy
       read(10,200) char_dummy

       if (f_chargeadjust.ne.1) then
         read(10,202) (qeqelectroneg(i_type),i_type=1,n_type)
         read(10,202) (qeqhardness(i_type),i_type=1,n_type)
       end if
       read(10,200) char_dummy

        do i_type = 1,n_type
         read(10,202) (epsil(i_type,j_type),j_type=1,n_type)
        end do
        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (sigma(i_type,j_type),j_type=1,n_type)
        end do

         read(10,200) char_dummy
         do i_type = 1,n_type
          read(10,202) (c_vandwaals(i_type,j_type),j_type=1,n_type)
         end do


       read(10,200) char_dummy

        read(10,202) alpha, v_cut

       read(10,200) char_dummy
       do i_type = 1,n_type
        read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
       end do

     end if ! i_dummy
     close(10)
     else  !i_run
      tk = tk - 0.25
     end if  !i_run

      alpha_prefac = 2*alpha/sqrt(pi)
      r_cutoff(1,1) = 3./alpha
      r_cutoff(2,2) = 3./alpha
      r_cutoff(2,1) = 3./alpha
      r_cutoff(1,2) = r_cutoff(2,1)
!      if (r_cutoff(1,1).lt.r_cut_short)
!     & stop "cut off too small for short range"
```

```
      q_cut_2 = (6.*alpha)**2
!---  any peak |f(k|^2 in order 1/sqrt(N) gets cut off
      do i_type = 1,n_type
      do j_type = 1,n_type
       sigma_2(i_type,j_type) = sigma(i_type,j_type)**2
      end do
      end do
      r_cutoff_max = 0.0
      do i_type = 1,n_type
      do j_type = 1,n_type
       r_cutoff_max = max(r_cutoff_max,r_cutoff(i_type,j_type))
      end do
      end do


      do i_type = 1,n_type
      do j_type = 1,n_type
       r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
       r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
    &  r_skin(i_type,j_type)
       r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
      end do
      end do

!---  define shifts in energy

       do i_type = 1,n_type
       do j_type = 1,n_type
!----   electrostatic
       e_shift(i_type,j_type) = erfcc(alpha*r_cutoff(i_type,j_type))*
    &  elec_prefac/r_cutoff(i_type,j_type)
!----   short range
       e_shift_short(i_type,j_type) = epsil(i_type,j_type)*
    &  exp(-sigma(i_type,j_type)*r_cut_short)-
    &  c_vandwaals(i_type,j_type)/r_cut_short**6
      end do
      end do
!----  tabulate potential v(r) and dv/dr
      do i_type = 1,n_type
      do j_type = 1,n_type
      do i_potential = 0,n_potential
       r_1 = i_potential*r_range(i_type,j_type)/n_potential
       r_2 = r_1**2
!------ cutoff at 5.5 \AA shortrange interaction (SiO_2)
       if(r_1.le.r_cut_short) then
        exp_pot = epsil(i_type,j_type)*exp(-sigma(i_type,j_type)*r_1)
       else
        exp_pot = 0.0
       end if
       if(r_2.lt.2.0) then
        if(i_type.eq.j_type) then
         r_2 = 2.0
         r_1 = sqrt2
        else if(r_2.lt.1.21) then
         r_2 = 1.21
         r_1 = 1.1
        end if
       end if
!------ cutoff at shortrange interaction (SiO_2)
       if(r_1.le.r_cut_short) then
        r_6 = c_vandwaals(i_type,j_type)*(1./r_2)**3
       else
        r_6 = 0.
       end if
       v_coulomb = erfcc(alpha*r_1)*elec_prefac/r_1
       pot_loc = exp_pot - r_6
       if(r_1.le.r_cut_short)
    &  pot_loc = pot_loc - e_shift_short(i_type,j_type)
!----  r_dummy = {\partial V \over \partial r} / r
       r_dummy = -sigma(i_type,j_type)*exp_pot/r_1 + 6*r_6/r_2
       r_pot_1(i_type,j_type,i_potential) = pot_loc
       r_pot_2(i_type,j_type,i_potential) = -e_shift(i_type,j_type)
    &                                       +v_coulomb
       r_pot_d_1(i_type,j_type,i_potential) = r_dummy
       r_pot_d_2(i_type,j_type,i_potential) = -(1./r_2) * (v_coulomb+
    &      elec_prefac*alpha_prefac*exp(-(alpha*r_1)**2))
      end do
      end do
      end do
      do i_potential = -n_potential,n_potential
       r_dummy = i_potential*2*pi/n_potential
       c_dummy = r_dummy *(0.,1.)
       tab_exp_ima(i_potential) = exp(c_dummy)
      end do

! ************** f_potential 4
     else if (f_potential.eq.4) then
     if (i_run.eq.1) then
     open(10, file="pimd.inp", status="old")
     read(10,210) i_dummy, char_dummy

     if (i_dummy.eq.0) then
!----  take default parameters
      n_relax = 50
      n_obser = 50
      n_safe  = 100
      iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
      jseed = iseed

       if (f_chargeadjust.ne.1) then
         qeqelectroneg(1) = 4.168*11604.5049  ! eV -> kB K
         qeqelectroneg(2) = 8.741*11604.5049
         qeqhardness(1) = 6.974*11604.5049
         qeqhardness(2) = 13.364*11604.5049
       end if

       n_relax = 50
       n_obser = 50
       n_safe = 100
       mass(0,1) = 4*mass_amu
       mass(0,2) = mass(0,1)
       r_cutoff(1,1) = 10.
       r_skin(1,1) = 1.1
       tk = 4.5
       dt = 0.5e-2
       f_pressure = 1
! ---   crit. isobar of He 3 ?
       press = .7243e-2
! ---   crit. isobar of Helium 4
       press = .164750e-1
! ---   initialization of fo_sca...cor in aziz_local

     if (n_order.eq.2) then
!----  check energy conservation
```

```
        friction(1) = 0.
        fric_box = 0.
        dt = dt/2
       else
        friction(1) = 0.01 / dt
        fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
       mass_box = mass_box / 5

      else ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"

       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box

        read(10,200) char_dummy
        read(10,201) press
        read(10,210) f_pressure

        read(10,200) char_dummy

        if (f_chargeadjust.ne.1) then
          read(10,202) (qeqelectroneg(i_type),i_type=1,n_type)
          read(10,202) (qeqhardness(i_type),i_type=1,n_type)
        end if
        read(10,200) char_dummy

        read(10,200) char_dummy

         do i_type = 1,n_type
          read(10,202) (r_cutoff(i_type,j_type),j_type=1,n_type)
         end do

        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
        end do

      end if !i_dummy
      close(10)
      else ! i_run
       tk = tk - 0.25
      end if ! i_run

       do i_type = 1,n_type
       do j_type = 1,n_type
```

```
        r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
        r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
     &  r_skin(i_type,j_type)
        r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
       end do
       end do

!---   define shifts in energy

        i_type = 1
        r_1 = r_cutoff(1,1)
        call aziz_local(r_1,d_dumm1)
        e_shift_short(1,1) = d_dumm1
        do i_potential = 5, n_potential
         r_1 = i_potential * r_range(1,1) / n_potential
         call aziz_local(r_1,d_dumm1)
         r_pot(1,1,i_potential) = d_dumm1 - e_shift_short(1,1)
         r_1 = (i_potential-1) * r_range(1,1) / n_potential
         r_2 = r_1**2
         call aziz_local(r_1,d_dumm1)
         r_1 = (i_potential+1) * r_range(1,1) / n_potential
         r_2 = r_1**2
         call aziz_local(r_1,d_dumm2)
         r_pot_d(1,1,i_potential) = (d_dumm2-d_dumm1)
     &   / (r_range(1,1) / n_potential)**2 / (2*i_potential)
       end do


! ************** f_potential 5 + 6 + 7

      else if ((f_potential.ge.5).and.(f_potential.le.7)) then

      if ((f_potential.eq.5).and.(f_chargeadjust.eq.3)) then
         stop "MS-Q potential does not work with Ext Lagrange"
      end if

      if ((f_potential.eq.6).and.(f_chargeadjust.eq.4)) then
         stop "Ms-fluc Q potential does not work with exact
     &        diagonalisation"
      end if

      constant_charge(1) = 1.3D0
      constant_charge(2) = -0.65D0


      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
       jseed = iseed

         n_relax = 30
         n_obser = 30
         n_safe  = 30
!----    silicium
         mass(0,1) = 28.1*mass_amu
```

```
!----    oxygen
         mass(0,2) = 16.0*mass_amu

        if (f_chargeadjust.ne.1) then
          qeqelectroneg(1) = 4.168*11604.5049  ! eV -> kB K
          qeqelectroneg(2) = 8.741*11604.5049
          qeqhardness(1) = 6.974*11604.5049
          qeqhardness(2) = 13.364*11604.5049
        end if

         d0(1,1) = 0.2956
         d0(2,2) = 0.5363
         d0(1,2) = 45.997 !26.3765
         d0(2,1) = d0(1,2)
         msr0(1,1) = 3.4103
         msr0(2,2) = 3.7835
         msr0(1,2) = 1.6148  !1.83779
         msr0(2,1) = msr0(1,2)
         gamma(1,1) = 11.7139
         gamma(2,2) = 10.4112
         gamma(1,2) = 8.8022  !8.1
         gamma(2,1) = gamma(1,2)

         msn(1)=3
         msn(2)=2
         mszeta(1,1)=0.148810e1
         mszeta(1,2)=0.148810e1
         mszeta(2,1)=0.186846e1
         mszeta(2,2)=0.186846e1

         alpha = 0.3  * (1002./n_part)**(1./6)
         alpha = 0.4
         v_cut = 0.01*elec_prefac*(1.2*2.4)/1.6/sqrt(1.*n_part)
         v_cut = 0.
         f_opt_rec = 0
         do i_type = 1,n_type
         do j_type = 1,n_type
          r_skin(i_type,j_type) = 0.45
         end do
         end do
         tk = 3600./n_trot
!----    time step = 1 fs
         dt = 1.3091e-4
         f_pressure = 1
         press = .0

         if (f_chargeadjust.eq.3) then
           qeqmass(1) = 0.04
           do i_type=2, n_type
             qeqmass(i_type)=
     &         qeqmass(1) * sqrt(qeqhardness(i_type)/qeqhardness(1))
           end do
           qeqfriction = 0.1 / dt
         end if


       if (n_order.eq.2) then
!----   check energy conservation
         friction(1) = 0.
         fric_box = 0.
         dt = dt/2
       else
         friction(1) = 0.01 / dt
         fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
        mass_box = mass_box / 8
      else    ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"

       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box

        read(10,200) char_dummy
        read(10,201) press
        read(10,210) f_pressure
        read(10,210) f_opt_rec
        read(10,200) char_dummy

        if (f_chargeadjust.ne.1) then
          read(10,202) (qeqelectroneg(i_type),i_type=1,n_type)
          read(10,202) (qeqhardness(i_type),i_type=1,n_type)
        end if
        read(10,200) char_dummy

        if (f_chargeadjust.eq.3) then
          read(10,202) (qeqmass(i_type),i_type=1,n_type)
          read(10,201) qeqfriction
        end if
        read(10,200) char_dummy

         do i_type = 1,n_type
          read(10,202) (d0(i_type,j_type),j_type=1,n_type)
         end do
         read(10,200) char_dummy
         do i_type = 1,n_type
          read(10,202) (msr0(i_type,j_type),j_type=1,n_type)
         end do

          read(10,200) char_dummy
          do i_type = 1,n_type
           read(10,202) (gamma(i_type,j_type),j_type=1,n_type)
          end do

          read(10,200) char_dummy
          do i_type = 1,n_type
           read(10,202) (mszeta(i_type,j_type),j_type=1,n_type)
          end do
```

```
        read(10,200) char_dummy

        read(10,220) (msn(i_type),i_type=1,n_type)

        read(10,200) char_dummy

         read(10,202) alpha, v_cut

        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
        end do

      end if ! i_dummy
      close(10)
      else  !i_run
       tk = tk - 0.25
      end if  !i_run

!       alpha=0.0      !cluster
       alpha_prefac = 2*alpha/sqrt(pi)
       r_cutoff(1,1) = 3./alpha
       r_cutoff(2,2) = r_cutoff(1,1)
       r_cutoff(2,1) = r_cutoff(1,1)
       r_cutoff(1,2) = r_cutoff(1,1)
!       if (r_cutoff(1,1).lt.r_cut_short)
!     & stop "cut off too small for short range"
       q_cut_2 = (6.*alpha)**2  !cluster
!---   any peak |f(k|^2 in order 1/sqrt(N) gets cut off

       r_cutoff_max = 0.0
       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_max = max(r_cutoff_max,r_cutoff(i_type,j_type))
       end do
       end do

       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
        r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
     &  r_skin(i_type,j_type)
        r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
       end do
       end do


!          charge(1) = 0.      ! 1.3  fuer den neuen Hamiltonian, wird
!          charge(2) = 0.   ! -0.65 nachher wieder ueberschrieben

 ! add morse-stretch term and coulomb term
         do i_type = 1,n_type
         do j_type = 1,n_type

         slatercutoff = slaterint(i_type,j_type,r_cutoff(i_type,j_type))

         do i_potential=1, n_potential
           r_1 = i_potential*r_range(i_type,j_type)/n_potential

           slaterijr = slaterint(i_type,j_type,r_1)

           ! tabulate potential

        ! morse stretch term

        if (r_1.lt.r_cut_short) then
           pot_loc = trans_units*d0(i_type,j_type)*
&    (exp(gamma(i_type,j_type)*(1-r_cut_short/msr0(i_type,j_type)))
&      -2*exp(gamma(i_type,j_type)/2
&          *(1-r_cut_short/msr0(i_type,j_type))))
         r_pot_1(i_type,j_type,i_potential) = - pot_loc
&          + trans_units*d0(i_type,j_type)*
&    (exp(gamma(i_type,j_type)*(1-r_1/msr0(i_type,j_type)))
&      -2*exp(gamma(i_type,j_type)/2
&          *(1-r_1/msr0(i_type,j_type))))

        if (f_potential.eq.6) then
           r_pot_1(i_type,j_type,i_potential) =
&             r_pot_1(i_type,j_type,i_potential) +
&             constant_charge(i_type)*constant_charge(j_type)
&             *elec_prefac*((1./r_1-slaterijr)
&             -(1./r_cutoff(i_type,j_type)-slatercutoff))
        end if

        else
          r_pot_1(i_type,j_type,i_potential) = 0.0
       end if

     ! Slater orbital overlap

       r_pot_2q(i_type,j_type,i_potential) =
&          elec_prefac*(slaterijr-slatercutoff)
&         - elec_prefac*(1./r_1-1./r_cutoff(i_type,j_type))
&         +  erfcc(alpha*r_1)*  !cluster
&          elec_prefac/r_1 -
&          erfcc(alpha*r_cutoff(i_type,j_type))* !cluster
&          elec_prefac/r_cutoff(i_type,j_type)

       if ((f_potential.eq.6).or.(f_potential.eq.7)) then
         r_pot_2(i_type,j_type,i_potential) =
&          r_pot_2q(i_type,j_type,i_potential)
       end if

       if (f_potential.eq.5) then
         r_pot_2(i_type,j_type,i_potential) =
&          erfcc(alpha*r_1)*  !cluster
&          elec_prefac/r_1 -
&          erfcc(alpha*r_cutoff(i_type,j_type))* !cluster
&          elec_prefac/r_cutoff(i_type,j_type)
       end if

       if (r_1.ge.r_cutoff(i_type,j_type)) then
          r_pot_2(i_type,j_type,i_potential)=0.0
          r_pot_2q(i_type,j_type,i_potential)=0.0
       end if
  end do
  end do
  end do

   ! numerical derivative for the two potential terms

  do i_potential = 2, n_potential-1
     do i_type=1, n_type
        do j_type=1, n_type
           r_1 = i_potential*r_range(i_type,j_type)/n_potential
```

```
              r_pot_d_1(i_type,j_type,i_potential)=1./r_1*
     &             (r_pot_1(i_type,j_type,i_potential+1)-
     &             r_pot_1(i_type,j_type,i_potential-1))/
     &              (2.*r_range(i_type,j_type)/n_potential)
              r_pot_d_2(i_type,j_type,i_potential)=1./r_1*
     &             (r_pot_2(i_type,j_type,i_potential+1)-
     &             r_pot_2(i_type,j_type,i_potential-1))/
     &              (2.*r_range(i_type,j_type)/n_potential)
              r_pot_d_2q(i_type,j_type,i_potential)=1./r_1*
     &             (r_pot_2q(i_type,j_type,i_potential+1)-
     &             r_pot_2q(i_type,j_type,i_potential-1))/
     &              (2.*r_range(i_type,j_type)/n_potential)
            end do
          end do
        end do

        do i_potential = -n_potential,n_potential
          r_dummy = i_potential*2*pi/n_potential
          c_dummy = r_dummy *(0.,1.)
          tab_exp_ima(i_potential) = exp(c_dummy)
        end do

! ************** f_potential 8
      else if (f_potential.eq.8) then

      if (f_chargeadjust.ne.1) then
         stop "Vashishta potential does not work with variable charge"
      end if

      n_ang(1)=n_angular
      n_ang(2)=1

      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      constant_charge(1) = 1.6D0
      constant_charge(2) = -0.8D0

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
       jseed = iseed

         n_relax = 30
         n_obser = 30
         n_safe  = 30
!----    silicium
         mass(0,1) = 28.1*mass_amu
!----    oxygen
         mass(0,2) = 16.0*mass_amu
         vash_h(1,1) = 0.057*elec_prefac
         vash_h(2,2) = 51.692*elec_prefac
         vash_h(2,1) = 11.387*elec_prefac
         vash_h(1,2) = vash_h(2,1)
         vash_eta(1,1) = 11.d0
         vash_eta(2,2) = 7.d0
         vash_eta(2,1) = 9.d0
         vash_eta(1,2) = vash_eta(2,1)
         vash_alpha(1) = 0.d0*elec_prefac
         vash_alpha(2) = 2.4d0*elec_prefac
         vash_B(1) = 0.35
         vash_B(2) = 1.4
         vash_l(1) = 1.0
         vash_l(2) = 1.0
         vash_ctheta(1) = cos(pi*109.47/180.)
         vash_ctheta(2) = cos(pi*141.00/180.)
         vash_r0(1) = 2.6
         vash_r0(2) = 2.6

         alpha = 0.3  * (1002./n_part)**(1./6)
         alpha = 0.3
         v_cut = 0.01*elec_prefac*(1.2*2.4)/1.6/sqrt(1.*n_part)
         f_opt_rec = 0
         do i_type = 1,n_type
         do j_type = 1,n_type
          r_skin(i_type,j_type) = 0.45
         end do
         end do
         tk = 3600./n_trot
! ----    time step = 1 fs
         dt = 1.3091e-4
         f_pressure = 1
         press = .0

       if (n_order.eq.2) then
!----   check energy conservation
        friction(1) = 0.
        fric_box = 0.
        dt = dt/2
       else
        friction(1) = 0.01 / dt
        fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
        mass_box = mass_box / 8
      else    ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"

       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box

        read(10,200) char_dummy
        read(10,201) press
        read(10,210) f_pressure
        read(10,210) f_opt_rec
```

```
        read(10,200) char_dummy
        read(10,200) char_dummy
        read(10,200) char_dummy

         do i_type = 1,n_type
          read(10,202) (vash_h(i_type,j_type),j_type=1,n_type)
         end do

        read(10,200) char_dummy
         do i_type = 1,n_type
          read(10,202) (vash_eta(i_type,j_type),j_type=1,n_type)
         end do
        read(10,200) char_dummy
       read(10,202) (vash_alpha(i_type),i_type=1,n_type)
       read(10,202) (vash_B(i_type),i_type=1,n_type)
       read(10,202) (vash_l(i_type),i_type=1,n_type)
       read(10,202) (vash_ctheta(i_type),i_type=1,n_type)
       read(10,202) (vash_r0(i_type),i_type=1,n_type)

        read(10,200) char_dummy

         read(10,202) alpha, v_cut

        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
        end do

      end if ! i_dummy
      close(10)
      else  !i_run
       tk = tk - 0.25
      end if  !i_run

       alpha_prefac = 2*alpha/sqrt(pi)
       r_cutoff(1,1) = 3./alpha
       r_cutoff(2,2) = 3./alpha
       r_cutoff(2,1) = 3./alpha
       r_cutoff(1,2) = r_cutoff(2,1)
!       if (r_cutoff(1,1).lt.r_cut_short)
!     & stop "cut off too small for short range"
       q_cut_2 = (6.*alpha)**2
!---   any peak |f(k|^2 in order 1/sqrt(N) gets cut off

       r_cutoff_max = 0.0
       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_max = max(r_cutoff_max,r_cutoff(i_type,j_type))
       end do
       end do


       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
        r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
     &  r_skin(i_type,j_type)
        r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
       end do
       end do

!---   define shifts in energy

       do i_type = 1,n_type
       do j_type = 1,n_type
!----    electrostatic
        e_shift(i_type,j_type) = erfcc(alpha*r_cutoff(i_type,j_type))*
     &   elec_prefac/r_cutoff(i_type,j_type)

        pot_loc = r_cut_short**(vash_eta(i_type,j_type))
        pot_loc = vash_h(i_type,j_type)/pot_loc
        exp_pot = 0.5*(vash_alpha(i_type)*constant_charge(j_type)**2
     &        + vash_alpha(j_type)*constant_charge(i_type)**2)
        exp_pot = exp_pot/(r_cut_short**4)
        exp_pot = exp_pot * exp(-r_cut_short/4.43d0)
        e_shift_short(i_type,j_type) = pot_loc - exp_pot
       end do
       end do
!----   tabulate potential v(r) and dv/dr
       do i_type = 1,n_type
       do j_type = 1,n_type
       do i_potential = 1,n_potential
        r_1 = i_potential*r_range(i_type,j_type)/n_potential
        r_2 = r_1**2
!------  cutoff at shortrange interaction
        if(r_1.le.r_cut_short) then
           pot_loc = r_1**(vash_eta(i_type,j_type))
           pot_loc = vash_h(i_type,j_type)/pot_loc
           exp_pot = 0.5*(vash_alpha(i_type)*constant_charge(j_type)**2
     &           + vash_alpha(j_type)*constant_charge(i_type)**2)
           exp_pot = exp_pot/(r_1**4)
           exp_pot = exp_pot * exp(-r_1/4.43d0)
           r_pot_1(i_type,j_type,i_potential) = pot_loc - exp_pot
     &          -e_shift_short(i_type,j_type)
        else
           r_pot_1(i_type,j_type,i_potential) = 0.d0
        end if

        if (r_1.le.r_cutoff(i_type,j_type)) then
           r_pot_2(i_type,j_type,i_potential) = -e_shift(i_type,j_type)
     &             + erfcc(alpha*r_1)*elec_prefac/r_1
        else
           r_pot_2(i_type,j_type,i_potential) = 0.d0
        end if

!         r_pot_d_2(i_type,j_type,i_potential) = -(1./r_2) * (v_coulomb+
!     &       elec_prefac*alpha_prefac*exp(-(alpha*r_1)**2))
       end do
       end do
       end do

      do i_potential = 2, n_potential-1
         do i_type=1, n_type
            do j_type=1, n_type
               r_1 = i_potential*r_range(i_type,j_type)/n_potential
               r_pot_d_1(i_type,j_type,i_potential)=1./r_1*
     &              (r_pot_1(i_type,j_type,i_potential+1)-
     &              r_pot_1(i_type,j_type,i_potential-1))/
     &               (2.*r_range(i_type,j_type)/n_potential)
               r_pot_d_2(i_type,j_type,i_potential)=1./r_1*
     &              (r_pot_2(i_type,j_type,i_potential+1)-
     &              r_pot_2(i_type,j_type,i_potential-1))/
     &               (2.*r_range(i_type,j_type)/n_potential)
            end do
```

```
          end do
        end do

         do i_potential = -n_potential,n_potential
          r_dummy = i_potential*2*pi/n_potential
          c_dummy = r_dummy *(0.,1.)
          tab_exp_ima(i_potential) = exp(c_dummy)
         end do

! ************** f_potential 9 (Scandolo)

      else if (f_potential.eq.9) then

      if (f_chargeadjust.ne.1) then
         stop "Scandolo potential does not work with variable charge"
      end if

      if (n_trot.ne.1) then
         stop "Scandolo potential does not work with n_trotter > 1"
      end if

      if (i_run.eq.1) then
      open(10, file="pimd.inp", status="old")
      read(10,210) i_dummy, char_dummy

      if (i_dummy.eq.0) then
!----  take default parameters
       n_relax = 50
       n_obser = 50
       n_safe  = 100
       iseed = 47591+2*int(n_part/2)+2*int(10*tk)+n_trot
       jseed = iseed

         n_relax = 0
         n_obser = 0
         n_safe  = 0
!----    silicium
         mass(0,1) = 28.1*mass_amu
!----    oxygen
         mass(0,2) = 16.0*mass_amu

         alpha = 0.3  * (1002./n_part)**(1./6)
         alpha = 0.3
         v_cut = 0.01*elec_prefac*(1.2*2.4)/1.6/sqrt(1.*n_part)
         f_opt_rec = 0
         do i_type = 1,n_type
         do j_type = 1,n_type
          r_skin(i_type,j_type) = 0.45
         end do
         end do
         tk = 3600./n_trot
! ----   time step = 1 fs
         dt = 1.3091e-4
         f_pressure = 1
         press = .0

       if (n_order.eq.2) then
!----   check energy conservation
        friction(1) = 0.
        fric_box = 0.
        dt = dt/2
       else
        friction(1) = 0.01 / dt
        fric_box = 0.01 / dt
       end if
       friction(2) = friction(1)
!---   define inertia for box deformation
       mass_box = (n_silic*mass(0,1) + n_oxy*mass(0,2)) / 12
        mass_box = mass_box / 8
      else    ! i_dummy
       if (i_dummy.ne.f_potential)
     &     stop "f_potential and pimd.inp mismatch"
       read(10,210) i_dummy, char_dummy
       if (i_dummy.ne.f_chargeadjust)
     &     stop "f_chargeadjust and pimd.inp mismatch"

       read(10,210) n_relax
       read(10,210) n_obser
       read(10,210) n_safe
       read(10,210) iseed
       read(10,200) char_dummy
       read(10,201) dt
       read(10,201) tk
       read(10,200) char_dummy
       read(10,202) (mass(0,i_type),i_type=1,n_type)
       read(10,202) (friction(i_type),i_type=1,n_type)
       read(10,202) mass_box, fric_box

        read(10,200) char_dummy
        read(10,201) press
        read(10,210) f_pressure

        read(10,200) char_dummy
        read(10,200) char_dummy
        read(10,200) char_dummy
        read(10,200) char_dummy

         read(10,202) alpha, v_cut

        read(10,200) char_dummy
        do i_type = 1,n_type
         read(10,202) (r_skin(i_type,j_type),j_type=1,n_type)
        end do

      end if ! i_dummy
      close(10)
      else  !i_run
       tk = tk - 0.25
      end if  !i_run

       alpha_prefac = 2*alpha/sqrt(pi)
       r_cutoff(1,1) = 3./alpha
       r_cutoff(2,2) = 3./alpha
       r_cutoff(2,1) = 3./alpha
       r_cutoff(1,2) = r_cutoff(2,1)
!       if (r_cutoff(1,1).lt.r_cut_short)
!     & stop "cut off too small for short range"
       q_cut_2 = (6.*alpha)**2
!---   any peak |f(k|^2 in order 1/sqrt(N) gets cut off

       r_cutoff_max = 0.0
       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_max = max(r_cutoff_max,r_cutoff(i_type,j_type))
```

```
       end do
       end do


       do i_type = 1,n_type
       do j_type = 1,n_type
        r_cutoff_2(i_type,j_type) = r_cutoff(i_type,j_type)**2
        r_range(i_type,j_type) = r_cutoff(i_type,j_type) +
     &  r_skin(i_type,j_type)
        r_range_2(i_type,j_type) = r_range(i_type,j_type)**2
       end do
       end do


       n_pbc=1
       do i_dim = 1, n_dim
          if ((int(r_cutoff_max/scal_0(i_dim))+1).gt.n_pbc) then
             n_pbc=(int(r_cutoff_max/scal_0(i_dim))+1)
          end if
       end do


! potential is calculated in paul tangneys routine

      else
        stoP "f_potential flagg makes no sense"
      end if

! ************** all values f_potential

      if (i_run.eq.1) mass_box = n_trot * mass_box

      jseed = iseed
      if (f_potential.eq.1) f_pressure = 0

      dt = dt/n_time_scal
      dt_2 = dt**2
      tkp = tk*n_trot
      sqrtp = sqrt(real(n_trot))
      beta = 1.d0/tk
      betap = 1.d0/tkp
      do i_type = 1,n_type
       k_quant(i_type) = mass(0,i_type)/betap**2
!----  spring constant in reciprocal space
       do i_trot = 0,n_trot-1
        k_eigen(i_trot,i_type) = 4*k_quant(i_type)
     &  *sin(i_trot*pi/n_trot)**2
       end do
       v_therm(0,i_type) = sqrt(tkp/mass(0,i_type))
       w_therm = sqrt(tkp/mass_box)
!----  ficticiuos masses
       do i_trot = 1,n_trot-1
        if(f_potential.eq.1) then
         mass(i_trot,i_type) = mass(0,i_type)*
     &   (1.+k_eigen(i_trot,i_type))
         mass(i_trot,i_type) = mass(i_trot,i_type) / n_time_scal**2
        else if (f_potential.eq.2) then
         mass(i_trot,i_type) = mass(0,i_type)*
     &   (256*epsil(i_type,i_type)/sigma(i_type,i_type)**2
     &   +k_eigen(i_trot,i_type))/
     &   (256*epsil(i_type,i_type)/sigma(i_type,i_type)**2)
         mass(i_trot,i_type) = mass(i_trot,i_type) / n_time_scal**2
        else if (f_potential.eq.3) then
         mass(i_trot,i_type) = mass(0,i_type)*
     &   (1.25e6+k_eigen(i_trot,i_type)) / 1.25e6
         mass(i_trot,i_type) = mass(i_trot,i_type) / n_time_scal**2
        else if (f_potential.eq.4) then
         mass(i_trot,i_type) = mass(0,i_type)*
     &   (425.733+k_eigen(i_trot,i_type)) / 425.733
         mass(i_trot,i_type) = mass(i_trot,i_type) / n_time_scal**2
        else if ((f_potential.ge.5).and.(f_potential.le.8)) then
         mass(i_trot,i_type) = mass(0,i_type)*
     &   (1.25e6+k_eigen(i_trot,i_type)) / 1.25e6
         mass(i_trot,i_type) = mass(i_trot,i_type) / n_time_scal**2
        end if
        v_therm(i_trot,i_type) = sqrt(tkp/mass(i_trot,i_type))
       end do
!----  width of Gaussian distributed noise
       do i_trot = 0,n_trot-1
        rf_width(i_trot,i_type) =
     &  sqrt(2*tkp*mass(i_trot,i_type)*friction(i_type)/dt)
!----   linear block distribution
        rf_width(i_trot,i_type) = sqrt3*rf_width(i_trot,i_type)
        if(mod(2*i_trot,n_trot).ne.0) then
         rf_width(i_trot,i_type) = rf_width(i_trot,i_type)/sqrt2
        end if
       end do
      end do
      rfb_width = sqrt( 2*tkp*mass_box*fric_box/dt )
      rfb_width = sqrt3 * rfb_width

!---  initialize predictor coefficients
      do i_order = 0,n_order_max-1
       do j_order = 0,n_order_max
        pred_coef(i_order,j_order) = 0.
       end do
       pred_coef(0,i_order+1) = 1.
       if(i_order.ge.1) pred_coef(1,i_order+1) = i_order+1
       pred_coef(i_order,i_order) = 1.
      end do
      pred_coef(2,3) =  3.
      pred_coef(2,4) =  6.
      pred_coef(2,5) = 10.
      pred_coef(3,4) =  4.
      pred_coef(3,5) = 10.
      pred_coef(4,5) =  5.
      do i_order = 0,n_order-1
       do j_order = 0,n_order
        predict_coef(i_order,j_order) = 0.
       end do
       do j_order = i_order,n_order
        predict_coef(i_order,j_order) = pred_coef(i_order,j_order)
       end do
      end do

!---  initialize corrector
      do i_order = 1,n_order
       corr_coef(i_order) = 0.
      end do
      i_order = n_order
      if(i_order.eq.2) then
!----  velocity Verlet algorithm
       corr_coef(0) = 0.
       corr_coef(1) = 1.
```

```
      else if (i_order.eq.3) then
       corr_coef(0) = 1./6
       corr_coef(1) = 5./6
       corr_coef(3) = 1./3
      else if (i_order.eq.4) then
       corr_coef(0) = 19./90
!      corr_coef(0) = 19./120  only if thermostat absent
       corr_coef(1) = 3./4
       corr_coef(3) = 1./2
       corr_coef(4) = 1./12
      else if (i_order.eq.5) then
       corr_coef(0) = 3./16
!      corr_coef(0) = 3./20    only if thermostat absent
       corr_coef(1) = 251./360
       corr_coef(3) = 11./18
       corr_coef(4) = 1./6
       corr_coef(5) = 1./60
      else
       stoP 'Order for predictor corrector unavailable'
      end if
      corr_coef(2) = 1.
      do i_order = 0,n_order
       correct_coef(i_order) = corr_coef(i_order)
      end do

      if(i_run.eq.1) then
!----  initialize random number generator
       kptr = 1
       call inr250(mz,iseed,kptr)
       if(n_run.gt.1) then
        n_relax = 1*n_relax
       end if
      else if(n_relax.eq.2) then
       n_relax = n_relax/1
      end if

  200 format(a)
  210 format(1i14,a)
  220 format(2i14,a)
  201 format(1e14.6,a)
  202 format(2e14.6,a)
  203 format(3e14.6,a)

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      double precision function slaterint(i_type_sl,j_type_sl,r_1_sl)
      implicit none
      include "pimd.com.f"
      double precision r_1_sl
      integer i_type_sl, j_type_sl
      double precision atemp, btemp
      integer binomial, factorial
      integer v,k,l
      double precision sum1, sum2, sum21, sum22

      if (r_1_sl.gt.1e-8) then
      sum1=0
      do v = 0, 2*msn(i_type_sl)-1
         sum1=sum1+binomial(2*msn(i_type_sl)-1,v)*
     &        atemp(2*msn(i_type_sl)-1-v,1.d0*
     &                  mszeta(i_type_sl,j_type_sl)*r_1_sl)*
     &        btemp(v,1.d0*mszeta(i_type_sl,j_type_sl)*r_1_sl)
      end do
      sum1=(r_1_sl/2)**(2*msn(i_type_sl))*sum1
      sum2=0
      do v = 1, 2*msn(j_type_sl)
         sum22=0
         do k = 0, 2*msn(i_type_sl)-1
            sum21=0
            do l = 0, 2*msn(j_type_sl)-v
               sum21=sum21+binomial(2*msn(j_type_sl)-v,l)*(-1)**l*
     &              atemp(2*msn(i_type_sl)+2*msn(j_type_sl)-k-v-l-1,
     &               1.d0*(mszeta(i_type_sl,j_type_sl)
     &                    +mszeta(j_type_sl,i_type_sl))*r_1_sl)*
     &      btemp(k+l,1.d0*(mszeta(i_type_sl,j_type_sl)
     &                     -mszeta(j_type_sl,i_type_sl))*r_1_sl)
            end do
            sum22=sum22+sum21*binomial(2*msn(i_type_sl)-1,k)

         end do
         sum22=sum22
         r_dummy=(r_1_sl/2)**(2*msn(i_type_sl)+2*msn(j_type_sl)-v)
         r_dummy=r_dummy*v*(2*
     &          mszeta(j_type_sl,i_type_sl))**(2*msn(j_type_sl)-v)
        r_dummy=r_dummy/(factorial(2*msn(j_type_sl)-v)*2*msn(j_type_sl))
         sum2=sum2+r_dummy*sum22
      end do

      r_dummy=0.5*(2*mszeta(i_type_sl,j_type_sl))**(2*msn(i_type_sl)+1)
      r_dummy=r_dummy/factorial(2*msn(i_type_sl))
      slaterint=r_dummy*(sum1-sum2)

      else ! special case r=0

      sum2=0

      do v = 1, 2*msn(j_type_sl)
         r_dummy=(2.0*(mszeta(i_type_sl,j_type_sl)
     &        +mszeta(j_type_sl,i_type_sl)))**
     &                    (2*msn(i_type_sl)+2*msn(j_type_sl)-v)
         sum21= 1.0* factorial(2*msn(i_type_sl)+2*msn(j_type_sl)-v-1)/
     &           r_dummy
         sum21= sum21 *v*((2*mszeta(j_type_sl,i_type_sl))**
     &         (2*msn(j_type_sl)-v))/
     &          (factorial(2*msn(j_type_sl)-v)*2*msn(j_type_sl))
         sum2=sum2+sum21
      end do

      r_dummy=(2.0*mszeta(i_type_sl,j_type_sl))**(2*msn(i_type_sl)+1)
      r_dummy=r_dummy*1.0/(factorial(2*msn(i_type_sl)))

      slaterint=-r_dummy*sum2+mszeta(i_type_sl,j_type_sl)/msn(i_type_sl)

      end if

      end function

!---------------------------------------

      double precision function slaterint_d(i_type_sl,j_type_sl,r_1_sl)
      implicit none
      include "pimd.com.f"
```

```
      double precision r_1_sl
      integer i_type_sl, j_type_sl
      double precision atemp, btemp
      integer binomial, factorial
      integer v,k,l
      double precision sum1, sum2, sum21, sum22, sum211

      sum1=0
      do v = 0, 2*msn(i_type_sl)-1
         sum1=sum1+binomial(2*msn(i_type_sl)-1,v)*
     &      ( msn(i_type_sl)*
     &        atemp(2*msn(i_type_sl)-1-v,1.d0
     &           *mszeta(i_type_sl,j_type_sl)*r_1_sl)*
     &        btemp(v,1.d0*mszeta(i_type_sl,j_type_sl)*r_1_sl)
     &      - r_1_sl/2*mszeta(i_type_sl,j_type_sl)*
     &        atemp(2*msn(i_type_sl)-v,1.d0
     &              *mszeta(i_type_sl,j_type_sl)*r_1_sl)*
     &        btemp(v,1.d0*mszeta(i_type_sl,j_type_sl)*r_1_sl)
     &      - r_1_sl/2*mszeta(i_type_sl,j_type_sl)*
     &        atemp(2*msn(i_type_sl)-1-v,
     &              1.d0*mszeta(i_type_sl,j_type_sl)*r_1_sl)*
     &        btemp(v+1,1.d0*mszeta(i_type_sl,j_type_sl)*r_1_sl) )
      end do
      sum1=(r_1_sl/2)**(2*msn(i_type_sl)-1)*sum1
      sum2=0
      do v = 1, 2*msn(j_type_sl)
         sum22=0
         do k = 0, 2*msn(i_type_sl)-1
            sum21=0
            do l = 0, 2*msn(j_type_sl)-v
               sum211 = (msn(i_type_sl)+msn(j_type_sl)-0.5*v)*
     &             atemp(2*msn(i_type_sl)+2*msn(j_type_sl)-k-v-l-1,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                        +mszeta(j_type_sl,i_type_sl))*r_1_sl)*
     &             btemp(k+l,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                         -mszeta(j_type_sl,i_type_sl))*r_1_sl)
               sum211 = sum211 - .5d0*(mszeta(i_type_sl,j_type_sl)
     &                         +mszeta(j_type_sl,i_type_sl))*r_1_sl*
     &             atemp(2*msn(i_type_sl)+2*msn(j_type_sl)-k-v-l,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                         +mszeta(j_type_sl,i_type_sl))*r_1_sl)*
     &             btemp(k+l,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                          -mszeta(j_type_sl,i_type_sl))*r_1_sl)
               sum211=sum211 - .5d0*(mszeta(i_type_sl,j_type_sl)
     &                          -mszeta(j_type_sl,i_type_sl))*r_1_sl*
     &             atemp(2*msn(i_type_sl)+2*msn(j_type_sl)-k-v-l-1,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                          +mszeta(j_type_sl,i_type_sl))*r_1_sl)*
     &             btemp(k+l+1,
     &             1.d0*(mszeta(i_type_sl,j_type_sl)
     &                         -mszeta(j_type_sl,i_type_sl))*r_1_sl)
               sum21=sum21+binomial(2*msn(j_type_sl)-v,l)*(-1)**l
     &              *sum211
            end do
            sum22=sum22+sum21*binomial(2*msn(i_type_sl)-1,k)
         end do
         sum22=sum22
         r_dummy=(r_1_sl/2)**(2*msn(i_type_sl)+2*msn(j_type_sl)-v-1)
         r_dummy=r_dummy*v*
     &           (2*mszeta(j_type_sl,i_type_sl))**(2*msn(j_type_sl)-v)
           r_dummy=r_dummy/(factorial(2*msn(j_type_sl)-v)*2*msn(j_type_sl))
            sum2=sum2+r_dummy*sum22
         end do

         r_dummy=0.5*(2*mszeta(i_type_sl,j_type_sl))**(2*msn(i_type_sl)+1)
         r_dummy=r_dummy/factorial(2*msn(i_type_sl))
         slaterint_d=r_dummy*(sum1-sum2)
         end function

!---------------------------------------

         integer function factorial(k)
         implicit none
         integer k
         integer i,fac
         fac=1
         do i=1, k
            fac=fac*i
         end do
         factorial=fac
         end function

!---------------------------------------

         integer function binomial(n,l)
         implicit none
         integer factorial

         integer n,l
         integer denom,num,i
         denom=1
         num=1
         do i=1,l
            denom=denom*i
            num=num*(n-i+1)
         end do
         binomial=num/denom
         end function

!---------------------------------------

         double precision function atemp(k,x)
         implicit none
         integer factorial

         integer k
         double precision x
         double precision sum1
         integer v
         sum1=0
         do v=0, k
            sum1=sum1+x**v/factorial(v)
         end do
         atemp=exp(-x)*factorial(k)/(x**(k+1))*sum1
         end function

!---------------------------------------

         double precision function btemp(k,x)
         implicit none
         integer factorial, k, v
         double precision atemp, x, sum1, sum1b, at, frac
```

```
      if (abs(x).gt.1e-9) then
         sum1=0.d0
         if (mod(k,2).eq.0) then
            frac=1.d0
            do v=0, 10000
               sum1b=2.d0*frac/(k+(2.d0*v)+1.d0)
               frac=frac*(x**2)/((2.d0*v+1.d0)*(2.d0*v+2.d0))
               sum1=sum1+sum1b
            end do
         else
            frac=-x
            do v=0, 10000
               sum1b=2.d0*frac/(k+(2.d0*v+1.d0)+1.d0)
               frac=frac*(x**2)/((2.d0*v+2.d0)*(2.d0*v+3.d0))
               sum1=sum1+sum1b
            end do
         end if
         btemp=sum1
      else
         btemp=1.d0*(1-(-1)**(k+1))/(k+1.d0)
      end if

      end function


! ! ! ! ! ! ! ! ! ! !

      subroutine aziz_local(r,aziz)
      implicit none

!---  Lennard Jones as check
!     real r
!     real eps, sig
!     double precision v_pot
!     eps = 10.22
!     sig = 2.556
!     v_pot = 4*eps * (sig/r)**6 * ((sig/r)**6-1.)
!---  true aziz potential
      double precision aziz
      double precision r, x, eps, rm, d, prefac
      integer i_coef, n_coef
      parameter (n_coef=3)
      double precision a, alpha, beta, c(0:n_coef-1)
      eps = 10.948
      rm = 2.963
      d = 1.4826
      a = 1.8443101e5
      alpha = 10.433295
      beta = -2.279651
      c(0) = 1.36744521
      c(1) = 0.42123807
      c(2) = 0.17473318
      x = r / rm
      aziz = 0.
      do i_coef = 0, n_coef-1
       aziz = aziz + c(i_coef) / x**(2*i_coef+6)
      end do
      if (x.lt.d) then
       prefac = exp(-(d/x-1.)**2)
      else
       prefac = 1.
      end if
      aziz = - prefac * aziz
      aziz = aziz + a * exp(-alpha*x+beta*x**2)
      aziz = eps * aziz
! 666 for_sca_mfa_cor = 4*pi * eps*c(0) * rm**6 / 3

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine conf_init
      implicit none
      include "pimd.com.f"
      integer di_trot
      double precision x_quant(2*n_trot),f_quant(2*n_trot)
      common /xf_quant/ x_quant,f_quant

      open (10, file="conf.sta", status="old")
      read(10,*) i_dummy
      if (i_dummy.eq.0) then
       call conf_default
      else
       read(10,*) i_trot
       if (i_trot.eq.n_trot) then
        di_trot = 1
       else if (i_trot.eq.2*n_trot) then
        di_trot = 1
       else if (2*i_trot.eq.n_trot) then
        di_trot = 2
       else
        di_trot = 1
        stoP "trotter number incompabtible"
       end if
       read(10,103) (scal_0(i_dim),i_dim=1,n_dim+3)
       read(10,103) (scal_x(1,i_dim),i_dim=1,n_dim+3)
       do i_trot = 0, n_trot-1, di_trot
        do i_part = 1,n_part
         read(10,144) type(i_part),
     &   (pbc0(i_dim,i_trot,i_part),i_dim=1,n_dim),
     &   (r0(i_dim,i_trot,i_part),i_dim=1,n_dim),
     &   charge(i_part)
         if (di_trot.eq.2) then
          do i_dim = 1, n_dim
           pbc0(i_dim,i_trot+1,i_part) = pbc0(i_dim,i_trot,i_part)
           r0(i_dim,i_trot+1,i_part) = r0(i_dim,i_trot,i_part)
          end do
         end if
        end do
       end do
!---   set pbc back to small numbers
       do i_part = 1,n_part
       do i_dim = 1, n_dim
       do i_trot = n_trot-1, 0, -1
!        pbc0(i_dim,i_trot,i_part) = pbc0(i_dim,i_trot,i_part)
!     &  - pbc0(i_dim,0,i_part)
! dont do that, we like bigger numbers
       end do
       end do
       end do
       do i_dim = 1, n_dim + e_dim
        scal_x(1,i_dim) = scal_x(1,i_dim)*dt
       end do
```

```
       if (e_dim.eq.0) then
        do i_dim = 1, n_dim
         scal_x(1,n_dim+i_dim) = 0.
        end do
       end if
      end if

!---- set BKS charges back to default
      if (f_potential.eq.3) then
         do i_part = 1, n_part
            if (type(i_part).eq.1) then
               charge(i_part) = 2.4
            else
               charge(i_part) = -1.2
            end if
         end do
      end if

 !---- set Scandolo charges back to default
       if (f_potential.eq.9) then
          do i_part = 1, n_part
             if (type(i_part).eq.1) then
                charge(i_part) = 2.76514
             else
                charge(i_part) = -1.38257
             end if
          end do
       end if


!---- calculate default interaction range and skin

      if (f_pressure.eq.0) then
       do i_dim = 1, n_dim + e_dim
        scal_x(1,i_dim) = 0.
       end do
      end if

      n_neigh_max = 0

!---  define referent points for particles
      do i_part = 1,n_part
      do i_dim = 1,n_dim
       r_ref(i_dim,i_part) = r0(i_dim,0,i_part)+
     & 1. * pbc0(i_dim,0,i_part)
      end do
      end do

! --  initialize u0 field
      do i_part = 1, n_part
      do i_dim = 1, n_dim
       do i_trot = 0, n_trot-1
        x_quant(2*i_trot+1) = r0(i_dim,i_trot,i_part) +
     &  1. * pbc0(i_dim,i_trot,i_part) - r_ref(i_dim,i_part)
        x_quant(2*i_trot+1) = x_quant(2*i_trot+1) / n_trot * sqrtp
        x_quant(2*i_trot+2) = 0.
       end do
! ---  transform x_quant into rec. space
       call four1(x_quant,n_trot,1)
       do i_trot = 1, 2*n_trot
        u0(i_dim,i_trot,i_part) = x_quant(i_trot)
       end do
      end do
      end do

!---- initialize volume and velocities
      call scal_0_update
      call vel_default(i_dummy)
      call calc_rv_real

      if (f_potential.ne.1) call pb_condit
      call trans_recip
      if (f_potential.ne.1) call binning

      do i_part = 1,n_part
      do i_trot = 0,n_trot-1
      do i_dim = 1,n_dim
       force(i_dim,i_trot,i_part) = 0.d0
      end do
      end do
      end do

      if(f_potential.eq.1) then
       call harm_osc
      else
       call scal_0_update
       if (f_potential.eq.2) then
        call lennard_jones
       else if (f_potential.eq.3) then
        self_energy = 0.
        do i_part = 1, n_part
         self_energy = self_energy + charge(i_part)**2
        end do
        self_energy = -n_trot*elec_prefac*alpha*self_energy/sqrt(pi)
        call bks_potential
       else if (f_potential.eq.4) then
        call bks_potential
       else if ((f_potential.ge.5).and.(f_potential.le.8)) then
! hier aendern ?
        self_energy = 0.
        do i_part = 1, n_part
         self_energy = self_energy + charge(i_part)**2
        end do
        self_energy = -n_trot*elec_prefac*alpha*self_energy/sqrt(pi)
        call goddard_potential
       end if
      end if
      call trans_recip
      if (f_potential.ne.1) call box_chain
      call thermostat

!---- convert forces into accelerations
      do i_dim = 1,n_dim
      do i_trot = 0,n_trot-1
      do i_part = 1,n_part
       i_type = type(i_part)
       ux(2,i_dim,2*i_trot+1,i_part) =
     & f0(i_dim,2*i_trot+1,i_part)/mass(i_trot,i_type)*dt_2
       ux(2,i_dim,2*i_trot+2,i_part) =
     & f0(i_dim,2*i_trot+2,i_part)/mass(i_trot,i_type)*dt_2
      end do
      end do
      end do
```

```
!---- box shape
      do i_dim = 1,n_dim + e_dim
       scal_x(2,i_dim) = force_scal(i_dim) / mass_box * dt_2 / 2
      end do

      call conf_save
      if (f_potential.eq.3) call init_bonds
      if ((f_potential.ge.5).and.(f_potential.le.9)) call init_bonds

!---- count atoms per type
      do i_type = 1, n_type
         n_atomtype(i_type) = 0
      end do
      do i_part = 1, n_part
         i_type = type(i_part)
         n_atomtype(i_type) = n_atomtype(i_type) + 1
      end do


  103 format(3f13.5)
  144 format(4i5,4f14.6)

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine init_bonds
      implicit none
      include "pimd.com.f"
      double precision r_2

      r_neigh_min = 1.0
      r_neigh_max = 2.3
      do i_part = 1, n_part
       i_type = type(i_part)
       do i_dummy = 1, 4
        i_neigh_ori(i_part,i_dummy) = 0
       end do
       i_dummy = 0
       do j_part = 1, n_part
        j_type = type(j_part)
        if (i_type.ne.j_type) then
         call calc_distance(r_2,0,i_part,j_part)
         r_dummy = sqrt(r_2)
         if (r_dummy.lt.r_neigh_max) then
! ----     neighbor found
          i_dummy = i_dummy + 1
          if (i_dummy.le.4) then
           i_neigh_ori(i_part,i_dummy) = j_part
          else
           write(*,*) "more than 4 nearest neighbors"
          end if
         end if
        end if
       end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine scal_0_update
      implicit none
      include "pimd.com.f"
      double precision vec_dummy(n_dim+3), mat_dummy(n_dim,n_dim),
     &q_2_min
      double precision r_mat(n_dim,n_dim)

      call trans_to_tens(scal_0,h_mat)
      call square_mat(h_mat,h_mat_2)
      call trans_to_voigt(h_mat_2,met_ten_voi)

      volume = scal_0(1)*scal_0(2)*scal_0(3)
      if (e_dim.eq.3) then
       volume = volume +
     & 2*scal_0(4)*scal_0(5)*scal_0(6) - scal_0(4)**2*scal_0(3)
     & - scal_0(5)**2*scal_0(1) - scal_0(6)**2*scal_0(2)
      end if

      do i_dim = 1, n_dim + 3
       vec_dummy(i_dim) = scal_x(1,i_dim)
      end do
      call trans_to_tens(vec_dummy,h_mat_dot)
      call mat_mul(h_mat,h_mat_dot,mat_dummy)
      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
       h_mat_2d(i_dim,j_dim) = mat_dummy(i_dim,j_dim)
      end do
      end do
      call mat_mul(h_mat_dot,h_mat,mat_dummy)
      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
       h_mat_2d(i_dim,j_dim) =  h_mat_2d(i_dim,j_dim) +
     & mat_dummy(i_dim,j_dim)
       h_mat_2d(i_dim,j_dim) = h_mat_2d(i_dim,j_dim) / 1
      end do
      end do
      call inv_mat(h_mat_2,h_mat_2_inv)
      call mat_mul(h_mat_2_inv,h_mat_2d,g_mat_dot)
      call trans_to_voigt(g_mat_dot,g_voi_dot)
      call inv_mat(h_mat,h_mat_inv)

      if (f_potential.eq.3) then
!---   change: cutoff criterion for rec vector
       do i_dim = 1, n_dim
        do j_dim = 1, n_dim
         dq(i_dim,j_dim) = twopi * h_mat_inv(i_dim,j_dim)
        end do
       end do
       call jacobi(n_dim, dq, rot_mat, eigen, 100)
       q_2_min = eigen(1,1)**2
       do i_dim = 2, n_dim
        r_dummy = eigen(i_dim,i_dim)**2
        if (r_dummy.lt.q_2_min) q_2_min = r_dummy
       end do
       do i_dim = 1, n_dim
        nq_max(i_dim) = int(sqrt(q_cut_2/q_2_min))
       end do
      end if

      if ((f_potential.ge.5).and.(f_potential.le.8)) then
! hier aendern ?
!---   change: cutoff criterion for rec vector
       do i_dim = 1, n_dim
```

```
        do j_dim = 1, n_dim
         dq(i_dim,j_dim) = twopi * h_mat_inv(i_dim,j_dim)
        end do
       end do
       call jacobi(n_dim, dq, rot_mat, eigen, 100)
       q_2_min = eigen(1,1)**2
       do i_dim = 2, n_dim
        r_dummy = eigen(i_dim,i_dim)**2
        if (r_dummy.lt.q_2_min) q_2_min = r_dummy
       end do
       do i_dim = 1, n_dim
        nq_max(i_dim) = int(sqrt(q_cut_2/q_2_min))
       end do
      end if


!---  see whether min image conv. holds
      if (f_pressure.eq.1) then
       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
        r_mat(i_dim,j_dim) = h_mat(i_dim,j_dim)
       end do
       end do
       if (f_potential.gt.1) then
        call jacobi(n_dim, r_mat, rot_mat, eigen, 100)
        do i_dim = 1, n_dim
!         if( abs(eigen(i_dim,i_dim)) .lt. 2*r_cutoff_max)
!     &   stop "cutoff radius growing too strongly"
        end do
       end if
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine inv_mat(mat,mat_inv)
      implicit none
      integer i_dim, j_dim, sign
      double precision mat(3,3), mat_inv(3,3), det_mat
      double precision determ

      det_mat = determ(mat)

      do i_dim = 1, 3
      do j_dim = 1, 3
       mat_inv(i_dim,j_dim)  = 0.d0
      end do
      end do
      mat_inv(1,1) = mat(2,2)*mat(3,3) - mat(2,3)**2
      mat_inv(2,2) = mat(3,3)*mat(1,1) - mat(3,1)**2
      mat_inv(3,3) = mat(1,1)*mat(2,2) - mat(1,2)**2
      mat_inv(1,2) = - (mat(2,1)*mat(3,3) - mat(2,3)*mat(3,1))
      mat_inv(2,3) = - (mat(3,2)*mat(1,1) - mat(3,1)*mat(1,2))
      mat_inv(1,3) = + (mat(1,2)*mat(2,3) - mat(1,3)*mat(2,2))
      do i_dim = 1, 3
      do j_dim = i_dim, 3
       mat_inv(i_dim,j_dim) = mat_inv(i_dim,j_dim) / det_mat
       mat_inv(j_dim,i_dim) = mat_inv(i_dim,j_dim)
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      double precision function determ(mat)
      implicit none
      double precision mat(3,3)
      determ = mat(1,1)*mat(2,2)*mat(3,3) + 2*mat(1,2)*mat(2,3)*mat(3,1)
     & -mat(1,1)*mat(2,3)**2 -mat(2,2)*mat(3,1)**2 -mat(3,3)*mat(1,2)**2
      end function

! ! ! ! ! ! ! ! ! ! ! !

      subroutine mat_mul(mat1,mat2,mat3)
      implicit none
      integer i_dim, j_dim, k_dim
      double precision mat1(3,3), mat2(3,3), mat3(3,3)
      call zero_mat(mat3)
      do i_dim = 1, 3
      do j_dim = 1, 3
      do k_dim = 1, 3
       mat3(i_dim,j_dim) = mat3(i_dim,j_dim) +
     & mat1(i_dim,k_dim) * mat2(k_dim,j_dim)
      end do
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine zero_mat(matrix)
      implicit none
      integer i_dim,j_dim
      double precision matrix(3,3)

      do i_dim = 1,3
      do j_dim = 1,3
       matrix(j_dim,i_dim) = 0.
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine trans_to_tens(vector,matrix)
      implicit none
      double precision matrix(3,3), vector(6)
      matrix(1,1) = vector(1)
      matrix(2,2) = vector(2)
      matrix(3,3) = vector(3)
      matrix(1,2) = vector(4)
      matrix(2,1) = vector(4)
      matrix(2,3) = vector(5)
      matrix(3,2) = vector(5)
      matrix(3,1) = vector(6)
      matrix(1,3) = vector(6)
      end subroutine

! ! ! ! ! ! ! ! ! ! ! !
```

```
      subroutine square_mat(mat,mat_2)
      implicit none
      integer i_dim, j_dim, k_dim
      double precision mat(3,3), mat_2(3,3)
      do i_dim = 1, 3
      do j_dim = 1, 3
       mat_2(i_dim,j_dim) = 0.
      end do
      end do
      do i_dim = 1, 3
      do j_dim = 1, 3
      do k_dim = 1, 3
       mat_2(i_dim,j_dim) = mat_2(i_dim,j_dim) +
     & mat(k_dim,i_dim) * mat(k_dim,j_dim)
      end do
      end do
      end do
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine trans_to_voigt(matrix,vector)
      implicit none
      double precision matrix(3,3), vector(6)
      vector(1) = matrix(1,1)
      vector(2) = matrix(2,2)
      vector(3) = matrix(3,3)
      vector(4) = 2 * matrix(1,2)
      vector(5) = 2 * matrix(2,3)
      vector(6) = 2 * matrix(3,1)
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine conf_default
      implicit none
      include "pimd.com.f"
      double precision r_2
      double precision r_dummy_1,r_dummy_2,r_dummy_3
      integer i_bin_x,i_bin_y,i_bin_z

      do i_dim = 1, n_dim + 3
       if(i_dim.le.n_dim) then
        scal_0(i_dim) = 1.
       else
        scal_0(i_dim) = 0.
       end if
       do i_order = 1,n_order
        scal_x(i_order,i_dim) = 0.
       end do
      end do

      if (f_potential.eq.1) then
!----  harmonic oscillator
       do i_part = 1,n_part
        type(i_part) = 1
        do i_dim = 1,n_dim
         r0(i_dim,0,i_part) = 0.
        end do
       end do

      else if(f_symmetry.eq.1) then

!----  fcc symmetry

      if (f_potential.eq.2) then
!----   three particles in one box at density 1.05 (fcc)
       r_dummy = (4.d0/1.05)**(1.d0/3)
!----   clasical ground state if LJ next neighbor interaction
!      r_dummy = 1.587401
      else if (f_potential.eq.3) then
!----   appropriate distance for SiO2 "melt"
       r_dummy = (4.d0/0.069)**(1./3)
      else if (f_potential.eq.4) then
!----   Helium 3 at critical density
       r_dummy = (4.d0*7.26)**(1./3) * 2.556
!----   Helium 4 at crit. density
       r_dummy = (4.d0*96.14)**(1./3)
      else if ((f_potential.ge.5).and.(f_potential.le.9)) then
! hier aendern ?
!----   appropriate distance for SiO2 "melt"
       r_dummy = (4.d0/0.069)**(1./3)
      end if
      scal_0(1) = n_bin_x*r_dummy
      scal_0(2) = n_bin_y*r_dummy
      scal_0(3) = n_bin_z*r_dummy
!----  scale box length on right length scale
      if (f_potential.eq.2) then
       do i_dim = 1,n_dim
        scal_0(i_dim) = sigma(1,1) * scal_0(i_dim)
       end do
      end if

      i_part = 0

      do i_bin_x = 0,n_bin_x-1
      do i_bin_y = 0,n_bin_y-1
      do i_bin_z = 0,n_bin_z-1

!----   set first atom on edge of box
       i_part = i_part + 1
       type(i_part) = 1
       r0(1,0,i_part) = (real(i_bin_x)+0.0)/n_bin_x
       r0(2,0,i_part) = (real(i_bin_y)+0.0)/n_bin_y
       r0(3,0,i_part) = (real(i_bin_z)+0.0)/n_bin_z

!----   set next three atoms on face diagonal
       i_part = i_part + 1
       type(i_part) = 1
       r0(1,0,i_part) = r0(1,0,i_part-1) + 0.5/n_bin_x
       r0(2,0,i_part) = r0(2,0,i_part-1) + 0.5/n_bin_y
       r0(3,0,i_part) = r0(3,0,i_part-1)
       i_part = i_part + 1
       type(i_part) = 1
       r0(1,0,i_part) = r0(1,0,i_part-2) + 0.5/n_bin_x
       r0(2,0,i_part) = r0(2,0,i_part-2)
       r0(3,0,i_part) = r0(3,0,i_part-2) + 0.5/n_bin_z
       i_part = i_part + 1
       type(i_part) = 1
       r0(1,0,i_part) = r0(1,0,i_part-3)
       r0(2,0,i_part) = r0(2,0,i_part-3) + 0.5/n_bin_y
       r0(3,0,i_part) = r0(3,0,i_part-3) + 0.5/n_bin_z

      end do
      end do
```

```
       end do

!----  make every third particle Si if BKS potential
       if (f_potential.eq.3) then
        do i_part = 1,n_part
         if(mod(i_part,3).eq.1) then
          type(i_part) = 1
         else
          type(i_part) = 2
         end if
        end do
       end if

! hier aendern ?
!----  make every third particle Si if BKS potential
       if ((f_potential.ge.5).and.(f_potential.le.8)) then
        do i_part = 1,n_part
         if(mod(i_part,3).eq.1) then
          type(i_part) = 1
         else
          type(i_part) = 2
         end if
        end do
       end if

      else if((f_symmetry.eq.2).or.(f_symmetry.eq.3)) then
!----  tridymite: f_symmetry = 2
!----  cristobalit: f_symmetry = 3
       r_dummy = 3.14
       scal_0(1) = r_dummy*sqrt(8./3)*n_bin_x
       scal_0(2) = r_dummy*sqrt(8.)*n_bin_y
       if(f_symmetry.eq.2) then
        scal_0(3) = 8.*r_dummy/3*n_bin_z
       else
        scal_0(3) = 12.*r_dummy/3*n_bin_z
       end if

       r_dummy_1 = r_dummy/scal_0(1)
       r_dummy_2 = r_dummy/scal_0(2)
       r_dummy_3 = r_dummy/scal_0(3)

       i_part = 0

!----  set up Si atoms on Wurtzit structure

       do i_bin_x = 0,n_bin_x-1
       do i_bin_y = 0,n_bin_y-1
       do i_bin_z = 0,n_bin_z-1

!------ A double layer

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.001
        r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.001
        r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.001

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part) = r0(1,0,i_part-1)
        r0(2,0,i_part) = r0(2,0,i_part-1)
        r0(3,0,i_part) = r0(3,0,i_part-1)+r_dummy/scal_0(3)

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-2)

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-2)

!----   B double layer

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-4)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-4)+sqrt(1./12)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-4)+(1.+1./3)*r_dummy_3

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part) = r0(1,0,i_part-1)
        r0(2,0,i_part) = r0(2,0,i_part-1)
        r0(3,0,i_part) = r0(3,0,i_part-1)+r_dummy_3

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-2)

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-2)

!----   C double layer
        if(f_symmetry.eq.3) then
        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-4)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-4)+sqrt(1./12)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-4)+(1.+1./3)*r_dummy_3

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part) = r0(1,0,i_part-1)
        r0(2,0,i_part) = r0(2,0,i_part-1)
        r0(3,0,i_part) = r0(3,0,i_part-1)+r_dummy_3

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
        r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
        r0(3,0,i_part)=r0(3,0,i_part-2)

        i_part = i_part + 1
        type(i_part) = 1
        r0(1,0,i_part)=r0(1,0,i_part-2)+0.5*sqrt(8./3)*r_dummy_1
```

```
          r0(2,0,i_part)=r0(2,0,i_part-2)+sqrt(3./4)*sqrt(8./3)*r_dummy_2
          r0(3,0,i_part)=r0(3,0,i_part-2)
          end if

         end do
         end do
         end do

        else if(f_symmetry.eq.4) then

         r_dummy = 3.14
         scal_0(1) = 4*r_dummy/sqrt(3.)*n_bin_x
         scal_0(2) = 4*r_dummy/sqrt(3.)*n_bin_y
         scal_0(3) = 4*r_dummy/sqrt(3.)*n_bin_z

         i_part = 0.
         do i_bin_x = 0,n_bin_x-1
         do i_bin_y = 0,n_bin_y-1
         do i_bin_z = 0,n_bin_z-1
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = r0(1,0,i_part-1) + 0.5/n_bin_x
          r0(2,0,i_part) = r0(2,0,i_part-1) + 0.5/n_bin_x
          r0(3,0,i_part) = r0(3,0,i_part-1)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = r0(1,0,i_part-2) + 0.5/n_bin_x
          r0(2,0,i_part) = r0(2,0,i_part-2)
          r0(3,0,i_part) = r0(3,0,i_part-2) + 0.5/n_bin_x
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = r0(1,0,i_part-3)
          r0(2,0,i_part) = r0(2,0,i_part-3) + 0.5/n_bin_x
          r0(3,0,i_part) = r0(3,0,i_part-3) + 0.5/n_bin_x
          do j_part = 1,4
           i_part = i_part + 1
           type(i_part) = 1
           r0(1,0,i_part) = r0(1,0,i_part-4) + 0.25/n_bin_x
           r0(2,0,i_part) = r0(2,0,i_part-4) + 0.25/n_bin_y
           r0(3,0,i_part) = r0(3,0,i_part-4) + 0.25/n_bin_z
          end do
         end do
         end do
         end do

        else if(f_symmetry.eq.6) then
!----  beta quarz

         scal_0(1) = 4.9977*n_bin_x*1.02439
         scal_0(2) = 8.6563*n_bin_y*1.02439
         scal_0(3) = 5.4601*n_bin_z*1.02439

         i_part = 0
         do i_bin_x = 0,n_bin_x-1
         do i_bin_y = 0,n_bin_y-1
         do i_bin_z = 0,n_bin_z-1
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*0)/(1*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*0)/(1*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*0)/(1*n_bin_z)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*1)/(4*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*3)/(4*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*1)/(3*n_bin_z)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*1)/(4*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*1)/(4*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*2)/(3*n_bin_z)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*1)/(2*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*1)/(2*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*0)/(1*n_bin_z)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*3)/(4*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*3)/(4*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*2)/(3*n_bin_z)
          i_part = i_part + 1
          type(i_part) = 1
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + (1.*3)/(4*n_bin_x)
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + (1.*1)/(4*n_bin_y)
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + (1.*1)/(3*n_bin_z)

          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0./n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.707/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.5/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0./n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.2928/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.5/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.1892265/n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.8964/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.1666667/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.1892265/n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.1036/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.8333333/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.31084354/n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.6036/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.16666667/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
          r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.31084354/n_bin_x
          r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.3964/n_bin_y
          r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.83333333/n_bin_z
          i_part = i_part + 1
          type(i_part) = 2
```

```
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.5/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.7928/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.5/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.5/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.2072/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.5/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.68929655/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.6036/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.83333333/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.68929655/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.3964/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.16666667/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.81091358/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.8964/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.83333333/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.81091358/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.1036/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.16666667/n_bin_z

    end do
    end do
    end do
  else if(f_symmetry.eq.7) then
    alpha quarz
!----

    scal_0(1) = 4.91485*n_bin_x
    scal_0(2) = 8.51277*n_bin_y
    scal_0(3) = 5.40629*n_bin_z

    i_part = 0
    do i_bin_x = 0,n_bin_x-1
    do i_bin_y = 0,n_bin_y-1
    do i_bin_z = 0,n_bin_z-1
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0./n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0./n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0./n_bin_z
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.2977/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.7659/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.666667/n_bin_z
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.2977/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.2341/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.333333/n_bin_z
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.5/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.5/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0./n_bin_z
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.7977/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.7341/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.333333/n_bin_z
      i_part = i_part + 1
      type(i_part) = 1
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.7977/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.2659/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.6666667/n_bin_z

      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.09135/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.70655/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.453567/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.09135/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.29345/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.54643369/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.1922/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.9265/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.7869/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.1922/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.0735/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.2131/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.31185/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.36695/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.12023369/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.31185/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.63305/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.879767/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.59135/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.20655/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.453567/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.59135/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.79345/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.54643369/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.6922/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.5735/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.2131/n_bin_z
      i_part = i_part + 1
      type(i_part) = 2
      r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.6922/n_bin_x
      r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.4265/n_bin_y
      r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.7869/n_bin_z
```

```
        i_part = i_part + 1
        type(i_part) = 2
        r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.81185/n_bin_x
        r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.86695/n_bin_y
        r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.12023369/n_bin_z
        i_part = i_part + 1
        type(i_part) = 2
        r0(1,0,i_part) = i_bin_x*1./n_bin_x + 0.81185/n_bin_x
        r0(2,0,i_part) = i_bin_y*1./n_bin_y + 0.13305/n_bin_y
        r0(3,0,i_part) = i_bin_z*1./n_bin_z + 0.879767/n_bin_z
       end do
       end do
       end do

      end if

      if((f_symmetry.ge.2).and.(f_symmetry.lt.6)) then
       call scal_0_update
       call pb_condit
!----  put oxygens on bonds
       i_dummy = i_part
       do i_part = 1,n_silic-1
       do j_part = i_part+1,n_silic
        call calc_distance(r_2,0,i_part,j_part)
        if(r_2.le.(1.001*r_dummy)**2) then
!----     oxygen found
         i_dummy = i_dummy + 1
         type(i_dummy) = 2
         do i_dim = 1,n_dim
          r0(i_dim,0,i_dummy) = r0(i_dim,0,i_part) -
     &    delta_cell(i_dim) / 2
         end do
        end if
       end do
       end do
      end if

!---- generate collapsed polymers and set periodic boundaries to zero
      do i_part = 1,n_part
      do i_trot = 0,n_trot-1
      do i_dim = 1,n_dim
       r0(i_dim,i_trot,i_part) = r0(i_dim,0,i_part)
       pbc0(i_dim,i_trot,i_part) = 0
      end do
      end do
      end do

!---- set all velocities and higher derivatives to zero
      do i_part = 1,n_part
      do i_trot = 1,2*n_trot
      do i_dim = 1,n_dim
      do i_order = 1,n_order
       ux(i_order,i_dim,i_trot,i_part) = 0.
      end do
      end do
      end do
      end do

!---- set default charges according to type

      if (f_potential.eq.3) then
         do i_part = 1,n_part
            if(type(i_part).eq.1) then
               charge(i_part) = 2.4
            else if(type(i_part).eq.2) then
               charge(i_part) = -1.2
            end if
         end do
      else if ((f_potential.ge.5).and.(f_potential.le.7)) then
         do i_part = 1,n_part
            if(type(i_part).eq.1) then
               charge(i_part) = 1.3
            else if(type(i_part).eq.2) then
               charge(i_part) = -1.3/2.
            end if
         end do
      else if (f_potential.eq.8) then
         do i_part = 1,n_part
            if(type(i_part).eq.1) then
               charge(i_part) = 1.6
            else if(type(i_part).eq.2) then
               charge(i_part) = -1.6/2.
            end if
         end do
      else
         do i_part = 1,n_part
            if(type(i_part).eq.1) then
               charge(i_part) = 2.4
            else if(type(i_part).eq.2) then
               charge(i_part) = -1.2
            end if
         end do
      end if

      call calc_rv_real
      call conf_save

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine pb_condit
      implicit none
      include "pimd.com.f"

!---- move particles into box
      do i_part = 1,n_part
       do i_trot = 0,n_trot-1
        do i_dim = 1,n_dim
         if(r0(i_dim,i_trot,i_part).gt.1.) then
          r0(i_dim,i_trot,i_part) = r0(i_dim,i_trot,i_part)-1.
          pbc0(i_dim,i_trot,i_part) = pbc0(i_dim,i_trot,i_part) + 1
         else if(r0(i_dim,i_trot,i_part).lt.(0.d0)) then
          r0(i_dim,i_trot,i_part) = r0(i_dim,i_trot,i_part)+1.
          pbc0(i_dim,i_trot,i_part) = pbc0(i_dim,i_trot,i_part) - 1
         end if
        end do
       end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !
```

```
      subroutine vel_default(i_dummy)
      implicit none
      include "pimd.com.f"

      do i_part = 1,n_part
       i_type = type(i_part)
       do i_trot = 0,n_trot-1
!----   Gaussian width
        r_dummy = sqrt(tkp/mass(i_trot,i_type))*dt
!----   linear box distribution
        r_dummy = sqrt3*r_dummy
!----   take twice kinetic energy at t = 0
        if (i_dummy.eq.0) r_dummy = sqrt2*r_dummy
        if(mod(2*i_trot,n_trot).ne.0) r_dummy = r_dummy/sqrt2
        call r250(mz,random,n_dim,n_dim,kptr)
!----   term init velocities
        do i_dim = 1, n_dim
         ux(1,i_dim,2*i_trot+1,i_part) = 0.
         do j_dim = 1, n_dim
          ux(1,i_dim,2*i_trot+1,i_part) = ux(1,i_dim,2*i_trot+1,i_part)
     &    + h_mat_inv(i_dim,j_dim) * (2*random(j_dim)-1.) * r_dummy
         end do
        end do
       end do
!----  symmetrize
       do i_trot = 1,n_trot/2-1
       do i_dim = 1,n_dim
        ux(1,i_dim,2*i_trot+2,i_part) =
     &  ux(1,i_dim,2*(n_trot-i_trot)+1,i_part)
        ux(1,i_dim,2*(n_trot-i_trot)+1,i_part) =
     &  ux(1,i_dim,2*i_trot+1,i_part)
        ux(1,i_dim,2*(n_trot-i_trot)+2,i_part) =
     &  - ux(1,i_dim,2*i_trot+2,i_part)
       end do
       end do

      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine conf_save
      call conf_save_nr(-1)
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine conf_save_nr(filenr)
      implicit none
      include "pimd.com.f"

      character(20) filename
      integer filenr

      if (filenr.ge.0) then
         filename="conf."
         call addnumtostring(filename,filenr)
      else
         filename="conf.xmo"
      end if
```

```
      open(10,file="conf.end",status="unknown")
      write(10,'(i1)') 1
      write(10,*) n_trot
      write(10,103) (scal_0(i_dim),i_dim=1,n_dim+3)
      write(10,103) (scal_x(1,i_dim)/dt,i_dim=1,n_dim+3)
      do i_trot = 0,n_trot-1
       do i_part = 1,n_part
        write(10,144) type(i_part),
     &  (pbc0(i_dim,i_trot,i_part),i_dim=1,n_dim),
     &  (r0(i_dim,i_trot,i_part),i_dim=1,n_dim),
     &  charge(i_part)
       end do
      end do
      write(10,*) n_neigh_max,"  max. nunber of neighbors seein in run"
      write(10,*) n_neigh," dimension of neighbor field"
      close(10)

      if(i_run.eq.1) open(11,file=filename,status="unknown")
      write(11,'(i5)') n_part
      write(11,*)
      do i_part = 1,n_part
       if(type(i_part).eq.1) then
         write(11,204) "Si  ",(r_real(i_dim,0,i_part),i_dim=1,n_dim),
     &         charge(i_part)
       else if(type(i_part).eq.2) then
         write(11,204) "O   ",(r_real(i_dim,0,i_part),i_dim=1,n_dim),
     &         charge(i_part)
       end if
      end do
      if(i_run.eq.n_run) close(11)

      open(35,file="confavg.xmo",status="unknown")
      write(35,'(i5)') n_part
      write(35,*)
      do i_part = 1,n_part
       if(type(i_part).eq.1) then
         write(35,204) "Si  ",(r_real_avg(i_dim,0,i_part)/i_time,
     &         i_dim=1,n_dim),
     &         charge(i_part)
       else if(type(i_part).eq.2) then
         write(35,204) "O   ",(r_real_avg(i_dim,0,i_part)/i_time,
     &         i_dim=1,n_dim),
     &         charge(i_part)
       end if
      end do
      close(35)

      open(34,file="confavg.pol",status="unknown")
      do i_part = 1, n_part
         write(34,104) (avg_dipole_part(i_dim,i_part)/i_time,
     &        i_dim=0,n_dim)
      end do
      close(34)

      open(36,file="conf.pol",status="unknown")
      do i_part = 1, n_part
         write(36,104) (dipole(i_dim,i_part),
     &        i_dim=1,n_dim)
      end do
      close(36)

  103 format(3f13.5)
```

```
  104 format(4f13.5)
  144 format(4i5,4f14.6)

  204 format(a,3e13.4,e30.21)

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine trans_recip
      implicit none
      include "pimd.com.f"
      double precision x_quant(2*n_trot),f_quant(2*n_trot)
      common /xf_quant/ x_quant,f_quant

      do i_part = 1,n_part
      do i_dim = 1,n_dim

       do i_trot = 0,n_trot-1
        f_quant(2*i_trot+1) = force(i_dim,i_trot,i_part)/n_trot*sqrtp
!----   above normalizations necessary because four1 incorrect
        f_quant(2*i_trot+2) = 0.
       end do

!----  transform into reciprocal space
       call four1(f_quant,n_trot,1)

       do i_trot = 1,2*n_trot
        f0(i_dim,i_trot,i_part) = f_quant(i_trot)
       end do

      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine trans_real
      implicit none
      include "pimd.com.f"

      double precision x_quant(2*n_trot), f_quant(2*n_trot),
     &     mat_dummy(3,3)
      double precision force_scal_ideal(n_dim+3),
     &force_scal_conf(n_dim+3)
      common /xf_quant/ x_quant, f_quant

      do i_part = 1,n_part
       i_type = type(i_part)
       do i_dim = 1,n_dim

        do i_trot = 1,2*n_trot
         x_quant(i_trot) = u0(i_dim,i_trot,i_part)/sqrtp
        end do

!-----  transform into real space
        call four1(x_quant,n_trot,-1)

        do i_trot = 0,n_trot-1
         r0(i_dim,i_trot,i_part) = x_quant(2*i_trot+1) -
     &   1. * pbc0(i_dim,i_trot,i_part) + r_ref(i_dim,i_part)
        end do

       end do
      end do

      call calc_rv_real

      v_press = 0.
      do i_dim = 1, n_dim + e_dim
       force_scal_conf(i_dim) = 0.
      end do

!---- TERM quant
      do i_trot = 0,n_trot-1
       j_trot = mod(i_trot+1,n_trot)
       do i_part = 1,n_part
        i_type = type(i_part)
        do i_dim = 1,n_dim
         force_scal_conf(i_dim) = force_scal_conf(i_dim) -
     &   k_quant(i_type) *
     &   (r0(i_dim,i_trot,i_part)+1.*pbc0(i_dim,i_trot,i_part)
     &   -r0(i_dim,j_trot,i_part)-1.*pbc0(i_dim,j_trot,i_part))*
     &   (r_real(i_dim,i_trot,i_part)-r_real(i_dim,j_trot,i_part))
        end do
       end do
       if (e_dim.eq.3) then
        do i_part = 1, n_part
         force_scal_conf(4) = force_scal_conf(4) -
     &   k_quant(i_type) * (
     &   ( r0(1,i_trot,i_part) + 1. * pbc0(1,i_trot,i_part)
     &   - r0(1,j_trot,i_part) - 1. * pbc0(1,j_trot,i_part) ) *
     &   ( r_real(2,i_trot,i_part) - r_real(2,j_trot,i_part) ) +
     &   ( r0(2,i_trot,i_part) + 1. * pbc0(2,i_trot,i_part)
     &   - r0(2,j_trot,i_part) - 1. * pbc0(2,j_trot,i_part) ) *
     &   ( r_real(1,i_trot,i_part) - r_real(1,j_trot,i_part) ) ) / 2
         force_scal_conf(5) = force_scal_conf(5) -
     &   k_quant(i_type) * (
     &   ( r0(2,i_trot,i_part) + 1. * pbc0(2,i_trot,i_part)
     &   - r0(2,j_trot,i_part) - 1. * pbc0(2,j_trot,i_part) ) *
     &   ( r_real(3,i_trot,i_part) - r_real(3,j_trot,i_part) ) +
     &   ( r0(3,i_trot,i_part) + 1. * pbc0(3,i_trot,i_part)
     &   - r0(3,j_trot,i_part) - 1. * pbc0(3,j_trot,i_part) ) *
     &   ( r_real(2,i_trot,i_part) - r_real(2,j_trot,i_part) ) ) / 2
         force_scal_conf(6) = force_scal_conf(6) -
     &   k_quant(i_type) * (
     &   ( r0(3,i_trot,i_part) + 1. * pbc0(3,i_trot,i_part)
     &   - r0(3,j_trot,i_part) - 1. * pbc0(3,j_trot,i_part) ) *
     &   ( r_real(1,i_trot,i_part) - r_real(1,j_trot,i_part) ) +
     &   ( r0(1,i_trot,i_part) + 1. * pbc0(1,i_trot,i_part)
     &   - r0(1,j_trot,i_part) - 1. * pbc0(1,j_trot,i_part) ) *
     &   ( r_real(3,i_trot,i_part) - r_real(3,j_trot,i_part) ) ) / 2
        end do
       end if
      end do

!---  term iii
      do i_dim = 1, n_dim + 3
       force_scal_ideal(i_dim) = 0.
      end do
      do i_part = 1, n_part
       i_type = type(i_part)
       do i_trot = 0, n_trot-1
```

```
!----   first part
       do i_dim = 1, n_dim
        force_scal_ideal(i_dim) = force_scal_ideal(i_dim) +
     &   mass(i_trot,i_type) * v_real(i_dim,2*i_trot+1,i_part) *
     &   ux(1,i_dim,2*i_trot+1,i_part)
       end do
       if (e_dim.eq.3) then
!---    symmetrize non diag elements
        force_scal_ideal(4) = force_scal_ideal(4) +
     &   mass(i_trot,i_type) *
     &   ( v_real(1,2*i_trot+1,i_part) * ux(1,2,2*i_trot+1,i_part)
     &   + v_real(2,2*i_trot+1,i_part) * ux(1,1,2*i_trot+1,i_part)  )
        force_scal_ideal(5) = force_scal_ideal(5) +
     &   mass(i_trot,i_type) *
     &   ( v_real(2,2*i_trot+1,i_part) * ux(1,3,2*i_trot+1,i_part)
     &   + v_real(3,2*i_trot+1,i_part) * ux(1,2,2*i_trot+1,i_part)  )
        force_scal_ideal(6) = force_scal_ideal(6) +
     &   mass(i_trot,i_type) *
     &   ( v_real(3,2*i_trot+1,i_part) * ux(1,1,2*i_trot+1,i_part)
     &   + v_real(1,2*i_trot+1,i_part) * ux(1,3,2*i_trot+1,i_part)  )
       end if
!----   second part
       if (n_trot.gt.1) then
        do i_dim = 1, n_dim
         force_scal_ideal(i_dim) = force_scal_ideal(i_dim) +
     &    mass(i_trot,i_type) * v_real(i_dim,2*i_trot+2,i_part)
     &    * ux(1,i_dim,2*i_trot+2,i_part)
        end do
        if (e_dim.eq.3) then
!---     symmetrize non diag elements
         force_scal_ideal(4) = force_scal_ideal(4) +
     &    mass(i_trot,i_type) *
     &    ( v_real(1,2*i_trot+2,i_part) * ux(1,2,2*i_trot+2,i_part)
     &    + v_real(2,2*i_trot+2,i_part) * ux(1,1,2*i_trot+2,i_part) )
         force_scal_ideal(5) = force_scal_ideal(5) +
     &    mass(i_trot,i_type) *
     &    ( v_real(2,2*i_trot+2,i_part) * ux(1,3,2*i_trot+2,i_part)
     &    + v_real(3,2*i_trot+2,i_part) * ux(1,2,2*i_trot+2,i_part) )
         force_scal_ideal(6) = force_scal_ideal(6) +
     &    mass(i_trot,i_type) *
     &    ( v_real(3,2*i_trot+2,i_part) * ux(1,1,2*i_trot+2,i_part)
     &    + v_real(1,2*i_trot+2,i_part) * ux(1,3,2*i_trot+2,i_part) )
        end if
       end if
      end do
     end do
     do i_dim = 1, n_dim + e_dim
      force_scal_ideal(i_dim) = force_scal_ideal(i_dim) / dt_2
      if (i_dim.gt.n_dim)
     & force_scal_ideal(i_dim) = force_scal_ideal(i_dim) / 2
     end do

! 666 take cutoff correction into calculation of elatic constants
      if (f_potential.ne.1) then
       do i_dim = 1, n_dim + e_dim
        force_scal(i_dim) = force_scal_ideal(i_dim)
     &  + force_scal_conf(i_dim)
       end do
!---   correction due to cutoff of r**(-6) potential
       v_press = v_press
     & - n_trot * n_part**2 * for_sca_mfa_cor / volume / 2
       do i_dim = 1, n_dim
        if (i_dim.le.n_dim) then
         r_dummy = h_mat_inv(i_dim,i_dim)
        else if (i_dim.eq.4) then
         r_dummy = ( h_mat_inv(1,2) + h_mat_inv(2,1) ) / 2
        else if (i_dim.eq.5) then
         r_dummy = ( h_mat_inv(2,3) + h_mat_inv(3,2) ) / 2
        else
         r_dummy = ( h_mat_inv(3,1) + h_mat_inv(1,3) ) / 2
        end if
        r_dummy = - n_part**2 * for_sca_mfa_cor * r_dummy / volume
        force_scal(i_dim) = force_scal(i_dim) + r_dummy
       end do
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine calc_rv_real
      implicit none
      include "pimd.com.f"

      do i_part = 1,n_part
      do i_trot = 0,n_trot-1
       do i_dim = 1, n_dim
        r_real(i_dim,i_trot,i_part) = scal_0(i_dim) *
     &  (r0(i_dim,i_trot,i_part) + 1.d0*pbc0(i_dim,i_trot,i_part))
        v_real(i_dim,2*i_trot+1,i_part) =
     &  scal_0(i_dim) * ux(1,i_dim,2*i_trot+1,i_part)
        if (n_trot.gt.1) v_real(i_dim,2*i_trot+2,i_part) =
     &  scal_0(i_dim) * ux(1,i_dim,2*i_trot+2,i_part)
       end do
       if (e_dim.eq.3) then
        r_real(1,i_trot,i_part) = r_real(1,i_trot,i_part) +
     &  scal_0(4) * (r0(2,i_trot,i_part) + 1.d0*pbc0(2,i_trot,i_part)) +
     &  scal_0(6) * (r0(3,i_trot,i_part) + 1.d0*pbc0(3,i_trot,i_part))
        r_real(2,i_trot,i_part) = r_real(2,i_trot,i_part) +
     &  scal_0(4) * (r0(1,i_trot,i_part) + 1.d0*pbc0(1,i_trot,i_part)) +
     &  scal_0(5) * (r0(3,i_trot,i_part) + 1.d0*pbc0(3,i_trot,i_part))
        r_real(3,i_trot,i_part) = r_real(3,i_trot,i_part) +
     &  scal_0(6) * (r0(1,i_trot,i_part) + 1.d0*pbc0(1,i_trot,i_part)) +
     &  scal_0(5) * (r0(2,i_trot,i_part) + 1.d0*pbc0(2,i_trot,i_part))
        v_real(1,2*i_trot+1,i_part) = v_real(1,2*i_trot+1,i_part) +
     &  scal_0(4) * ux(1,2,2*i_trot+1,i_part) +
     &  scal_0(6) * ux(1,3,2*i_trot+1,i_part)
        v_real(2,2*i_trot+1,i_part) = v_real(2,2*i_trot+1,i_part) +
     &  scal_0(4) * ux(1,1,2*i_trot+1,i_part) +
     &  scal_0(5) * ux(1,3,2*i_trot+1,i_part)
        v_real(3,2*i_trot+1,i_part) = v_real(3,2*i_trot+1,i_part) +
     &  scal_0(6) * ux(1,1,2*i_trot+1,i_part) +
     &  scal_0(5) * ux(1,2,2*i_trot+1,i_part)
        if (n_trot.gt.1) then
         v_real(1,2*i_trot+2,i_part) = v_real(1,2*i_trot+2,i_part) +
     &   scal_0(4) * ux(1,2,2*i_trot+2,i_part) +
     &   scal_0(6) * ux(1,3,2*i_trot+2,i_part)
         v_real(2,2*i_trot+2,i_part) = v_real(2,2*i_trot+2,i_part) +
     &   scal_0(4) * ux(1,1,2*i_trot+2,i_part) +
     &   scal_0(5) * ux(1,3,2*i_trot+2,i_part)
         v_real(3,2*i_trot+2,i_part) = v_real(3,2*i_trot+2,i_part) +
     &   scal_0(6) * ux(1,1,2*i_trot+2,i_part) +
     &   scal_0(5) * ux(1,2,2*i_trot+2,i_part)
        end if
```

```
       end if
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine binning
      implicit none
      include "pimd.com.f"

      integer i_bin,j_bin

      integer i_bin_part,ni_bin_part
      integer j_bin_part,nj_bin_part

      integer i_bin_x,i_bin_y,i_bin_z
      integer j_bin_x,j_bin_y,j_bin_z
      integer dj_bin_x,dj_bin_y,dj_bin_z

      double precision r_2

      do i_part = 1,n_part
       neighbor(0,i_part) = 0
      end do

      do i_part = 1,n_part-1
       i_type = type(i_part)
       do j_part = i_part+1,n_part
        j_type = type(j_part)
        do i_trot = 0,n_trot-1
!----    calculate distance with right boundaries
         call calc_distance(r_2,i_trot,i_part,j_part)
         if(r_2.le.r_range_2(i_type,j_type)) then
          if(delta_r(1).lt.0.) then
           neighbor(0,i_part) = neighbor(0,i_part) + 1
           i_dummy = neighbor(0,i_part)
           neighbor(i_dummy,i_part) = j_part
          else
           neighbor(0,j_part) = neighbor(0,j_part) + 1
           if (neighbor(0,j_part).gt.n_neigh) stop "neighbor field"
           i_dummy = neighbor(0,j_part)
           neighbor(i_dummy,j_part) = i_part
          end if
          n_neigh_max = max(n_neigh_max,neighbor(0,j_part))
          n_neigh_max = max(n_neigh_max,neighbor(0,i_part))
          goto 150
         end if
        end do
  150   continue
       end do
      end do

      if(n_neigh_max.gt.n_neigh) stop 'make neighbor field larger'

      f_skin = 0
      do i_part = 1,n_part
      do i_trot = 0,n_trot-1
      do i_dim = 1,n_dim
       r0_old(i_dim,i_trot,i_part) = r_real(i_dim,i_trot,i_part)
      end do
      end do
      end do
      end do

! 666
      if (f_potential.eq.3) call rec_binning
      if (f_potential.eq.5) call rec_binning
      if (f_potential.eq.6) call rec_binning
      if (f_potential.eq.7) call rec_binning

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine rec_binning
      implicit none
      include "pimd.com.f"
      double precision q_vec(n_dim),q_2,loc_prefac,pot_loc
      complex*16 f_of_k

      rec_pot_offset = 0.
      do i_dim = 1, n_dim + e_dim
       rec_force_offset(i_dim) = 0.
      end do
      loc_prefac = elec_prefac*twopi/volume

      n_rec_vec = 0
      do k_dim = 0, nq_max(3)
       do j_dim = -nq_max(2), nq_max(2)
        do i_dim = -nq_max(1), nq_max(1)
         q_vec(1) = i_dim * dq(1,1) + j_dim * dq(1,2) + k_dim * dq(1,3)
         q_vec(2) = i_dim * dq(2,1) + j_dim * dq(2,2) + k_dim * dq(2,3)
         q_vec(3) = i_dim * dq(3,1) + j_dim * dq(3,2) + k_dim * dq(3,3)
         q_2 = 0.
         do l_dim = 1,n_dim
          q_2 = q_2 + q_vec(l_dim)**2
         end do
         if((q_2.le.q_cut_2).and.(q_2.gt.0.)) then
!-----    calculate FT of charge density
          f_of_k = (0.,0.)
          do i_part = 1,n_part
!---       opt
           r_dummy =
     &     i_dim*(r_ref(1,i_part)+u0(1,1,i_part)/sqrtp) +
     &     j_dim*(r_ref(2,i_part)+u0(2,1,i_part)/sqrtp) +
     &     k_dim*(r_ref(3,i_part)+u0(3,1,i_part)/sqrtp)
           r_dummy = dmod(r_dummy,1.d0)
           i_dummy = nint(r_dummy*n_potential)
           f_of_k = f_of_k + charge(i_part)*
!     &     tab_exp_ima(i_dummy)
     &     exp(2.*pi*(0.,1.)*r_dummy)
          end do
          r_dummy = loc_prefac*exp(-q_2/(2*alpha)**2)/q_2
          pot_loc = r_dummy * abs(f_of_k)**2
!         write(30+i_time,*) pot_loc,(q_vec(l_dim),l_dim=1,n_dim)
          if(pot_loc.ge.v_cut) then
           n_rec_vec = n_rec_vec + 1
           if(n_rec_vec.gt.n_rec_vec_max) stop "n_rec_vec too big"
           rec_vec_list(n_rec_vec,1) = i_dim
           rec_vec_list(n_rec_vec,2) = j_dim
           rec_vec_list(n_rec_vec,3) = k_dim
          else
           if(k_dim.ne.0) r_dummy = r_dummy*2
```

```
      pot_loc = r_dummy*abs(f_of_k)**2
      rec_pot_offset = rec_pot_offset + pot_loc
      if(f_pressure.eq.1) then
        r_dummy = (1./(2*alpha**2)+2./q_2)*pot_loc
        do l_dim = 1,n_dim
          rec_force_offset(l_dim) = rec_force_offset(l_dim) +
     &      pot_loc - r_dummy*q_vec(l_dim)**2
        end do
      end if
    end if
  end do
 end do

!---- change
 do i_dim = 1,n_dim
   rec_force_offset(i_dim) = n_trot*rec_force_offset(i_dim)
 end do
 rec_pot_offset = n_trot*rec_pot_offset

 end subroutine

! ! ! ! ! ! ! ! ! ! !


      subroutine rec_ewald
      implicit none
      include "pimd.com_f"
      integer i_q(n_dim), trot_max, k_trot
      double precision q_vec(n_dim),q_2,loc_prefac,rec_prefac
      double precision two_rec_prefac,v_mode,force_scal_rec(n_dim)
      complex*16 qeqreck(n_part)
      complex*16 f_of_k
      double precision f_of_k_real, f_of_k_imag

      do i_part=1, n_part
        qeqrec(i_part)=0.
      end do

      v_ewald = 0.
      do i_dim = 1,n_dim
       force_scal_rec(i_dim) = 0.
      end do
      loc_prefac = elec_prefac*twopi/volume

      j_trot = mod(int(n_trot*random(1)),n_trot)
      do i_rec_vec = 1,n_rec_vec
        q_2 = 0.
        do i_dim = 1,n_dim
          q_vec(i_dim) = 0.
          do j_dim = 1, n_dim
            i_q(j_dim) = rec_vec_list(i_rec_vec,j_dim)
            q_vec(i_dim) = q_vec(i_dim) + i_q(j_dim)*dq(j_dim,i_dim)
          end do
          q_2 = q_2 + q_vec(i_dim)**2
        end do
        rec_prefac = loc_prefac*exp(-q_2/(2*alpha)**2)/q_2
        if(rec_vec_list(i_rec_vec,3).ne.0) rec_prefac = 2 * rec_prefac
        two_rec_prefac = 2*rec_prefac
        trot_max = n_trot-1
        if (f_opt_rec.eq.1) trot_max = 0
        do i_trot = 0, trot_max
          if (f_opt_rec.eq.0) j_trot = i_trot
          f_of_k = (0.,0.)
          f_of_k_real = 0.d0
          f_of_k_imag = 0.d0
          do i_part = 1,n_part
            r_dummy = 0.
            if (f_opt_rec.eq.0) then
              do i_dim = 1,n_dim
                r_dummy = r_dummy + i_q(i_dim)*r0(i_dim,j_trot,i_part)
              end do
            else
              do i_dim = 1, n_dim
                r_dummy = r_dummy + i_q(i_dim) *
     &            (r_ref(i_dim,i_part) + u0(i_dim,1,i_part)/sqrtp )
              end do
            end if
            r_dummy = dmod(r_dummy,1.d0)
            i_dummy = nint(r_dummy*n_potential)
            f_of_k = f_of_k + charge(i_part)*
     !        tab_exp_ima(i_dummy)
     &      exp(2.*pi*(0.,1.)*r_dummy)
            f_of_k_real = f_of_k_real+ charge(i_part)*
     &        cos(2.*pi*r_dummy)
            f_of_k_imag = f_of_k_imag + charge(i_part)*
     &          sin(2.*pi*r_dummy)

          end do

          v_mode = rec_prefac * abs(f_of_k)**2

          if (f_opt_rec.eq.1) v_mode = n_trot * v_mode
          v_ewald = v_ewald + v_mode
!---------  calculate force from mode "i_rec_vec" on each atom
          do i_part = 1,n_part
            r_dummy = 0.
            do i_dim = 1,n_dim
              r_dummy = r_dummy + i_q(i_dim)*r0(i_dim,j_trot,i_part)
            end do
            r_dummy = -dmod(r_dummy,1.d0)
            i_dummy = nint(r_dummy*n_potential)
            c_dummy = tab_exp_ima(i_dummy)
!           c_dummy = exp(2.*pi*(0.,1.)*r_dummy)
!           qeqrec(i_part) = qeqrec(i_part) +
!    &        two_rec_prefac*dble(c_dummy*f_of_k)
            qeqrec(i_part) = qeqrec(i_part) + two_rec_prefac *
     &          ( cos(2.*pi*r_dummy)*f_of_k_real
     &            - sin(2.*pi*r_dummy)*f_of_k_imag )
            r_dummy = imag(c_dummy*f_of_k)

            if (f_opt_rec.eq.0) then
              do i_dim = 1,n_dim
                force(i_dim,i_trot,i_part) = force(i_dim,i_trot,i_part)
     &            - two_rec_prefac*charge(i_part)*q_vec(i_dim)*r_dummy
     &            * h_mat_inv(i_dim,i_dim)
              end do
            else
              do k_trot = n_trot-1,0,-1
              do i_dim = 1,n_dim
                force(i_dim,k_trot,i_part) = force(i_dim,k_trot,i_part)
     &            - two_rec_prefac*charge(i_part)*q_vec(i_dim)*r_dummy
     &            * h_mat_inv(i_dim,i_dim)
              end do
```

```
          end do
         end if
        end do
!----   calculate force from each mode "i_rec_vec" on box shape
        if(f_pressure.eq.1) then
         r_dummy = (1./(2*alpha**2)+2./q_2)*v_mode
         do i_dim = 1,n_dim
          force_scal_rec(i_dim) = force_scal_rec(i_dim) +
     &    (v_mode - r_dummy*q_vec(i_dim)**2)
         end do
        end if
       end do

      end do


      v_ewald = v_ewald + rec_pot_offset

      if(f_pressure.eq.1) then
       do i_dim = 1,n_dim
        force_scal_rec(i_dim) = force_scal_rec(i_dim) +
     &  rec_force_offset(i_dim)
        do j_dim = 1, n_dim
         force_scal(i_dim) = force_scal(i_dim) +
     &   h_mat_inv(i_dim,j_dim) * force_scal_rec(j_dim)
        end do
       end do
      end if

      v_inter = v_inter + v_ewald

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_distance(r_2,i_trot,i_part,j_part)
      implicit none
      include "pimd.com.f"
      double precision r_2

      do i_dim = 1,n_dim
       delta_cell(i_dim) =
     & r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
       if (delta_cell(i_dim).gt.0.5) then
        delta_cell(i_dim) = delta_cell(i_dim) - 1.
       else if(delta_cell(i_dim).lt.-0.5) then
        delta_cell(i_dim) = delta_cell(i_dim) + 1.
       end if
      end do
!---- term square
      r_2 = 0.
      do i_dim = 1, n_dim
       r_2 = r_2 + met_ten_voi(i_dim) * delta_cell(i_dim)**2
      end do
      if (e_dim.eq.3) then
       r_2 = r_2
     & + met_ten_voi(4) * delta_cell(1) * delta_cell(2)
     & + met_ten_voi(5) * delta_cell(2) * delta_cell(3)
     & + met_ten_voi(6) * delta_cell(3) * delta_cell(1)
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_dist_pbc(r_2,i_trot,i_part,j_part,
     &     pbc_x,pbc_y,pbc_z)
      implicit none
      include "pimd.com.f"
      double precision r_2
      integer pbc_x,pbc_y,pbc_z,pbc(n_dim)

      pbc(1)=pbc_x
      pbc(2)=pbc_y
      pbc(3)=pbc_z

      do i_dim = 1,n_dim
       delta_cell(i_dim) =
     & r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
       if (delta_cell(i_dim).gt.0.5) then
        delta_cell(i_dim) = delta_cell(i_dim) - 1.
       else if(delta_cell(i_dim).lt.-0.5) then
        delta_cell(i_dim) = delta_cell(i_dim) + 1.
       end if
      end do
      do i_dim = 1,n_dim
         delta_cell(i_dim) = delta_cell(i_dim) +
     &        1.0*pbc(i_dim)
      end do
!---- term square
      r_2 = 0.
      do i_dim = 1, n_dim
       r_2 = r_2 + met_ten_voi(i_dim) * delta_cell(i_dim)**2
      end do
      if (e_dim.eq.3) then
       r_2 = r_2
     & + met_ten_voi(4) * delta_cell(1) * delta_cell(2)
     & + met_ten_voi(5) * delta_cell(2) * delta_cell(3)
     & + met_ten_voi(6) * delta_cell(3) * delta_cell(1)
      end if

      end subroutine


! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_dist_int(r_2)
      implicit none
      include "pimd.com.f"
      double precision r_2
!---- term square
      do i_dim = 1,n_dim
       delta_r(i_dim) = scal_0(i_dim) * delta_cell(i_dim)
       r_2 = r_2 + met_ten_voi(i_dim) * delta_cell(i_dim)**2
      end do
      if (e_dim.eq.3) then
       r_2 = r_2
     & + met_ten_voi(4) * delta_cell(1)*delta_cell(2)
     & + met_ten_voi(5) * delta_cell(2)*delta_cell(3)
     & + met_ten_voi(6) * delta_cell(3)*delta_cell(1)
      end if
      end subroutine

! ! ! ! ! ! ! ! ! ! ! !
```

```
      subroutine harm_osc
      implicit none
      include "pimd.com.f"
      double precision v_inter_loc
      double precision r_1,r_2

      v_inter = 0.
!---- harmonic oscillator
      do i_part = 1,n_part
       v_inter_loc = 0.
       do i_dim = 1,n_dim
       do i_trot = 0,n_trot-1
         r_dummy = r0(i_dim,i_trot,i_part)
        force(i_dim,i_trot,i_part) = force(i_dim,i_trot,i_part) -
     &  r_dummy
        v_inter_loc = v_inter_loc + r_dummy**2
       end do
       end do
       v_inter = v_inter + v_inter_loc
      end do
      v_inter = v_inter/2

      viri_est = v_inter

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine box_chain
      implicit none
      include "pimd.com.f"

!---  term coupl h_dot and velocities
      do i_part = 1,n_part
       i_type = type(i_part)
       do j_trot = 0,n_trot-1
        i_trot = j_trot + 1
        if(i_trot.eq.2) i_trot = n_trot + 1
        i_dummy = (i_trot-1)/2
        do i_dim = 1,n_dim
         f0(i_dim,i_trot,i_part) = f0(i_dim,i_trot,i_part) -
     &   g_voi_dot(i_dim) * mass(i_dummy,i_type) *
     &   ux(1,i_dim,i_trot,i_part) / dt_2
        end do
        if (e_dim.eq.3) then
         f0(1,i_trot,i_part) = f0(1,i_trot,i_part) - (
     &   g_voi_dot(4)*mass(i_dummy,i_type)*ux(1,2,i_trot,i_part) +
     &   g_voi_dot(6)*mass(i_dummy,i_type)*ux(1,3,i_trot,i_part) ) /dt_2
     &   / 2
         f0(2,i_trot,i_part) = f0(2,i_trot,i_part) - (
     &   g_voi_dot(5)*mass(i_dummy,i_type)*ux(1,3,i_trot,i_part) +
     &   g_voi_dot(4)*mass(i_dummy,i_type)*ux(1,1,i_trot,i_part) ) /dt_2
     &   / 2
         f0(3,i_trot,i_part) = f0(3,i_trot,i_part) - (
     &   g_voi_dot(6)*mass(i_dummy,i_type)*ux(1,1,i_trot,i_part) +
     &   g_voi_dot(5)*mass(i_dummy,i_type)*ux(1,2,i_trot,i_part) ) /dt_2
     &   / 2
        end if
       end do
      end do

!---- symmetrize
      do i_trot = 1,n_trot/2-1
      do i_dim = 1,n_dim
      do i_part = 1,n_part
       f0(i_dim,2*(n_trot-i_trot)+1,i_part) =
     & f0(i_dim,2*i_trot+1,i_part)
      end do
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine lennard_jones
      implicit none
      include "pimd.com.f"
      double precision v_inter_loc
      double precision r_2, r_6, r_12, pot_loc, r_4_inv
      double precision dyn_mat(n_dim,n_dim), loc_dyn_mat(n_dim,n_dim)

      v_inter = 0.
      do i_dim = 1, n_dim+3
       force_scal_inter(i_dim) = 0.
      end do

      if ( (i_time.gt.n_relax) .and. (mod(i_time,20).eq.0) ) then
       do i_dim = 1, n_dim
       do j_dim = i_dim, n_dim
        do k_dim = i_dim ,n_dim
        do l_dim = k_dim ,n_dim
         if (i_dim+j_dim.le.k_dim+l_dim) then
          ela_born(i_dim,j_dim,k_dim,l_dim) = 0.
         end if
        end do
        end do
       end do
       end do
       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
        strain(i_dim,j_dim) = 0.
       end do
       end do
      end if

      do i_part = 1,n_part
       i_type = type(i_part)
       v_inter_loc = 0.

       do i_neigh = 1,neighbor(0,i_part)
        j_part = neighbor(i_neigh,i_part)
        j_type = type(j_part)
        pot_loc = 0.
        do i_trot = 0,n_trot-1

         call calc_distance(r_2,i_trot,i_part,j_part)

         if(r_2.lt.r_cutoff_2(i_type,j_type)) then
          r_6 = (sigma_2(i_type,j_type)/r_2)**3
          r_12 = r_6**2
          pot_loc = pot_loc + (r_12-r_6) - e_shift(i_type,j_type)
!----     r_dummy = {\partial V \over \partial r} / r
```

```
          r_dummy = epsil(i_type,j_type)*(-12*r_12+6*r_6)/r_2

          call calc_force_loc(r_dummy,i_trot,i_part,j_part)

          if ( (i_time.gt.n_relax) .and. (mod(i_time,20).eq.0) ) then
!----      calculate born elastic constants
          r_4_inv = 1. / r_2**2
          call loc_dynmat(loc_dyn_mat, delta_r, r_2, -12.d0)
          do i_dim = 1, n_dim
          do j_dim = 1, n_dim
           dyn_mat(i_dim,j_dim) = sigma_2(i_type,j_type)**6 *
     &     loc_dyn_mat(i_dim,j_dim)
          end do
          end do
          call loc_dynmat(loc_dyn_mat, delta_r, r_2, -6.d0)
          do i_dim = 1, n_dim
          do j_dim = 1, n_dim
           dyn_mat(i_dim,j_dim) = epsil(i_type,j_type) * r_4_inv *
     &     ( dyn_mat(i_dim,j_dim) -
     &       sigma_2(i_type,j_type)**3 * loc_dyn_mat(i_dim,j_dim) )
          end do
          end do
          do i_dim = 1, n_dim
          do j_dim = i_dim, n_dim
          do k_dim = i_dim, n_dim
          do l_dim = k_dim, n_dim
           if (i_dim+j_dim.le.k_dim+l_dim) then
            ela_born(i_dim, j_dim, k_dim, l_dim) =
     &      ela_born(i_dim, j_dim, k_dim, l_dim) + delta_r(i_dim) *
     &      dyn_mat(j_dim,k_dim) * delta_r(l_dim)
           end if
          end do
          end do
          end do
          end do
!----      calculate strain tensor
          do i_dim = 1, n_dim
          do j_dim = 1, n_dim
           strain(i_dim,j_dim) = strain(i_dim,j_dim) +
     &     r_dummy * delta_r(i_dim) * delta_r(j_dim)
          end do
          end do
         end if

        end if

       end do
       v_inter_loc = v_inter_loc + epsil(i_type,j_type)*pot_loc
      end do

      v_inter = v_inter + v_inter_loc
     end do

     call force_on_scal

     if ( (i_time.gt.n_relax) .and. (mod(i_time,20).eq.0) ) then
      time_born = time_born + dt
      do i_dim = 1, n_dim
      do j_dim = i_dim, n_dim
       do k_dim = i_dim, n_dim
       do l_dim = k_dim, n_dim
        if (i_dim+j_dim.le.k_dim+l_dim) then
          ela_born_1(i_dim, j_dim, k_dim, l_dim) =
     &    ela_born_1(i_dim, j_dim, k_dim, l_dim) + dt *
     &    ela_born(i_dim, j_dim, k_dim, l_dim) / (volume*n_trot)
         end if
        end do
        end do
       end do
       end do
       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
        strain_1(i_dim,j_dim) = strain_1(i_dim,j_dim)
     &  + strain(i_dim,j_dim) * dt
        do k_dim = i_dim, n_dim
        do l_dim = k_dim, n_dim
         strain_2(i_dim, j_dim, k_dim, l_dim) =
     &   strain_2(i_dim, j_dim, k_dim, l_dim) +
     &   strain(i_dim, j_dim) * strain(k_dim, l_dim) * dt
        end do
        end do
       end do
       end do
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine loc_dynmat(dyn_mat, dr, r_2, n)
       implicit none
       integer i_dim, j_dim, n_dim
       parameter (n_dim=3)
       double precision dr(n_dim), r_2, r_n, n
       double precision dyn_mat(n_dim,n_dim), r_dummy

       if (n.ge.0.) then
        r_n = r_2**(0.5*n)
       else
        r_n = 1. / r_2**(-0.5*n)
       end if

       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
        if(i_dim.eq.j_dim) r_dummy = n * r_2
        if(i_dim.ne.j_dim) r_dummy = 0.
        dyn_mat(i_dim,j_dim) = r_dummy
     &  + n*(n-2) * dr(i_dim) * dr(j_dim)
        dyn_mat(i_dim,j_dim) = dyn_mat(i_dim,j_dim) * r_n
       end do
       end do

       end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine calc_force_loc(r_dummy,i_trot,i_part,j_part)
      implicit none
      include "pimd.com.f"

      do i_dim = 1, n_dim
       delta_r(i_dim) = scal_0(i_dim)*delta_cell(i_dim)
      end do
      if (e_dim.eq.3) then
```

```
        delta_r(1) = delta_r(1)
     & + scal_0(4)*delta_cell(2) + scal_0(6)*delta_cell(3)
        delta_r(2) = delta_r(2)
     & + scal_0(5)*delta_cell(3) + scal_0(4)*delta_cell(1)
        delta_r(3) = delta_r(3)
     & + scal_0(6)*delta_cell(1) + scal_0(5)*delta_cell(2)
       end if

       do i_dim = 1, n_dim
!----  term ii
        force_loc(i_dim) = r_dummy * delta_cell(i_dim)
        force(i_dim,i_trot,i_part) = - force_loc(i_dim) +
     & force(i_dim,i_trot,i_part)
        force(i_dim,i_trot,j_part) = + force_loc(i_dim) +
     & force(i_dim,i_trot,j_part)
        force_scal_inter(i_dim) = force_scal_inter(i_dim) -
     & force_loc(i_dim) * delta_r(i_dim)
       end do
!---- term iv
       if (e_dim.eq.3) then
!---   symmetrized "stress" field
        force_scal_inter(4) = force_scal_inter(4) -
     & ( force_loc(1)*delta_r(2) + force_loc(2)*delta_r(1) ) / 2
        force_scal_inter(5) = force_scal_inter(5) -
     & ( force_loc(2)*delta_r(3) + force_loc(3)*delta_r(2) ) / 2
        force_scal_inter(6) = force_scal_inter(6) -
     & ( force_loc(3)*delta_r(1) + force_loc(1)*delta_r(3) ) / 2
       end if
       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine force_on_scal
       implicit none
       include "pimd.com.f"

!---- change transform
       do i_dim = 1,n_dim + e_dim
        force_scal(i_dim) = force_scal(i_dim) + force_scal_inter(i_dim)
       end do

!---  add external pressure
!---  term press
       v_press = v_press + n_trot * press * volume
       do i_dim = 1, n_dim
        force_scal(i_dim) = force_scal(i_dim) -
     & n_trot * press *volume * h_mat_inv(i_dim,i_dim)
       end do
       if (e_dim.eq.3) then
        force_scal(4) = force_scal(4) -
     & n_trot * press *volume * h_mat_inv(1,2)
        force_scal(5) = force_scal(5) -
     & n_trot * press *volume * h_mat_inv(2,3)
        force_scal(6) = force_scal(6) -
     & n_trot * press *volume * h_mat_inv(3,1)
       end if

       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine calc_3body_force_loc(fi,fj,fk,
     &     i_trot,i_part,j_part,k_part)
       implicit none
       include "pimd.com.f"
       double precision fi,fj,fk
       integer k_part

       do i_dim = 1, n_dim
        delta_r(i_dim) = scal_0(i_dim)*delta_cell(i_dim)
       end do
       if (e_dim.eq.3) then
        delta_r(1) = delta_r(1)
     & + scal_0(4)*delta_cell(2) + scal_0(6)*delta_cell(3)
        delta_r(2) = delta_r(2)
     & + scal_0(5)*delta_cell(3) + scal_0(4)*delta_cell(1)
        delta_r(3) = delta_r(3)
     & + scal_0(6)*delta_cell(1) + scal_0(5)*delta_cell(2)
       end if

       do i_dim = 1, n_dim
!----  term ii
       force(i_dim,i_trot,i_part) = + fi * delta_cell(i_dim) +
     &      force(i_dim,i_trot,i_part)
       force(i_dim,i_trot,j_part) = + fj * delta_cell(i_dim) +
     &      force(i_dim,i_trot,j_part)
       force(i_dim,i_trot,k_part) = + fk * delta_cell(i_dim) +
     &      force(i_dim,i_trot,k_part)
       force_loc(i_dim) = fj * delta_cell(i_dim)
       force_scal_inter(i_dim) = force_scal_inter(i_dim) -
     &      force_loc(i_dim) * delta_r(i_dim)
       end do
!---- term iv
       if (e_dim.eq.3) then
!---   symmetrized "stress" field
       force_scal_inter(4) = force_scal_inter(4) -
     & ( force_loc(1)*delta_r(2) + force_loc(2)*delta_r(1) ) / 2
       force_scal_inter(5) = force_scal_inter(5) -
     & ( force_loc(2)*delta_r(3) + force_loc(3)*delta_r(2) ) / 2
       force_scal_inter(6) = force_scal_inter(6) -
     & ( force_loc(3)*delta_r(1) + force_loc(1)*delta_r(3) ) / 2
       end if

       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine goddard_potential
       implicit none
       include "pimd.com.f"
       call twobody_potential
       call force_on_scal
       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine vashishta_potential
       implicit none
       include "pimd.com.f"
       integer i_ang, k_part
       double precision cosjik, rij, rik, fij, fik, fjj, fjk, fkj, fkk
       double precision factor1, term1, termj, termk,
     &     r_ij(n_dim), r_ik(n_dim),
     &     delta_ij(n_dim), delta_ik(n_dim)
```

```
      call twobody_potential
! three body contributions: j-i-k
      call bond_analysis
      v_threebody= 0.d0

      do i_part=1, n_part
         i_type=type(i_part)
         do i_ang=1, n_ang(i_type)
            j_part=triple_partj(i_part,i_ang)
            k_part=triple_partk(i_part,i_ang)
            cosjik=triple_cos(i_part,i_ang)
            rij=triple_rij(i_part,i_ang)
            rik=triple_rik(i_part,i_ang)

            if ((rij.ge.vash_r0(i_type)).or.(rik.ge.vash_r0(i_type)))
     &      then
!              v_threebody==0: add no forces, no energies
            else ! only if rij and rik < r0

            factor1=1/(rij-vash_r0(i_type))+1/(rik-vash_r0(i_type))
            factor1=exp(vash_l(i_type)*factor1)
            factor1=factor1*(cosjik-vash_ctheta(i_type))
            factor1=vash_B(i_type)*factor1
            factor1=factor1*elec_prefac

            termj=vash_l(i_type)*(cosjik-vash_ctheta(i_type))
            termk=termj
            termj=termj/(rij*(rij-vash_r0(i_type))**2)
            termj=2*cosjik/(rij**2)+termj

            termk=termk/(rik*(rik-vash_r0(i_type))**2)
            termk=2*cosjik/(rik**2)+termk

            term1=2/(rik*rij)

!            write(*,*) i_part,i_type,i_ang,j_part,k_part,
!     &           180*acos(cosjik)/pi,rij,rik

! calculate forces in ij direction
            do i_dim=1, n_dim
               delta_ij(i_dim)=triple_deltaij(i_part,i_ang,i_dim)
            end do
            fij=-factor1*(term1-termj)
            fjj=-factor1*termj
            fkj=+factor1*term1
!            call calc_3body_force_loc(fi,fj,fk,0,i_part,j_part,k_part)

! calculate forces in ik direction
            do i_dim=1, n_dim
               delta_ik(i_dim)=triple_deltaik(i_part,i_ang,i_dim)
            end do
            fik=-factor1*(term1-termk)
            fjk=+factor1*term1
            fkk=-factor1*termk
!            call calc_3body_force_loc(fi,fj,fk,0,i_part,j_part,k_part)

      do i_dim = 1, n_dim
         r_ij(i_dim) = scal_0(i_dim)*delta_ij(i_dim)
      end do
      if (e_dim.eq.3) then
       r_ij(1) = r_ij(1)
     & + scal_0(4)*delta_ij(2) + scal_0(6)*delta_ij(3)
       r_ij(2) = r_ij(2)
     & + scal_0(5)*delta_ij(3) + scal_0(4)*delta_ij(1)
       r_ij(3) = r_ij(3)
     & + scal_0(6)*delta_ij(1) + scal_0(5)*delta_ij(2)
      end if

      do i_dim = 1, n_dim
         r_ik(i_dim) = scal_0(i_dim)*delta_ik(i_dim)
      end do
      if (e_dim.eq.3) then
       r_ik(1) = r_ik(1)
     & + scal_0(4)*delta_ik(2) + scal_0(6)*delta_ik(3)
       r_ik(2) = r_ik(2)
     & + scal_0(5)*delta_ik(3) + scal_0(4)*delta_ik(1)
       r_ik(3) = r_ik(3)
     & + scal_0(6)*delta_ik(1) + scal_0(5)*delta_ik(2)
      end if

      i_trot=0

      do i_dim = 1, n_dim
!----  term ii
       force(i_dim,i_trot,i_part) = + fij * delta_ij(i_dim) +
     &      fik * delta_ik(i_dim) +
     &      force(i_dim,i_trot,i_part)
       force(i_dim,i_trot,j_part) = + fjj * delta_ij(i_dim) +
     &      fjk * delta_ik(i_dim) +
     &      force(i_dim,i_trot,j_part)
       force(i_dim,i_trot,k_part) = + fkj * delta_ij(i_dim) +
     &      fkk * delta_ik(i_dim) +
     &      force(i_dim,i_trot,k_part)

       force_scal_inter(i_dim) = force_scal_inter(i_dim)
     &      - fjj * delta_ij(i_dim) * r_ij(i_dim)
     &      - fjk * delta_ik(i_dim) * r_ij(i_dim)
     &      - fkj * delta_ij(i_dim) * r_ik(i_dim)
     &      - fkk * delta_ik(i_dim) * r_ik(i_dim)
      end do
!---- term iv
      if (e_dim.eq.3) then
!---   symmetrized "stress" field
       force_scal_inter(4) = force_scal_inter(4) -
     & ( fjj * delta_ij(1)*r_ij(2) + fjj * delta_ij(2)*r_ij(1) ) / 2 -
     & ( fjk * delta_ik(1)*r_ij(2) + fjk * delta_ik(2)*r_ij(1) ) / 2 -
     & ( fkj * delta_ij(1)*r_ik(2) + fkj * delta_ij(2)*r_ik(1) ) / 2 -
     & ( fkk * delta_ik(1)*r_ik(2) + fkk * delta_ik(2)*r_ik(1) ) / 2
       force_scal_inter(5) = force_scal_inter(5) -
     & ( fjj * delta_ij(2)*r_ij(3) + fjj * delta_ij(3)*r_ij(2) ) / 2 -
     & ( fjk * delta_ik(2)*r_ij(3) + fjk * delta_ik(3)*r_ij(2) ) / 2 -
     & ( fkj * delta_ij(2)*r_ik(3) + fkj * delta_ij(3)*r_ik(2) ) / 2 -
     & ( fkk * delta_ik(2)*r_ik(3) + fkk * delta_ik(3)*r_ik(2) ) / 2
       force_scal_inter(6) = force_scal_inter(6) -
     & ( fjj * delta_ij(3)*r_ij(1) + fjj * delta_ij(1)*r_ij(3) ) / 2 -
     & ( fjk * delta_ij(3)*r_ij(1) + fjk * delta_ik(1)*r_ij(3) ) / 2 -
     & ( fkj * delta_ij(3)*r_ik(1) + fkj * delta_ij(1)*r_ik(3) ) / 2 -
     & ( fkk * delta_ik(3)*r_ik(1) + fkk * delta_ik(1)*r_ik(3) ) / 2
      end if

            v_threebody = v_threebody
     &           + factor1*(cosjik - vash_ctheta(i_type))

         end if ! only if rij and rik < r0
```

```
            end do
         end do

         v_inter = v_inter + v_threebody

         call force_on_scal
         end subroutine

! ! ! ! ! ! ! ! ! ! ! ! ! !

         subroutine scandolo_potential
c
           use metric
           use neighbour
           use polar ! to access dipoles
           implicit double precision(a-h,o-z)
         include "pimd.com2.f"
c
           character*25 filepos,filecel
           real*8, allocatable :: r(:,:)
           real*8, allocatable :: sforce(:,:)
           real*8 stress(3,3),htin(3,3)
           logical restart,tcel
           integer n_part_type(n_type)
           integer i_class


! assign the variables
           n_relax_sc = n_relax
           nat=n_part
           nsp=n_type
           A_ew=1.0d-8
           rcut=18.02703
           nnatmax=n_neigh
           iesr=n_pbc  ! search cell and neighboring image cells

c
         allocate(sforce(3,nat))
         allocate(r(3,nat))
         allocate(spind(nat))
c
            do i=1,nat
               do j=1, 3
                  r(j,i) = r_real(j,0,i)
                  spind(i) = type(i)
               end do
            enddo

            call trans_to_tens(scal_0, ht)
c
            r = r/0.5292d0
            ht = ht/0.5292d0
c
         allocate(nnlist(nat,nnatmax))
         allocate(nnat(nat))
         allocate(nn_imag(3,nat,nnatmax))

c
            call inv3(ht,htm1,omega)
            call nbrlist(r,nnatmax)
c
            htin = ht
c
! set restart true if first MD step
         if (i_time.eq.1) then
            restart = .true.
         else
            restart = .false.
         end if

         if (f_pressure.eq.1) then
            tcel = .true.
         else
            tcel = .false.
         end if
c
         call force_ft(r,energy,sforce,stress,a_ew,i_time,.false.,
     &        .false.,htin,restart,tcel)
c
c
! convert energy from Hartree to kB K
        v_inter = energy*315773.2

! convert stress:
! stress(i,j)*omega is r_i*F_j/volume *volume in hartree
! force_scal_inter() is r_i * (h^-1 * F)_j in kB K / angstrom
! htm is in a_Bohr, so htm1*stress*omega is in hartree/a_Bohr
       force_scal_inter = 0.d0
       stress = +stress
       do j=1,3
       do i_dim=1,3
        force_scal_inter(i_dim) = force_scal_inter(i_dim) +
     &         htm1(i_dim,j)*stress(i_dim,j)*omega* 315773.2 / 0.5292d0
       end do
        if (e_dim.eq.3) then
!---   symmetrized "stress" field
        force_scal_inter(4) = force_scal_inter(4) +
     &     ( htm1(2,j)*stress(1,j)*omega* 315773.2 / 0.5292d0 +
     &       htm1(1,j)*stress(2,j)*omega* 315773.2 / 0.5292d0 ) *0.5
        force_scal_inter(5) = force_scal_inter(5) +
     &     ( htm1(3,j)*stress(2,j)*omega* 315773.2 / 0.5292d0 +
     &       htm1(2,j)*stress(3,j)*omega* 315773.2 / 0.5292d0 ) *0.5
        force_scal_inter(6) = force_scal_inter(6) +
     &     ( htm1(3,j)*stress(1,j)*omega* 315773.2 / 0.5292d0 +
     &       htm1(1,j)*stress(3,j)*omega* 315773.2 / 0.5292d0 ) *0.5
        end if
        end do

! convert force:
! sforce() is F in hartree / a_Bohr
! force() is h^-1 * F in kB K / angstrom^2
! htm is in a_Bohr, so htm1*sforce is in hartree/a_Bohr^2
        do i_part=1,nat
           do j=1,3
              force(j,0,i_part)=0
              do i=1,3
                 force(j,0,i_part)=force(j,0,i_part) +
     &                 htm1(j,i)*sforce(i,i_part)*
     &                 315773.2 /(0.5292d0**2)
              end do
           end do
        end do

! convert dipoles
```

```
      do i_part=1, n_part
         do i_dim=1, 3
            dipole(i_dim,i_part) = dip(i_dim,i_part) * 0.5292d0
         end do
      end do
      avg_dipole = 0.d0
      n_part_type = 0

      do i_part = 1, n_part
         i_type = type(i_part)
         avg_dipole(0,i_type) = avg_dipole(0,i_type) +
     &        sqrt( dipole(1,i_part)**2
     &             + dipole(2,i_part)**2 + dipole(3,i_part)**2 )
         do i_dim = 1, n_dim
            avg_dipole(i_dim,i_type) = avg_dipole(i_dim,i_type)
     &           + dipole(i_dim,i_part)
         end do
         n_part_type(i_type) = n_part_type(i_type) + 1
      end do

      do i_type = 1, n_type
         do i_dim = 0, n_dim
            avg_dipole(i_dim,i_type) = avg_dipole(i_dim,i_type) /
     &           n_part_type(i_type)
         end do
      end do

      if (i_time.eq.1) then
         avg_dipole_part = 0.d0
      end if

      do i_part = 1, n_part
         avg_dipole_part(0,i_part) = avg_dipole_part(0,i_part) +
     &        sqrt( dipole(1,i_part)**2
     &             + dipole(2,i_part)**2 + dipole(3,i_part)**2 )
         do i_dim = 1, n_dim
            avg_dipole_part(i_dim,i_part) =
     &           avg_dipole_part(i_dim,i_part) + dipole(i_dim,i_part)
         end do
      end do

      dipole_class = 0.d0

      do i_part = 1, n_part
         if (type(i_part).eq.2) then
            i_class = mod(i_part-1,18)-5
            dipole_class(0,i_class) =
     &              dipole_class(0,i_class)
     &              + sqrt(dipole(1,i_part)**2 + dipole(2,i_part)**2 +
     &           dipole(3,i_part)**2 )/(n_oxy/12.)
            do i_dim=1,3
               dipole_class(i_dim,i_class) =
     &              dipole_class(i_dim,i_class)
     &              + dipole(i_dim,i_part)/(n_oxy/12.)
!              if (i_dim.eq.3) write(37,*) i_part, i_class,
!     &             dipole(i_dim,i_part)
            end do
         end if
      end do
!      write(37,*) "================================================"

      do i_class = 1,12
         do i_dim = 0,3
            dipole_class1(i_dim,i_class) = dipole_class1(i_dim,i_class)
     &           + dipole_class(i_dim,i_class)
            dipole_class2(i_dim,i_class) = dipole_class2(i_dim,i_class)
     &           + (dipole_class(i_dim,i_class))**2
            dipole_class4(i_dim,i_class) = dipole_class4(i_dim,i_class)
     &           + (dipole_class(i_dim,i_class))**4
         end do
      end do

      dipole_count=dipole_count + 1

      if (i_time.eq.1) open(33,file="pimd.pol",status="unknown")

      write(33,'(15f15.7)') r_time,                     ! 1
     &     avg_dipole(0,2),                             ! 2
     &  avg_dipole(1,2),avg_dipole(2,2),avg_dipole(3,2),! 3,4,5
     &     (dipole(i_dim, 33), i_dim = 1,3),            ! 6,7,8
     &     (dipole(i_dim, 64), i_dim = 1,3)             ! 9,10,11


      deallocate(sforce,r,spind,nnlist,nnat,nn_imag)

      call force_on_scal

      end subroutine


! ! ! ! ! ! ! ! ! ! !
      subroutine applied_field
      implicit none
      include "pimd.com.f"
      double precision e_applied(n_dim)
      integer i_e_applied_bks

      e_applied(1) = 0.
      e_applied(2) = 0.
      e_applied(3) = 0.

!      if (f_e_applied.gt.0) then
!         if (i_time.ge.n_relax) then
!            i_e_applied_bks = f_e_applied
! i_e_applied_bks exists just to please the compiler
!            e_applied(1) = 0.01*(i_time)/50000
!         end if
!      end if

      do i_dim = 1, n_dim
! hartree /a_bohr /e -> kB K /angstrom /e
            e_applied(i_dim)=596724.9*e_applied(i_dim)
      end do

      do i_part = 1, n_part
         do i_dim = 1, n_dim
            v_inter = v_inter + charge(i_part)*r_real(i_dim,0,i_part)
     &           * e_applied(i_dim)
            do j_dim = 1, n_dim
               force(i_dim,0,i_part) = force(i_dim,0,i_part) +
     &           h_mat_inv(i_dim, j_dim) * charge(i_part)
     &           * e_applied(j_dim)
            end do
```

```
         end do
      end do


      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine bks_potential
      implicit none
      include "pimd.com.f"

      call twobody_potential

      call force_on_scal
      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine twobody_potential
! all 2 body potentials with short range plus coulomb term
      implicit none
      include "pimd.com.f"
      double precision v_inter_loc
      double precision r_1,r_2,pot_loc
      double precision r_2_max
      double precision mu_i(n_part), lagrange_nom, lagrange_denom
      integer i_dim_max

      v_inter = 0.
      v_realcoulomb = 0.
      v_shortrange = 0.

      self_energy = 0.
      do i_part = 1, n_part
         self_energy = self_energy + charge(i_part)**2
      end do
      self_energy = -n_trot*elec_prefac*alpha*self_energy/sqrt(pi)

      do i_dim = 1, n_dim + 3
       force_scal_inter(i_dim) = 0.
      end do

      if (f_potential.eq.3) call rec_ewald
      if ((f_potential.ge.5).and.(f_potential.le.8)) call rec_ewald

      if (f_chargeadjust.eq.3) then
         do i_part = 1, n_part
            i_type=type(i_part)
            mu_i(i_part) = qeqelectroneg(i_type) + qeqrec(i_part)
     &           + ( qeqhardness(i_type)
     &           - 2*elec_prefac*alpha/sqrt(pi) )*charge(i_part)
         end do
      end if

      do i_part = 1,n_part
       i_type = type(i_part)
       v_inter_loc = 0.

       do i_neigh = 1,neighbor(0,i_part)
        j_part = neighbor(i_neigh,i_part)
        j_type = type(j_part)

      pot_loc = 0.
      do i_trot = 0,n_trot-1

       do i_dim = 1,n_dim
        delta_cell(i_dim) =
&     r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
        if(delta_cell(i_dim).gt.0.5) then
         delta_cell(i_dim) = delta_cell(i_dim) - 1.
        else if(delta_cell(i_dim).lt.-0.5) then
         delta_cell(i_dim) = delta_cell(i_dim) + 1.
        end if
       end do

       r_2 = 0.
       call calc_dist_int(r_2)

       if(r_2.lt.r_cutoff_2(i_type,j_type)) then
        r_1 = sqrt(r_2)
        i_potential = nint(n_potential*r_1/r_range(i_type,j_type))
        pot_loc = pot_loc + r_pot_1(i_type,j_type,i_potential)
&          + charge(i_part)*charge(j_part)
&             *r_pot_2(i_type,j_type,i_potential)

        v_realcoulomb = v_realcoulomb
&          + charge(i_part)*charge(j_part)
&             *r_pot_2(i_type,j_type,i_potential)
        v_shortrange = v_shortrange
&          + r_pot_1(i_type,j_type,i_potential)

        r_dummy = r_pot_d_1(i_type,j_type,i_potential)
&          + charge(i_part)*charge(j_part)
&             *r_pot_d_2(i_type,j_type,i_potential)
        call calc_force_loc(r_dummy,i_trot,i_part,j_part)

        if (f_chargeadjust.eq.3) then
           mu_i(i_part) = mu_i(i_part)
&             + r_pot_2(i_type,j_type,i_potential) * charge(j_part)
           mu_i(j_part) = mu_i(j_part)
&             + r_pot_2(j_type,i_type,i_potential) * charge(i_part)
        end if
       end if

      end do
      v_inter_loc = v_inter_loc + pot_loc
     end do

     v_inter = v_inter + v_inter_loc
    end do

    v_inter = v_inter + self_energy


    if (f_chargeadjust.eq.3) then
       lagrange_nom = 0.
       do i_part = 1, n_part
          i_type = type(i_part)
          lagrange_nom = lagrange_nom +
&            mu_i(i_part) / qeqmass(i_type)
       end do
       lagrange_denom = 0.
       do i_type = 1, n_type
          lagrange_denom = lagrange_denom +
```

```
     &           n_atomtype(i_type) / qeqmass(i_type)
         end do
       end if

       t_dynamic_charge = 0.
       v_charge = 0.
       do i_part=1, n_part
          i_type = type(i_part)
          if (f_chargeadjust.eq.3) then
!----  calculate force on charges
             fQ(i_part) = -mu_i(i_part) + lagrange_nom / lagrange_denom
!----  calculate energies of the extended lagrangian part
             t_dynamic_charge = t_dynamic_charge
     &          + 0.5* qeqmass(i_type)*(chargex(1,i_part)/dt)**2
          end if
          v_charge = v_charge
     &        + qeqelectroneg(i_type)*charge(i_part)
     &        + 0.5*qeqhardness(i_type)*charge(i_part)**2
       end do


       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine electronegativity_save
       implicit none
       include "pimd.com.f"
       double precision en(n_part), r_1, r_2
       double precision avg_en(n_type), min_en(n_type), max_en(n_type)
       integer num_part(n_type)

       do i_part = 1,n_part
        i_type = type(i_part)
        en(i_part)=0

        do j_part = 1, n_part
         if (j_part.ne.i_part) then
          j_type = type(j_part)
          do i_trot = 0,n_trot-1

           do i_dim = 1,n_dim
            delta_cell(i_dim) =
     &       r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
            if(delta_cell(i_dim).gt.0.5) then
             delta_cell(i_dim) = delta_cell(i_dim) - 1.
            else if(delta_cell(i_dim).lt.-0.5) then
             delta_cell(i_dim) = delta_cell(i_dim) + 1.
            end if
           end do

           r_2 = 0.
           call calc_dist_int(r_2)

           if(r_2.lt.r_cutoff_2(i_type,j_type)) then
            r_1 = sqrt(r_2)
            i_potential = nint(n_potential*r_1/r_range(i_type,j_type))
            en(i_part)=en(i_part)
     &       + r_pot_2(i_type,j_type,i_potential) * charge(j_part)
           end if

          end do
         end if

        end do
        en(i_part)=en(i_part)+qeqelectroneg(i_type)
     &      -2*elec_prefac*alpha/sqrt(pi)*charge(i_part)
     &    + charge(i_part)*qeqhardness(i_type)
       end do

       do i_part=1, n_part
          write(*,*) i_part, en(i_part)
       end do

! hier stehts

!      write(62,'(1000e15.7)') (en(i_part),i_part=1,n_part)

       do i_type = 1, n_type
          avg_en(i_type)=0
          max_en(i_type)=-1.e30
          min_en(i_type)=1.e30
          num_part(i_type)=0
       end do
       do i_part = 1, n_part
          i_type = type(i_part)
          num_part(i_type)=num_part(i_type)+1
       end do
       do i_part = 1, n_part
          i_type = type(i_part)
          avg_en(i_type)=avg_en(i_type)+en(i_part)/num_part(i_type)
          if (en(i_part).gt.max_en(i_type))
     &         max_en(i_type)=en(i_part)
          if (en(i_part).lt.min_en(i_type))
     &         min_en(i_type)=en(i_part)
         end do

       write(63,'(20e15.7)') (avg_en(i_type),i_type=1,n_type),
     &     (min_en(i_type),i_type=1,n_type),
     &     (max_en(i_type),i_type=1,n_type)

       end subroutine

!!!!!!!!!!!!!!!!!!!!

       subroutine extdlagrange_predictor
       implicit none
       include "pimd.com.f"

!---- predict charges
       do i_part=1, n_part
          do i_order=1, n_order
             charge(i_part) = charge(i_part)
     &           + predict_coef(0,i_order)*chargex(i_order,i_part)
          end do
       end do

!---- predict higher order charges
       do i_part = 1, n_part
         do i_order = 1, n_order-1
           do j_order = i_order+1, n_order
             chargex(i_order,i_part) =
     &          chargex(i_order,i_part) +
     &          predict_coef(i_order,j_order) * chargex(j_order,i_part)
```

```
          end do
        end do
      end do

      end subroutine

!!!!!!!!!!!!!!!!!!!!

      subroutine extdlagrange_corrector
      implicit none
      include "pimd.com.f"
      double precision deltaQ2(n_part)

!---- subract friction
      do i_part = 1,n_part
         i_type = type(i_part)
         fQ(i_part) = fQ(i_part)
     &   - qeqmass(i_type) * qeqfriction * chargex(1,i_part)/dt
      end do

!---- calculate second order correction

      do i_part = 1, n_part
         i_type = type(i_part)
         deltaQ2(i_part) = (dt_2*fQ(i_part)/qeqmass(i_type) / 2
     &        -chargex(2,i_part))
      end do

!---- correct charges
      do i_part = 1, n_part
         charge(i_part) = charge(i_part)
     &        + correct_coef(0)*deltaQ2(i_part)
         do i_order = 1, n_order
            chargex(i_order,i_part) = chargex(i_order,i_part)
     &           + correct_coef(i_order)*deltaQ2(i_part)
         end do
      end do

      end subroutine


!!!!!!!!!!!!!!!!!!!!


      subroutine extendedlagrange
      implicit none
      include "pimd.com.f"
      double precision r_1, r_2
      double precision qeqa(n_part,n_part), qeqb(n_part)
      double precision mu_i(n_part), sum_mu
      double precision v_qabhaengig_etest
      common /extlagetest/ v_qabhaengig_etest
      double precision qeqrec2(n_part,n_part)
      common /qeqrectest/ qeqrec2

      integer i_test
      double precision pmix
      common /pmisch/ pmix
      double precision avg_charge(n_type), max_charge(n_type),
     &     min_charge(n_type)
      double precision v_ewald2
      double precision qeqa12(n_part,n_part)


!     (can be included in potential routine later)
      do i_part = 1,n_part
       i_type = type(i_part)
       do j_part = 1, n_part
          j_type = type(j_part)
          i_trot = 0
          do i_dim = 1,n_dim
             delta_cell(i_dim) =
     &            r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
             if(delta_cell(i_dim).gt.0.5) then
                delta_cell(i_dim) = delta_cell(i_dim) - 1.
             else if(delta_cell(i_dim).lt.-0.5) then
                delta_cell(i_dim) = delta_cell(i_dim) + 1.
             end if
          end do

          r_2=0.
          call calc_dist_int(r_2)

          if(r_2.le.r_cutoff_2(i_type,j_type)) then
             r_1 = sqrt(r_2)
             i_potential = nint(n_potential*r_1/r_range(i_type,j_type))
             qeqa(i_part,j_part) =
     &            + r_pot_2(i_type,j_type,i_potential)
             qeqa12(i_part,j_part) = 0.5*(
     &            + r_pot_2(i_type,j_type,i_potential)*charge(i_part)
     &            *charge(j_part) + r_pot_1(i_type,j_type,i_potential))
             if (i_potential.ge.n_potential) write(68,*) "grosse Panne"
          else
             qeqa(i_part,j_part)= 0.0
             qeqa12(i_part,j_part)= 0.0
          end if
       end do
      end do

!     calculate aux variable B_i
      do i_part=1, n_part
         i_type = type(i_part)
         qeqb(i_part) = qeqhardness(i_type)
     &        - 2*elec_prefac*alpha/sqrt(pi)
       end do

!----  calculate chemical potential
      sum_mu = 0.
      do i_part=1, n_part
         i_type = type(i_part)
         mu_i(i_part) = qeqelectroneg(i_type) + qeqrec(i_part)
     &        + qeqb(i_part)*charge(i_part)
         do j_part=1, n_part
            mu_i(i_part) = mu_i(i_part) + qeqa(i_part,j_part)
     &           * charge(j_part)
         end do
         sum_mu = sum_mu + mu_i(i_part)
       end do

!----  calculate resulting force
      do i_part=1, n_part
         fQ(i_part) = -mu_i(i_part) + sum_mu / n_part
       end do

!----  calculate energies
```

```
 t_dynamic_charge = 0.
 v_charge = 0.
 v_qabhaengig_etest = 0.
 v_inter = 0.
 v_ewald2 = 0.

 do i_part = 1, n_part
    i_type = type(i_part)
    t_dynamic_charge = t_dynamic_charge
&        + 0.5* qeqmass(i_type)*(chargex(1,i_part)/dt)**2
    v_charge = v_charge
&        + 0.5*(mu_i(i_part)+qeqelectroneg(i_type))*charge(i_part)
&        + qeqelectroneg(i_type)*charge(i_part)
&        + 0.5*qeqhardness(i_type)*charge(i_part)**2
 end do

 do i_part = 1, n_part
    i_type = type(i_part)
    do j_part = 1, n_part
       if (i_part.ne.j_part) then
          j_type = type(j_part)
          v_qabhaengig_etest = v_qabhaengig_etest +
&              0.5* qeqa(i_part,j_part)*charge(i_part)
&              *charge(j_part)

          v_inter = v_inter + qeqa12(i_part,j_part)
       end if
    end do
    v_qabhaengig_etest = v_qabhaengig_etest
&        + 0.5* qeqb(i_part)*charge(i_part)**2
&        + qeqelectroneg(i_type)*charge(i_part)
    v_inter = v_inter - elec_prefac*alpha/sqrt(pi)
&        *charge(i_part)*charge(i_part)
&        + 0.5*qeqrec(i_part)*charge(i_part)
    v_ewald2 = v_ewald2 + 0.5*qeqrec(i_part)*charge(i_part)
 end do
 v_qabhaengig_etest = v_qabhaengig_etest + v_ewald


 do i_type = 1, n_type
    avg_charge(i_type)=0
    max_charge(i_type)=-1.e30
    min_charge(i_type)=1.e30
 end do
 do i_part = 1, n_part
    i_type = type(i_part)
    avg_charge(i_type)=avg_charge(i_type)+charge(i_part)
    if (charge(i_part).gt.max_charge(i_type))
&        max_charge(i_type)=charge(i_part)
    if (charge(i_part).lt.min_charge(i_type))
&        min_charge(i_type)=charge(i_part)
 end do
 avg_charge(1)=1.d0*avg_charge(1)/n_silic
 avg_charge(2)=1.d0*avg_charge(2)/n_oxy

 write(64,'(i6,20e30.22)') i_test,
2        avg_charge(1),
3        min_charge(1),
4        max_charge(1),
5        avg_charge(2),
6        min_charge(2),
7        max_charge(2),
8        pmix,
9        t_dynamic_charge/n_part,
&        v_charge/n_part,                !10
&        v_qabhaengig_etest/n_part,      !11
&        v_ewald/n_part,                 !12
&        v_inter/n_part,                 !13
&        v_ewald2/n_part                 !14

      end subroutine


!!!!!!!!!!!!!!!!!!!!

      subroutine matrixinversion
      implicit none
      include "pimd.com.f"

      integer m,n,np,mp
      PARAMETER (np=n_part,mp=1)
      double precision a(np,np), b(np,mp)
      double precision r_2, r_1
      integer k_part
      double precision qeqrec2(n_part,n_part)
      common /qeqrectest/ qeqrec2
      double precision qeqa(n_part,n_part), qeqb(n_part)
      double precision en
      double precision row_dummy
      integer indx(n_part)

      n=np
      m=mp

      if (f_potential.ne.1) then
        call check_skin
        if (f_skin.eq.1) then
         call binning
        end if
      end if

      call matrix_rec(qeqrec2)

      b(1,1)=0.  ! Q_total
      a(1,1)=1.

      j_type=type(1)
      do i_part = 2, n_part         ! first row 1
         i_type=type(i_part)
         a(1,i_part)=1.
         b(i_part,1)=-(qeqelectroneg(i_type)-qeqelectroneg(j_type))
      end do

      i_type=type(1)            ! first column -B1
      do i_part = 2, n_part
         a(i_part,1)=-(qeqhardness(i_type)-
     &        2*elec_prefac*alpha/sqrt(pi) + qeqrec2(1,1))
         qeqb(1)=-a(i_part,1)
      end do

      i_part = 1                !i=2..n x j=2..n  -A1j
      i_type = type(i_part)
      do j_part = 2, n_part
```

```
            j_type = type(j_part)
            i_trot = 0

            do i_dim = 1,n_dim
               delta_cell(i_dim) =
     &            r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
               if(delta_cell(i_dim).gt.0.5) then
                  delta_cell(i_dim) = delta_cell(i_dim) - 1.
               else if(delta_cell(i_dim).lt.-0.5) then
                  delta_cell(i_dim) = delta_cell(i_dim) + 1.
               end if
            end do

            r_2=0.
            call calc_dist_int(r_2)

            if(r_2.le.r_cutoff_2(i_type,j_type)) then
               r_1 = sqrt(r_2)

               i_potential = nint(n_potential*r_1/r_range(i_type,j_type))

               do k_part=2, n_part
                  a(k_part,j_part)=-r_pot_2q(i_type,j_type,i_potential)
     &                 -qeqrec2(i_part,j_part)
               end do
            else
               do k_part=2, n_part
                  a(k_part,j_part)=-qeqrec2(i_part,j_part)
               end do
            end if
            qeqa(1,j_part)=-a(2,j_part)
         end do

         do i_part = 2,n_part
          i_type = type(i_part)
          do j_part = 1, n_part
             if (i_part.ne.j_part) then   ! without diagonal: Aij
                j_type = type(j_part)
                i_trot = 0
                do i_dim = 1,n_dim
                   delta_cell(i_dim) =
     &                 r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
                   if(delta_cell(i_dim).gt.0.5) then
                     delta_cell(i_dim) = delta_cell(i_dim) - 1.
                   else if(delta_cell(i_dim).lt.-0.5) then
                     delta_cell(i_dim) = delta_cell(i_dim) + 1.
                   end if
                end do

                r_2=0.
                call calc_dist_int(r_2)

!                write(*,*) i_part, j_part, sqrt(r_2)

                if(r_2.le.r_cutoff_2(i_type,j_type)) then
                 r_1 = sqrt(r_2)
                 i_potential = nint(n_potential*r_1/r_range(i_type,j_type))
                 a(i_part,j_part)= a(i_part,j_part)
     &               + r_pot_2q(i_type,j_type,i_potential)
     &               + qeqrec2(i_part,j_part)
                 qeqa(i_part,j_part)= r_pot_2q(i_type,j_type,i_potential)
     &               + qeqrec2(i_part,j_part)
                else
                 a(i_part,j_part)= a(i_part,j_part)
     &               + qeqrec2(i_part,j_part)
                 qeqa(i_part,j_part)= qeqrec2(i_part,j_part)
                end if
             else                    ! diagonal: Bi
                a(i_part,i_part)= a(i_part,i_part) + qeqhardness(i_type)
     &               - 2*elec_prefac*alpha/sqrt(pi) + qeqrec2(i_part,i_part)
                qeqb(i_part) = qeqhardness(i_type)
     &               - 2*elec_prefac*alpha/sqrt(pi) + qeqrec2(i_part,i_part)
             end if
           end do
          end do

          call ludcmp(a,n,np,indx,row_dummy)
          call lubksb(a,n,np,indx,b)

          v_qeqself=0.
          do i_part=1, n_part
             i_type=type(i_part)
             charge(i_part)=b(i_part,1)
             v_qeqself=v_qeqself
     &         + charge(i_part)*qeqelectroneg(i_type)
     &         + 0.5*charge(i_part)*charge(i_part)*qeqhardness(i_type)
          end do

          end subroutine


! ! ! ! ! !
          subroutine matrix_rec(qeqrec2)
          implicit none
          include "pimd.com.f"

          integer i_q(n_dim), trot_max
          double precision q_vec(n_dim),q_2,loc_prefac,rec_prefac
          double precision two_rec_prefac
          complex*16 plus_iqr(n_part),minus_iqr(n_part),
         &     diff_iqr(n_part,n_part)
          double precision qeqrec2(n_part,n_part)

          do i_part=1, n_part
             do j_part=1, n_part
                qeqrec2(i_part,j_part)=0.
             end do
          end do

          loc_prefac = elec_prefac*twopi/volume

          j_trot = mod(int(n_trot*random(1)),n_trot)
          do i_rec_vec = 1,n_rec_vec
           q_2 = 0.
           do i_dim = 1,n_dim
            i_q(i_dim) = rec_vec_list(i_rec_vec,i_dim)
            q_vec(i_dim) = i_q(i_dim)*dq(i_dim,i_dim)
            q_2 = q_2 + q_vec(i_dim)**2
           end do
           rec_prefac = loc_prefac*exp(-q_2/(2*alpha)**2)/q_2
           if(rec_vec_list(i_rec_vec,3).ne.0) rec_prefac = 2 * rec_prefac
           two_rec_prefac = 2*rec_prefac
           trot_max = n_trot-1
           if (f_opt_rec.eq.1) trot_max = 0
```

```
        do i_trot = 0, trot_max
         if (f_opt_rec.eq.0) j_trot = i_trot

        do i_part = 1, n_part
         r_dummy = 0.
         if (f_opt_rec.eq.0) then
          do i_dim = 1,n_dim
           r_dummy = r_dummy + i_q(i_dim)*r0(i_dim,j_trot,i_part)
          end do
         else
          do i_dim = 1, n_dim
           r_dummy = r_dummy + i_q(i_dim) *
     &      ( r_ref(i_dim,i_part) + u0(i_dim,1,i_part)/sqrtp )
          end do
         end if
         r_dummy = dmod(r_dummy,1.d0)
         i_dummy = nint(r_dummy*n_potential)
         plus_iqr(i_part)=
!     &   tab_exp_ima(i_dummy)
     &   exp(2.*pi*(0.,1.)*r_dummy)
         i_dummy = nint(-r_dummy*n_potential)
         minus_iqr(i_part)=
!     &   tab_exp_ima(i_dummy)
     &   exp(-2.*pi*(0.,1.)*r_dummy)
        end do

        goto 234
        do i_part = 1, n_part
           do j_part = 1, n_part
             do i_dim = 1,n_dim
                delta_cell(i_dim) =
     &               r0(i_dim,i_trot,i_part) - r0(i_dim,i_trot,j_part)
                if(delta_cell(i_dim).gt.0.5) then
                  delta_cell(i_dim) = delta_cell(i_dim) - 1.
                else if(delta_cell(i_dim).lt.-0.5) then
                  delta_cell(i_dim) = delta_cell(i_dim) + 1.
                end if
             end do
             r_dummy = 0.
             do i_dim = 1,n_dim
                  r_dummy = r_dummy + i_q(i_dim)
     &                 *delta_cell(i_dim)
             end do
!            r_dummy = dmod(r_dummy,1.d0)
!            i_dummy = nint(r_dummy*n_potential)
!            diff_iqr(i_part,j_part)=
!     &           tab_exp_ima(i_dummy)
!    &           exp(2.*pi*(0.,1.)*r_dummy)
             diff_iqr(i_part,j_part)=
!     &           tab_exp_ima(i_dummy)
     &           exp(2.*pi*(0.,1.)*r_dummy)
           end do
        end do

 234    continue

        do i_part=1, n_part
           do j_part=1, n_part
              qeqrec2(i_part,j_part)=qeqrec2(i_part,j_part)
     &           + two_rec_prefac*dble(plus_iqr(i_part)*
     &           minus_iqr(j_part))
!             qeqrec2(i_part,j_part)=qeqrec2(i_part,j_part)
!     &           + two_rec_prefac*dble(diff_iqr(i_part,j_part))
         end do
           end do

        end do
       end do

       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine thermostat
       implicit none
       include "pimd.com.f"

!---- only n_trot*n_part independent (random) forces
!---- real part of i_trot = 0, i_trot = n_trot/2
!---- both parts of 1 .le. i_trot .le. n_trot/2

       do j_trot = 0, n_trot-1
        i_trot = j_trot + 1

!---   choose larger damping for internal modes
        if (i_trot.eq.1) then
         r_dummy = 1. / n_time_scal
        else
         r_dummy = sqrt(1.*n_time_scal)
         if(i_trot.eq.2) i_trot = n_trot + 1
        end if

        i_dummy = (i_trot-1)/2
        do i_part = 1,n_part
         i_type = type(i_part)
         call r250(mz,random,n_dim,n_dim,kptr)
         do i_dim = 1,n_dim
          f0(i_dim,i_trot,i_part) = f0(i_dim,i_trot,i_part)
     &    - mass(i_dummy,i_type) * friction(i_type) * r_dummy**2
     &    * (ux(1,i_dim,i_trot,i_part)/dt)
     &    - k_eigen(i_dummy,i_type) * u0(i_dim,i_trot,i_part)
          fr(i_dim,i_trot,i_part) = r_dummy
     &    * (2*random(i_dim)-1.) / mass(i_dummy,i_type)
!----    width of random force is known in predictor
        end do
       end do

       do i_dim = 1, n_dim + e_dim
        force_scal(i_dim) = force_scal(i_dim)
      & - mass_box * fric_box * (scal_x(1,i_dim)/dt)
       end do
       if (fric_box.gt.0.) call r250(mz,random,n_dim,n_dim,kptr)
       do i_dim = 1, n_dim
        fr_box(i_dim) = (2*random(i_dim)-1.) / mass_box
       end do
       if (e_dim.eq.3) then
        call r250(mz,random,n_dim,n_dim,kptr)
        do i_dim = 1, 3
         fr_box(i_dim+n_dim) = (2*random(i_dim)-1.) / mass_box
        end do
       end if

       end subroutine
```

```
! ! ! ! ! ! ! ! ! ! !

      subroutine obse_init
      implicit none
      include "pimd.com.f"
      integer jseed
      common /cjseed/ jseed

      t_dynamic_1 = 0.
      t_dynamic_2 = 0.

      t_diag_1 = 0.
      t_ndiag_1 = 0.

      v_inter_1 = 0.
      v_inter_2 = 0.

      viri_est_1 = 0.d0
      viri_est_2 = 0.d0

      prim_est_1 = 0.d0
      prim_est_2 = 0.d0

      do i_dim = 1, n_dim + 3
       scal_1(i_dim) = 0.
       do j_dim = 1,n_dim + 3
        scal_2(j_dim,i_dim) = 0.
       end do
      end do
      vol_1 = 0.
      vol_2 = 0.

      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
       strain_1(i_dim,j_dim) = 0.
       ela_kin(i_dim,j_dim) = 0.
       do k_dim = 1, n_dim
       do l_dim = 1, n_dim
        ela_born_1(i_dim, j_dim, k_dim, l_dim) = 0.
        strain_2(i_dim, j_dim, k_dim, l_dim) = 0.
       end do
       end do
      end do
      end do

      di_time = 1
      ieff_time = 0

      do i_trot = 0,n_trot-1
       cor_ifunc(i_trot) = 0.d0
      end do

      do i_cor = 0,n_cor
       cor_obs(i_cor) = 0.d0
       qdot_cor(i_cor) = 0.d0
       do i_type = 1, n_type
          cor_func(i_cor,i_type) = 0.d0
       end do
       cor_func_A(i_cor) = 0.d0
       cor_func_B(i_cor) = 0.d0
      end do

      do i_type = 1,n_type
      do j_type = 1,n_type
       do i_grid = 0,n_grid+1
        g_r(i_type,j_type,i_grid) = 0.
       end do
      end do
      end do

      time_obs = 0.d0
      count_ifunc = 0
      i_measure = n_cor

      time_born = 0.

      ord_par_1 = 0.d0
      ord_par_2 = 0.d0
      ord_par_4 = 0.d0
      ord_parB_1 = 0.d0
      ord_parB_2 = 0.d0
      ord_parB_4 = 0.d0
      ord_parC_1 = 0.d0
      ord_parC_2 = 0.d0
      ord_parC_4 = 0.d0

!     movie init
      if (f_movie.eq.1) then
         open(46,file="movie.xyz",status="new")
         open(47,file="movie.pol",status="new")
         write(46,*) n_part
         write(46,*)
      end if

      if (f_confmovie.eq.1) then
         open(48,file="movie.conf",status="new")
      end if

      if (i_run.eq.1) then
       open(20,file="pimd.out",status="unknown")

       write(20,210) f_potential,"   f_potential, information available"
       write(20,210) f_chargeadjust,"   f_chargeadjust"
       write(20,210) n_relax,"   # of relaxation steps"
       write(20,210) n_obser,"   # of observation steps"
       write(20,210) n_safe,"   # of steps between configuration output"
       write(20,210) jseed+2,"   random # init"
       write(20,*)
       write(20,201) dt*n_time_scal,"   time step"
       write(20,201) tk,"   temperature"
       write(20,*)
       write(20,202) (mass(0,i_type),i_type=1,n_type),"   masses"
       write(20,202) (friction(i_type),i_type=1,n_type),"   friction"
       write(20,202) mass_box/n_trot, fric_box,"   mass + fr. of box"
       if(f_potential.gt.1) then
        write(20,201)
        write(20,201) press,"   pressure"
        write(20,210) f_pressure,"   0 = NVT; 1 = NpT"
        if (f_potential.eq.3.or.(
     &       (f_potential.ge.5).and.(f_potential.le.8)))
     &  write(20,210) f_opt_rec,"   1 = optimized Ewald"
        write(20,*)
        if (f_chargeadjust.ne.1) then
```

```
     write(20,202) (qeqelectroneg(i_type),i_type=1,n_type),
&    "   electronegativity"
     write(20,202) (qeqhardness(i_type),i_type=1,n_type),
&    "   hardness of EN"
   end if
   write(20,*)
   if (f_chargeadjust.eq.3) then
     write(20,202) (qeqmass(i_type),i_type=1,n_type),
&    "   fictitious charge mass"
     write(20,201) qeqfriction, "   charge friction"
   end if
   write(20,*)
   if(f_potential.eq.2) then
    do i_type = 1,n_type
     write(20,202) (epsil(i_type,j_type)/4,j_type=1,n_type),
&    "   energy parameters"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (sigma(i_type,j_type),j_type=1,n_type),
&    "   length scale parameters"
    end do
   else if(f_potential.eq.3) then
    do i_type = 1,n_type
     write(20,202) (epsil(i_type,j_type),j_type=1,n_type),
&     "    exp. energy parameters"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (sigma(i_type,j_type),j_type=1,n_type),
&     "   inverse exp. lenght scales"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (c_vandwaals(i_type,j_type),j_type=1,n_type),
&     "   van de Waals strength"
    end do
   else if((f_potential.ge.5).and.(f_potential.le.7)) then
    do i_type = 1,n_type
     write(20,202) (d0(i_type,j_type),j_type=1,n_type),
&     "   MS bond strength D0"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (msr0(i_type,j_type),j_type=1,n_type),
&     "   MS bond length R0"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (gamma(i_type,j_type),j_type=1,n_type),
&     "   MS force constant gamma"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (mszeta(i_type,j_type),j_type=1,n_type),
&     "   slater orbital exponent zeta"
    end do
    write(20,*)
    write(20,220) (msn(i_type),i_type=1,n_type),
&     "   principal quantum number n"
   else if(f_potential.eq.8) then
    do i_type = 1,n_type
     write(20,202) (vash_h(i_type,j_type),j_type=1,n_type),
&     "    repulsive strength H"
    end do
    write(20,*)
    do i_type = 1,n_type
     write(20,202) (vash_eta(i_type,j_type),j_type=1,n_type),
&     "   repulsive exponents eta"
    end do
    write(20,*)
    write(20,202) (vash_alpha(i_type),i_type=1,n_type),
&     "   polarizability alpha"
    write(20,202) (vash_B(i_type),i_type=1,n_type),
&     "   three body B"
    write(20,202) (vash_l(i_type),i_type=1,n_type),
&     "   three body l"
    write(20,202) (vash_ctheta(i_type),i_type=1,n_type),
&     "   three body cos theta"
    write(20,202) (vash_r0(i_type),i_type=1,n_type),
&     "   three body r0"
   end if
   write(20,*)
   if(f_potential.eq.3.or.
&       ((f_potential.ge.5).and.(f_potential.le.9))) then
    write(20,202) alpha, v_cut,"   alpha, v_cut"
   end if
   if (f_potential.le.2.or.f_potential.eq.4)  then
    do i_type = 1,n_type
     write(20,202) (r_cutoff(i_type,j_type),j_type=1,n_type),
&    "   cutoff radius"
    end do
   end if
   write(20,*)
   do i_type = 1,n_type
    write(20,202) (r_skin(i_type,j_type),j_type=1,n_type),
&   "   skin radius"
   end do
  end if
  write(20,*)
  write(20,210) n_silic,"   # of silicium atoms"
  write(20,210) n_oxy,"   # of oxygen atoms"
  write(20,210) n_trot,"   Trotter #"
  write(20,210) n_order,"   order of integrator"
  write(20,210) n_time_scal,"    time scale separation int/ext"
  if (f_potential.eq.3.or.
&      ((f_potential.ge.5).and.(f_potential.le.8))) then
   write(20,*)
   write(20,202) alpha,sqrt(q_cut_2),"   alpha,q_cut"
   write(20,202) erfcc(alpha*r_cutoff(1,1))
&  ,exp(-q_cut_2/(2*alpha)**2),"   max. ind. errors"
   write(20,201) r_cut_short,"   r_cut_short"
   write(20,210) f_symmetry,"   symmetry"
  end if
 end if

  if (f_potential.eq.3.or.
&      ((f_potential.ge.5).and.(f_potential.le.8)))
&     call hist_ang_init


  dipole_class1 = 0.d0
  dipole_class2 = 0.d0
  dipole_class4 = 0.d0
```

```
       dipole_count = 0

       open(86,file='pimd.siosi',status='unknown')
       open(87,file='pimd.osio',status='unknown')

!       open(65,file='pimd.dip',status='unknown')

  201 format(1e14.6,a)
  202 format(2e14.6,a)
  220 format(2i14,a)
  203 format(3e14.6,a)
  210 format(1i14,a)

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine hist_ang_init
      implicit none
      include "pimd.com.f"
      do i_grid = 1, n_grid
       do i_dummy = 1, 3
        hist_ang(i_dummy, i_grid) = 0.
       end do
      end do
      neigh_time = 0.
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine observation
      implicit none
      include "pimd.com.f"
      integer j_cor,k_cor, j_measure, i_class
      double precision r_2,r_1, vel(n_dim), pbc_norm
      integer pbc_x,pbc_y,pbc_z

      t_dynamic_1 = t_dynamic_1 + t_dynamic*dt
      t_dynamic_2 = t_dynamic_2 + t_dynamic**2*dt

      t_diag_1 = t_diag_1 + t_diag*dt
      t_ndiag_1 = t_ndiag_1 + t_ndiag*dt

      v_inter_1 = v_inter_1 + (v_inter-v_gauge)*dt
      v_inter_2 = v_inter_2 + (v_inter-v_gauge)**2*dt

      if (f_potential.gt.1) call viri_estimator
      viri_est_1 = viri_est_1 + viri_est*dt
      viri_est_2 = viri_est_2 + viri_est**2*dt

      call prim_estimator
!     r_dummy = n_dim*n_part*n_trot*tkp/2 - v_chain
      r_dummy = t_dynamic - v_chain
      prim_est_1 = prim_est_1 + r_dummy*dt
      prim_est_2 = prim_est_2 + r_dummy**2*dt

      do i_dim = 1, n_dim + 3
       scal_1(i_dim) = scal_1(i_dim) + scal_0(i_dim)*dt
       do j_dim = 1,n_dim + 3
        scal_2(j_dim,i_dim) = scal_2(j_dim,i_dim)
     &  + scal_0(j_dim)*scal_0(i_dim)*dt
       end do
      end do
      vol_1 = vol_1 + volume*dt
      vol_2 = vol_2 + volume**2*dt

      time_obs = time_obs + dt

      if (mod(ieff_time,ni_time).eq.0) then

!----  imaginary time correlation function

       call r250(mz,random,n_dim,1,kptr)
       i_trot = int(n_trot*random(1))
       if (i_trot.eq.n_trot) i_trot = 0
       count_ifunc = count_ifunc + 1

!----  TERM meas cor_ifunc
       do i_dummy = 0, n_trot-1
       j_trot = mod(i_trot+i_dummy,n_trot)
        do i_part = 1,n_part
        i_type = type(i_part)
        do i_dim = 1,n_dim
         cor_ifunc(i_dummy) = cor_ifunc(i_dummy) + mass(0,i_type) * (
     &   r_real(i_dim,i_trot,i_part) - r_real(i_dim,j_trot,i_part) ) **2
        end do
        end do
       end do

!----  centroid time correlation function

!----- TERM meas conf_centr
       if(i_measure.eq.n_cor) then
        do i_part = 1,n_part
         i_type = type(i_part)
         do i_dim = 1,n_dim
          conf_centr(i_dim,i_part) = 0.
          qdot0(i_dim,i_part) = 0. ! Q'(t=0)
          do j_dim = 1, n_dim
             if ( (e_dim.eq.3) .or. (i_dim.eq.j_dim) ) then
                conf_centr(i_dim,i_part) = conf_centr(i_dim,i_part)
     &               + h_mat(i_dim,j_dim) * ux(1,j_dim,1,i_part)
                qdot0(i_dim,i_part) = qdot0(i_dim,i_part)
     &               + chargex(1,i_part)
             end if
          end do
          conf_centr(i_dim,i_part) = conf_centr(i_dim,i_part) / dt
          qdot0(i_dim,i_part) = qdot0(i_dim,i_part) / dt
         end do
        end do
        i_measure = 0
        cor_func(n_cor,1) = cor_func(n_cor,1) + 1.
        qdot_cor(n_cor) = qdot_cor(n_cor) + 1.
        cor_func_A(n_cor) =cor_func_A(n_cor) +1.
       end if

       do i_part = 1,n_part
       i_type = type(i_part)
       do i_dim = 1,n_dim
        vel(i_dim) = 0.
        do j_dim = 1, n_dim
         if ( (e_dim.eq.3) .or. (i_dim.eq.j_dim) )
     &   vel(i_dim) = vel(i_dim)
     &   + h_mat(i_dim,j_dim) * ux(1,j_dim,1,i_part)
```

```
         end do
         vel(i_dim) = vel(i_dim) / dt
         cor_func(i_measure,i_type)
     &    = cor_func(i_measure,i_type) + mass(0,i_type) *
     &   conf_centr(i_dim,i_part) * vel(i_dim) / n_trot
         qdot_cor(i_measure) = qdot_cor(i_measure) + qeqmass(i_type) *
     &        qdot0(i_dim,i_part) * chargex(1, i_part) / dt
         cor_func_A(i_measure)
     &    = cor_func_A(i_measure) + mass(0,i_type) *
     &   conf_centr(i_dim,i_part) * vel(i_dim) / n_trot
!---     method B: better statistics, less independence:
         stored_vel(i_dim,i_measure,i_part)=vel(i_dim)
         if (i_time.gt.(n_relax+n_cor)) then
            cor_func_B(n_cor) =cor_func_B(n_cor) +1.
            do j_measure = 0, i_measure
               cor_func_B(j_measure)=cor_func_B(j_measure) +
     &              mass(0,i_type)*vel(i_dim)
     &              *stored_vel(i_dim,i_measure-j_measure,i_part)
     &              / n_trot
            end do
            do j_measure = i_measure+1, n_cor-1
               cor_func_B(j_measure)=cor_func_B(j_measure) +
     &              mass(0,i_type)*vel(i_dim)
     &              *stored_vel(i_dim,n_cor+i_measure-j_measure,i_part)
     &              / n_trot
            end do
         end if

        end do
        end do
        i_measure = i_measure + 1

       end if

!----  measure radial correlation function
        if ( (f_potential.ge.2) .and. (mod(i_time,5).eq.0) ) then
!----    chose random trotter time
         call r250(mz,random,n_dim,1,kptr)
         i_trot = int(random(1)*n_trot)
         if(i_trot.eq.n_trot) i_trot = 0
         g_r(1,1,n_grid+1) = g_r(1,1,n_grid+1) + dt
         if (f_potential.eq.9) then
! Scandolo: small system size, count periodic images as well
            pbc_norm=2.0 !*(2.0*n_pbc+1.0)**3
            do i_part = 1,n_part
               do j_part = 1,n_part
                  do pbc_x = -n_pbc, n_pbc
                  do pbc_y = -n_pbc, n_pbc
                  do pbc_z = -n_pbc, n_pbc
                     i_type = type(i_part)
                     j_type = type(j_part)
                     call calc_dist_pbc(r_2,i_trot,i_part,j_part,
     &                    pbc_x,pbc_y,pbc_z)
                     r_1 = sqrt(r_2)
                     i_grid = nint(n_grid*r_1/r_cutoff(i_type,j_type))
                     if(i_grid.le.n_grid)
     &                    g_r(i_type,j_type,i_grid) =
     &                    g_r(i_type,j_type,i_grid)
     &                    + dt/pbc_norm
                  end do
                  end do
                  end do
               end do
            end do
         else
            do i_part = 1,n_part
               i_type = type(i_part)
               do i_neigh = 1,neighbor(0,i_part)
                  j_part = neighbor(i_neigh,i_part)
                  j_type = type(j_part)
                  call calc_distance(r_2,i_trot,i_part,j_part)
                  r_1 = sqrt(r_2)
                  i_grid = nint(n_grid*r_1/r_cutoff(i_type,j_type))
                  if(i_grid.le.n_grid)
     &                 g_r(i_type,j_type,i_grid) =
     &                   g_r(i_type,j_type,i_grid) + dt
               end do
            end do
          end if



        end if

        if ((f_potential.eq.3).or.(
     &       (f_potential.ge.5).and.(f_potential.le.9))) then
          if (f_potential.ne.8) call bond_analysis
          if ((f_symmetry.eq.6).or.(f_symmetry).eq.7) then
         ord_c = ord_c + 1.d0
         ord_par = ord_par - 45.
         ord_par_1 = ord_par_1 + ord_par
         ord_par_2 = ord_par_2 + ord_par**2
         ord_par_4 = ord_par_4 + ord_par**4

         ord_parC_1 = ord_parC_1 + ord_parC
         ord_parC_2 = ord_parC_2 + ord_parC**2
         ord_parC_4 = ord_parC_4 + ord_parC**4
          end if
        end if

        if  (f_potential.eq.9) then
          if ((f_symmetry.eq.6).or.(f_symmetry).eq.7) then

         ord_parB = 0.d0
         do i_part = 1, n_part
            if ((type(7)==2).and.(type(i_part)==2)) then
               i_class = mod(i_part-1,18)-5
              if ((i_class.eq.2).or.(i_class.eq.4).or.(i_class.eq.6).or.
     &             (i_class.eq.8).or.(i_class.eq.9).or.(i_class.eq.12))
     &             then
                 ord_parB = ord_parB + dipole(3,i_part)
!                 write(81,*) i_part, i_class,
!     &                dipole(3,i_part)
              else if
     &          ((i_class.eq.1).or.(i_class.eq.3).or.(i_class.eq.5).or.
     &          (i_class.eq.7).or.(i_class.eq.10).or.(i_class.eq.11))
     &                then
                 ord_parB = ord_parB - dipole(3,i_part)
!                 write(82,*) i_part, i_class,
!     &                dipole(3,i_part)
              else
                 stop "could not assign class to particle"
              end if
            end if
```

```
!            if (type(i_part)==2) then
!               ord_parB = ord_parB + abs(dipole(3,i_part))
!            end if
         end do
         ord_parB = ord_parB / n_oxy

         ordB_c = ordB_c + 1.d0
         ord_parB_1 = ord_parB_1 + ord_parB
         ord_parB_2 = ord_parB_2 + ord_parB**2
         ord_parB_4 = ord_parB_4 + ord_parB**4
        end if
      end if
!     write movie
      if (f_movie.eq.1) then
         do i_part=1, n_part
           if(type(i_part).eq.1) then
             write(46,204) "Si  ",(r_real(i_dim,0,i_part),i_dim=1,n_dim)
           else if(type(i_part).eq.2) then
             write(46,204) "O   ",(r_real(i_dim,0,i_part),i_dim=1,n_dim)
           end if
           write(47,103) (dipole(i_dim,i_part),i_dim=1,n_dim)
         end do
         write(46,*) "END"
         write(47,*) "END"
      end if

      if (f_confmovie.eq.1) then
         if (mod(i_time,10).eq.9) then
            write(48,103) (scal_0(i_dim),i_dim=1,n_dim+3)
            write(48,103) (scal_x(1,i_dim)/dt,i_dim=1,n_dim+3)
            do i_trot = 0,n_trot-1
               do i_part = 1,n_part
                  write(48,141) type(i_part),
     &                 (r0(i_dim,i_trot,i_part),i_dim=1,n_dim)
               end do
            end do
         end if
      end if

 103    format(3e13.4)
 204      format(a,3e13.4)
 141       format(i5,3f14.6)


      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine kihara_bonds
      implicit none
      include "pimd.com.f"
      integer j_dummy, k_part
      double precision r_2, s_2, delta_s(n_dim)
      double precision r_avg2(n_dim),r_avg6(n_dim),r_avg(n_part,n_dim),
     &     r_avg_real(n_part,n_dim)

      integer i,j,k, count

      ! equivalent number pimd and number kihara:
! si1  o6  o1  o2  o5
! 3    10  13  11  8
! 5    15   7  17  14
      ! kihara listed bonds
! o1-si-o2, o1-si-o6, o1-si-o5, o2-si-o5

! si1-o1-si3 = 3 - 13 - 6; 5 - 7 - 2

      count =0
      do i = 1, 3
         r_avg2(i)=0.
         r_avg6(i)=0.
      end do

      do i=0,n_bin_x-1
      do j=0,n_bin_y-1
      do k=0,n_bin_z-1
         r_avg2(1)=r_avg2(1)+(r0(1,0,18*count+2) - i*1./n_bin_x)
         r_avg2(2)=r_avg2(2)+(r0(2,0,18*count+2) - j*1./n_bin_y)
         r_avg2(3)=r_avg2(3)+(r0(3,0,18*count+2) - k*1./n_bin_z)

         r_avg6(1)=r_avg6(1)+(r0(1,0,18*count+6) - i*1./n_bin_x)
         r_avg6(2)=r_avg6(2)+(r0(2,0,18*count+6) - j*1./n_bin_y)
         r_avg6(3)=r_avg6(3)+(r0(3,0,18*count+6) - k*1./n_bin_z)

         count = count +1
      end do
      end do
      end do

      r_avg2(1)=r_avg2(1) / count
      r_avg2(2)=r_avg2(2) / count
      r_avg2(3)=r_avg2(3) / count

      r_avg6(1)=r_avg6(1) / count
      r_avg6(2)=r_avg6(2) / count
      r_avg6(3)=r_avg6(3) / count

      count = 0
      do i=0,n_bin_x-1
      do j=0,n_bin_y-1
      do k=0,n_bin_z-1
         r_avg(18*count+2,1)=r_avg2(1) + i*1./n_bin_x
         r_avg(18*count+2,2)=r_avg2(2) + j*1./n_bin_y
         r_avg(18*count+2,3)=r_avg2(3) + k*1./n_bin_z

         r_avg(18*count+6,1)=r_avg6(1) + i*1./n_bin_x
         r_avg(18*count+6,2)=r_avg6(2) + j*1./n_bin_y
         r_avg(18*count+6,3)=r_avg6(3) + k*1./n_bin_z

         count = count +1
      end do
      end do
      end do


      do i_part = 1,n_part
       do i_dim = 1, n_dim
        r_avg_real(i_part,i_dim) = scal_0(i_dim) *
     &  (r_avg(i_part,i_dim) + 1.d0*pbc0(i_dim,0,i_part))
       end do
       if (e_dim.eq.3) then
        r_avg_real(i_part,1) = r_avg_real(i_part,1) +
     &  scal_0(4) * (r_avg(i_part,2) + 1.d0*pbc0(2,0,i_part)) +
     &  scal_0(6) * (r_avg(i_part,3) + 1.d0*pbc0(3,0,i_part))
```

```
   r_avg_real(i_part,2) = r_avg_real(i_part,2) +
&  scal_0(4) * (r_avg(i_part,1) + 1.d0*pbc0(1,0,i_part)) +
&  scal_0(5) * (r_avg(i_part,3) + 1.d0*pbc0(3,0,i_part))
   r_avg_real(i_part,3) = r_avg_real(i_part,3) +
&  scal_0(6) * (r_avg(i_part,1) + 1.d0*pbc0(1,0,i_part)) +
&  scal_0(5) * (r_avg(i_part,2) + 1.d0*pbc0(2,0,i_part))
  end if
 end do

 do i_part =1, n_part
   if ((mod(i_part,18).eq.3).or.(mod(i_part,18).eq.5)) then
      do i_dummy = 1, 4
         do j_dummy = 1, 4
            j_part = i_neigh_ori(i_part,i_dummy)
            k_part = i_neigh_ori(i_part,j_dummy)
            i=mod(i_part,18)
            j=mod(j_part,18)
            k=mod(k_part,18)
            if ( ((i.eq.3).and.(j.eq.13).and.(k.eq.11)).or.
&                ((i.eq.3).and.(j.eq.13).and.(k.eq.10)).or.
&                ((i.eq.3).and.(j.eq.13).and.(k.eq.8)).or.
&                ((i.eq.3).and.(j.eq.11).and.(k.eq.8)).or.
&                ((i.eq.5).and.(j.eq.7).and.(k.eq.17)).or.
&                ((i.eq.5).and.(j.eq.7).and.(k.eq.15)).or.
&                ((i.eq.5).and.(j.eq.7).and.(k.eq.14)).or.
&                ((i.eq.5).and.(j.eq.17).and.(k.eq.14)) ) then

               i_type = type(i_part)
               call calc_distance(s_2,0,i_part,j_part)
               do i_dim = 1, n_dim
                  delta_s(i_dim) = delta_cell(i_dim)
               end do
               call calc_distance(r_2,0,i_part,k_part)
               r_dummy = 0.
               do i_dim = 1, n_dim
                  r_dummy = r_dummy + met_ten_voi(i_dim) *
&                       delta_s(i_dim) * delta_cell(i_dim)
               end do
               if (e_dim.eq.3) then
                  r_dummy = r_dummy
&                       + met_ten_voi(4) * (
&                       delta_s(1) *delta_cell(2)
&                       + delta_s(2) * delta_cell(1) ) / 2
&                       + met_ten_voi(5) * (
&                       delta_s(2) *delta_cell(3)
&                       + delta_s(3) * delta_cell(2) ) / 2
&                       + met_ten_voi(6) * (
&                       delta_s(3) *delta_cell(1)
&                       + delta_s(1) * delta_cell(3) ) / 2
               end if
               r_dummy = r_dummy / sqrt( r_2 * s_2 )
               r_dummy = acos(r_dummy)

               write(87,'(3i,f14.5)') i,j,k, r_dummy*180./pi
            end if
         end do
      end do
   end if
   if (mod(i_time,15).eq.0) then
   if ((mod(i_part,18).eq.13).or.(mod(i_part,18).eq.7)) then
      do i_dummy = 1, 2
         do j_dummy = 1, 2
            j_part = i_neigh_ori(i_part,i_dummy)
            k_part = i_neigh_ori(i_part,j_dummy)
            i=mod(i_part,18)
            j=mod(j_part,18)
            k=mod(k_part,18)

            if ( ((i.eq.13).and.(j.eq.3).and.(k.eq.6)).or.
&                ((i.eq.7).and.(j.eq.5).and.(k.eq.2)) ) then
               write(88,'(3e16.7)')
&                    r_real(1,0,i_part)-r_real(1,0,k_part),
&                    r_real(2,0,i_part)-r_real(2,0,k_part),
&                    r_real(3,0,i_part)-r_real(3,0,k_part)
               write(89,'(3e16.7)')
&                    r_real(1,0,i_part)-r_avg_real(k_part,1),
&                    r_real(2,0,i_part)-r_avg_real(k_part,2),
&                    r_real(3,0,i_part)-r_avg_real(k_part,3)
            end if
         end do
      end do
     end if
     end if
   end do

   end subroutine



! ! ! ! ! ! ! ! ! ! !

      subroutine bond_analysis
      implicit none
      include "pimd.com.f"
      integer j_dummy, k_part, i_ang, iii, jjj
      double precision dr_neigh, r_2, s_1, s_2, delta_s(n_dim)
      double precision rord_dummy
      double precision rij(3), rik(3)

      call kihara_bonds

      ord_par = 0.d0
      ord_parC = 0.d0
      dr_neigh = (r_neigh_max-r_neigh_min) / n_grid
      neigh_time = neigh_time + 1.
      do i_part = 1, n_part
       i_ang = 0
       i_type = type(i_part)
       if (i_type.eq.1) then
! ---    Si
        do i_dummy = 1, 4
         j_part = i_neigh_ori(i_part,i_dummy)
         call calc_distance(s_2,0,i_part,j_part)
         do i_dim = 1, n_dim
          delta_s(i_dim) = delta_cell(i_dim)
         end do
         s_1 = sqrt(s_2)
         r_dummy = s_1
         i_grid = nint( (r_dummy-r_neigh_min) / dr_neigh )
         if( (i_grid.gt.1) .and. (i_grid.lt.n_grid) )
     &    hist_ang(1,i_grid) = hist_ang(1,i_grid) + 1.d0
        do j_dummy = i_dummy + 1, 4
         i_ang = i_ang + 1
```

```
      if (i_ang.gt.6) stop "more than 6 angles around Si atom"
      k_part = i_neigh_ori(i_part,j_dummy)
      call calc_distance(r_2,0,i_part,k_part)
      r_dummy = 0.
      do i_dim = 1, n_dim
       r_dummy = r_dummy + met_ten_voi(i_dim) *
     &  delta_s(i_dim) * delta_cell(i_dim)
      end do
      if (e_dim.eq.3) then
       r_dummy = r_dummy
     &  + met_ten_voi(4) * (
     &  delta_s(1) *delta_cell(2) + delta_s(2) * delta_cell(1) ) / 2
     &  + met_ten_voi(5) * (
     &  delta_s(2) *delta_cell(3) + delta_s(3) * delta_cell(2) ) / 2
     &  + met_ten_voi(6) * (
     &  delta_s(3) *delta_cell(1) + delta_s(1) * delta_cell(3) ) / 2
      end if
      r_dummy = r_dummy / sqrt( r_2 * s_2 )

! store variables for 3 body potential
      if (f_potential.eq.8) then
          triple_cos(i_part, i_ang) = r_dummy
          triple_partj(i_part, i_ang) = j_part
          triple_rij(i_part, i_ang) = s_1
          triple_partk(i_part, i_ang) = k_part
          triple_rik(i_part, i_ang) = sqrt(r_2)
          do i_dim=1, n_dim
             triple_deltaij(i_part, i_ang, i_dim) = delta_s(i_dim)
             triple_deltaik(i_part, i_ang, i_dim) = delta_cell(i_dim)
          end do
      end if

      r_dummy = acos(r_dummy)

      i_grid = nint(n_grid*r_dummy/pi )
      if( (i_grid.ge.1).and.(i_grid.le.n_grid) )
     &  hist_ang(3,i_grid) = hist_ang(3,i_grid) + 1.d0
      end do
      end do
     else if (i_type.eq.2) then
      o
! ---  i_ang=1
      j_part = i_neigh_ori(i_part,1)
      call calc_distance(s_2,0,i_part,j_part)
      do i_dim = 1, n_dim
       delta_s(i_dim) = delta_cell(i_dim)
      end do
! ----  quartz evaluate ord_param
      if ((f_symmetry.eq.6).or.(f_symmetry.eq.7)) then
       if ( ((type(7).eq.1).and. ! 1/3n Si, 2/3n O Config
     &       ((mod(j_part,6).eq.3).or.(mod(j_part,6).eq.4))).or.
     &      ((type(7).eq.2).and. ! 6 Si, 12 O, 6 Si, 12 O config
     &       ((mod(j_part,18).eq.1).or.(mod(j_part,18).eq.4))) )
     &       then
        ord_dummy = delta_cell(2) /
     &  sqrt( delta_cell(2)**2 + delta_cell(3)**2 )
        if ((delta_cell(2).gt.0.) then
         ord_dummy = acos(ord_dummy)
        else
         ord_dummy = asin(ord_dummy)
        end if
        ord_par = i_neigh_ori(i_part,2)
        call calc_distance(r_2,0,i_part,k_part)
        r_dummy = 0.
        do i_dim = 1, n_dim
         r_dummy = r_dummy + met_ten_voi(i_dim) *
     &   delta_s(i_dim) * delta_cell(i_dim)
        end do
! ----  quartz evaluate ord_param
        if ((f_symmetry.eq.6).or.(f_symmetry.eq.7)) then
         if ( ((type(7).eq.1).and. ! 1/3n Si, 2/3n O Config
     &         ((mod(k_part,6).eq.3).or.(mod(k_part,6).eq.4))).or.
     &        ((type(7).eq.2).and. ! 6 Si, 12 O, 6 Si, 12 O config
     &         ((mod(k_part,18).eq.1).or.(mod(k_part,18).eq.4))) )
     &         then
          ord_dummy = delta_cell(2) /
     &    sqrt( delta_cell(2)**2 + delta_cell(3)**2 )
          if (delta_cell(3).gt.0.) then
           ord_dummy = acos(ord_dummy)
          else
           ord_dummy = asin(ord_dummy)
          end if
          ord_par = ord_par + rord_dummy
          ord_parc = ord_parc + abs(rord_dummy*180./pi - 45.)
          write(44,*) i_part, type(i_part), k_part, type(k_part),
     &      mod(k_part,18), 180*rord_dummy/pi-45
         end if
         end if
         end if

         if (e_dim.eq.3) then
          r_dummy = r_dummy
     &  + met_ten_voi(4) * (
     &  delta_s(1) *delta_cell(2) + delta_s(2) * delta_cell(1) ) / 2
     &  + met_ten_voi(5) * (
     &  delta_s(2) *delta_cell(3) + delta_s(3) * delta_cell(2) ) / 2
     &  + met_ten_voi(6) * (
     &  delta_s(3) *delta_cell(1) + delta_s(1) * delta_cell(3) ) / 2
         end if
         r_dummy = r_dummy / sqrt( r_2 * s_2 )

! store variables for 3 body potential
         if (f_potential.eq.8) then
             triple_cos(i_part, i_ang) = r_dummy
             triple_partj(i_part, i_ang) = j_part
             triple_rij(i_part, i_ang) = sqrt(s_2)
             triple_partk(i_part, i_ang) = k_part
             triple_rik(i_part, i_ang) = sqrt(r_2)
             do i_dim=1, n_dim
                triple_deltaij(i_part, i_ang, i_dim) = delta_s(i_dim)
                triple_deltaik(i_part, i_ang, i_dim) = delta_cell(i_dim)
             end do
         end if

         r_dummy = acos(r_dummy)
```

```
! write vector from middle of si-si connection line to o

        do iii=1,3
           rij(iii)=0.
           rik(iii)=0.
           do jjj=1,3
              rij(iii)=rij(iii)+h_mat(iii,jjj)*delta_s(jjj)
              rik(iii)=rik(iii)+h_mat(iii,jjj)*delta_cell(jjj)
           end do
        end do

        if (i_part.eq.15)
     &    write(86,'(3i,3f14.5)') i_part,j_part,k_part,
     &       (0.5*(rij(iii)+rik(iii)),iii=1,3)


        i_grid = nint(n_grid*r_dummy/pi )
        if( (i_grid.gt.0).and.(i_grid.lt.n_grid) )
     &  hist_ang(2,i_grid) = hist_ang(2,i_grid) + 1.d0
       end if
      end do
      if ((f_symmetry.eq.6).or.(f_symmetry.eq.7)) then
       ord_par = 180 * ord_par / pi
       ord_par = 1.5 * ord_par / n_silic
       ord_parC = 1.5 * ord_parC / n_silic
      end if

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine viri_estimator
      implicit none
      include "pimd.com.f"
      double precision r_local(n_dim),r_help(n_dim),f_local(n_dim)

!---- change transform
      viri_est = 0.
!---- term meas viri_est
      do i_part = 1,n_part
      do i_trot = 0,n_trot-1
       do i_dim = 1,n_dim
        r_local(i_dim) = 0.
        f_local(i_dim) = 0.
        r_help(i_dim) = r0(i_dim,i_trot,i_part)
     &  + 1. * pbc0(i_dim,i_trot,i_part) - r_ref(i_dim,i_part)
       end do
       do i_dim = 1,n_dim
       do j_dim = 1,n_dim
        r_local(i_dim) = r_local(i_dim) +
     &  h_mat(i_dim,j_dim) * r_help(j_dim)
        f_local(i_dim) = f_local(i_dim) +
     &  h_mat(i_dim,j_dim) * ( force(j_dim,i_trot,i_part) )
!    &  0 * mass(i_dummy,i_type) * friction(i_type) * r_dummy**2
!    &  * (ux(1,j_dim,i_trot,i_part)/dt) )
       end do
! 666 lines above
       end do
       do i_dim = 1, n_dim
        viri_est = viri_est + r_local(i_dim) * f_local(i_dim)
       end do
      end do
      end do
      viri_est = -viri_est/2

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine prim_estimator
      implicit none
      include "pimd.com.f"

      v_chain = 0.
      do i_trot = 0, n_trot-1
       j_trot = mod(i_trot+1,n_trot)
       do i_part = 1, n_part
        i_type = type(i_part)
!----   TERM meas prim_est
       do i_dim = 1, n_dim
        v_chain = v_chain + k_quant(i_type) * (
     &   r_real(i_dim,i_trot,i_part) - r_real(i_dim,j_trot,i_part) )**2
       end do
      end do
      end do
      v_chain = v_chain / 2

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine obser_out
      implicit none
      include "pimd.com.f"
      integer number_spec(n_type), i_class
      double precision norm_spec(n_type,n_type)

       do i_type = 1,n_type
        number_spec(i_type) = 0
       end do
       do i_part = 1,n_part
        i_type = type(i_part)
        number_spec(i_type) = number_spec(i_type) + 1
       end do

      write(20,*)
      write(20,201) time_obs,"   time observed"

      t_dynamic_1 = t_dynamic_1/time_obs
      t_dynamic_2 = t_dynamic_2/time_obs  - t_dynamic_1**2
      t_dynamic_1 = t_dynamic_1 / (n_part*n_trot)
      t_dynamic_2 = t_dynamic_2 / (n_part*n_trot*tkp**2)
      write(20,*)
      write(20,202) t_dynamic_1, t_dynamic_2,"   dynamic kin. energy"

      t_diag_1 = t_diag_1/time_obs
      t_ndiag_1 = t_ndiag_1/time_obs
      write(20,202) t_diag_1, t_ndiag_1,"   diag and non_diag T_kin"

      v_inter_1 = v_inter_1 / time_obs
      v_inter_2 = v_inter_2 / time_obs - v_inter_1**2
      v_inter_1 = v_inter_1 / (n_part*n_trot)
      v_inter_2 = v_inter_2 / (n_part*n_trot*tkp**2)
      write(20,*)
```

```
 write(20,202) v_inter_1, v_inter_2,"   interaction energy"

 viri_est_1 = viri_est_1 / time_obs
 viri_est_2 = viri_est_2 / time_obs - viri_est_1**2
 viri_est_1 = viri_est_1 / (n_part*n_trot)
 viri_est_2 = viri_est_2 / (n_part*n_trot*tkp**2)
 write(20,*)
 write(20,202) viri_est_1, viri_est_2,"   virial estimator"

 prim_est_1 = prim_est_1 / time_obs
 prim_est_2 = prim_est_2 / time_obs - prim_est_1**2
 prim_est_1 = prim_est_1 / (n_part*n_trot)
 prim_est_2 = prim_est_2 / (n_part*n_trot*tkp**2)
 write(20,202) prim_est_1, prim_est_2,"   Primitive energy"

 if ((f_symmetry.eq.6).or.(f_symmetry.eq.7)) then
  ord_par_1 = ord_par_1 / ord_c
  ord_par_2 = ord_par_2 / ord_c
  ord_par_4 = ord_par_4 / ord_c
  write(20,*) "ord_par delta_phi", ord_par_1, ord_par_2, ord_par_4
  ord_par_2 = ord_par_2 - ord_par_1**2
  ord_par_2 = ord_par_2 / tk
  write(20,*) "sussz.", ord_par_2
  write(20,*)
  ord_parB_1 = ord_parB_1 / ordB_c
  ord_parB_2 = ord_parB_2 / ordB_c
  ord_parB_4 = ord_parB_4 / ordB_c
  write(20,*) "ord_par mu_z", ord_parB_1, ord_parB_2, ord_parB_4
  ord_parB_2 = ord_parB_2 - ord_parB_1**2
  ord_parB_2 = ord_parB_2 / tk
  write(20,*) "sussz.", ord_parB_2
  write(20,*)
  ord_parC_1 = ord_parC_1 / ord_c
  ord_parC_2 = ord_parC_2 / ord_c
  ord_parC_4 = ord_parC_4 / ord_c
  write(20,*) "ord_par mu_z", ord_parC_1, ord_parC_2, ord_parC_4
  ord_parC_2 = ord_parC_2 - ord_parC_1**2
  ord_parC_2 = ord_parC_2 / tk
  write(20,*) "sussz.", ord_parC_2
 end if

 if (f_potential.eq.9) then
    do i_dim=0,3
       do i_class = 1,12
           dipole_class1(i_dim,i_class)=
&             dipole_class1(i_dim,i_class)/dipole_count
        end do
     end do
     write(20,*)
     do i_class=1,12
     write(20,209) (dipole_class1(i_dim,i_class),i_dim=0,3),
&          "    oxygen type ",i_class
     end do
  end if

 if (f_potential.gt.1) then
  vol_1 = vol_1 / time_obs
  vol_2 = vol_2 / time_obs
  vol_2 = vol_2 - vol_1**2
  vol_2 = (vol_2/vol_1) / tk
  write(20,*)
  write(20,202) vol_1/n_part,vol_2,"   volume"
```

```
   do i_dim = 1, n_dim + 3
    scal_1(i_dim) = scal_1(i_dim) / time_obs
   end do
   do i_dim = 1, n_dim + 3
    do j_dim = 1, n_dim + 3
     scal_2(j_dim,i_dim) = scal_2(j_dim,i_dim) / time_obs
     scal_2(j_dim,i_dim) = ( scal_2(j_dim,i_dim) -
&    scal_1(j_dim) * scal_1(i_dim) ) / tk
    end do
   end do

   r_dummy = scal_1(1)*scal_1(2)*scal_1(3) +
& 2*scal_1(4)*scal_1(5)*scal_1(6) - scal_1(4)**2*scal_1(3)
& - scal_1(5)**2*scal_1(1) - scal_1(6)**2*scal_1(2)
   write(20,202) r_dummy/n_part

   write(20,*)
   if (f_potential.gt.1) then
    write(20,*) "scal. variable"
    write(20,206) (scal_1(i_dim), i_dim=1, n_dim+3)
    if (f_pressure.eq.1) then
     write(20,*)
     do i_dim = 1, n_dim + 3
      write(20,206) (scal_2(i_dim,j_dim), j_dim=1, n_dim+3)
     end do
    end if
   end if
  end if

  if ( (f_potential.gt.1) .and. (f_pressure.eq.0) ) then
   write(20,*)
   write(20,*) "elastic constants"
   write(20,*)
   do i_dim = 1, n_dim
   do j_dim = 1, n_dim
    strain_1(i_dim,j_dim) = strain_1(i_dim,j_dim) / time_born
   end do
   end do
   do i_dim = 1, n_dim
   do j_dim = i_dim, n_dim
    do k_dim = i_dim, n_dim
    do l_dim = k_dim, n_dim
     if (i_dim+j_dim.le.k_dim+l_dim) then
      ela_born_1(i_dim, j_dim, k_dim, l_dim) =
&     ela_born_1(i_dim, j_dim, k_dim, l_dim) / time_born
      strain_2(i_dim, j_dim, k_dim, l_dim) =
&     strain_2(i_dim, j_dim, k_dim, l_dim) / time_born
      strain_2(i_dim, j_dim, k_dim, l_dim) =
&     strain_2(i_dim, j_dim, k_dim, l_dim) -
&     strain_1(i_dim, j_dim) * strain_1(k_dim, l_dim)
      strain_2(i_dim, j_dim, k_dim, l_dim) = -beta*
&     strain_2(i_dim, j_dim, k_dim, l_dim) / vol_1
      write(20,'(4i2,3e14.4)')
&     i_dim, j_dim, k_dim, l_dim,
&     ela_born_1(i_dim, j_dim, k_dim, l_dim),
&     strain_2(i_dim, j_dim, k_dim, l_dim),
&     ela_born_1(i_dim, j_dim, k_dim, l_dim) +
&     strain_2(i_dim, j_dim, k_dim, l_dim)
     end if
    end do
    end do
```

```
  end do
  end do
  write(20,*)
  write(20,*) "strain"
  write(20,*)
  do i_dim = 1, n_dim
  do j_dim = 1, n_dim
   write(20,'(2i2,1f10.4)') i_dim, j_dim,
&  strain_1(i_dim, j_dim)/ (n_part*n_trot)
  end do
  end do
 end if

 if (i_run.eq.n_run) close(20)

 if (n_run.ge.2) write(71,*) tk, v_inter_1, viri_est_1,
&prim_est_1, vol_1/n_part, v_inter_2, vol_2

 if(i_run.eq.1) open(21,file='pimd.ima',status='unknown')
 do i_trot = 0,n_trot-1
  write(21,'(2e15.5)') i_trot*betap,
& cor_ifunc(i_trot)/count_ifunc/n_part
 end do
 write(21,'(2e15.5)') beta,cor_ifunc(0)/count_ifunc/n_part
 if(i_run.eq.n_run) close(21)

 open(22,file='pimd.cor.SiO',status='unknown')
 open(23,file='pimd.cor.q',status='unknown')
 open(25,file='pimd.cor_B',status='unknown')
 open(26,file='pimd.cor_A',status='unknown')

 do i_cor = 0,i_measure-1
  do i_type=1, n_type
     cor_func(i_cor,i_type) = 2.*cor_func(i_cor,i_type)/
&         cor_func(n_cor,1)/number_spec(i_type)
  end do
  qdot_cor(i_cor) = 2.*qdot_cor(i_cor)/qdot_cor(n_cor)/n_part
  cor_func_A(i_cor) =2.*cor_func_A(i_cor)
&      /cor_func_A(n_cor)/n_part
  cor_func_B(i_cor) =2.*cor_func_B(i_cor)/n_part/cor_func_B(n_cor)
  write(22,'(3e15.5)') ni_time*i_cor*dt/di_time,
&       (cor_func(i_cor,i_type),i_type=1,n_type)
  write(23,'(2e15.5)') ni_time*i_cor*dt/di_time,qdot_cor(i_cor)
  write(25,'(2e15.5)') ni_time*i_cor*dt/di_time,cor_func_B(i_cor)
  write(26,'(2e15.5)') ni_time*i_cor*dt/di_time,cor_func_A(i_cor)
 end do
 if(i_measure+1.lt.n_cor) then
  do i_cor = i_measure,n_cor-1
   do i_type=1, n_type
      cor_func(i_cor,i_type) = 2.*cor_func(i_cor,i_type)/
&          (cor_func(n_cor,1)-1.)/number_spec(i_type)
   end do
   qdot_cor(i_cor) = 2.*qdot_cor(i_cor)/(qdot_cor(n_cor)-1.)/n_part
  cor_func_A(i_cor) =2.*cor_func_A(i_cor)/(cor_func_A(n_cor)-1.)
&       /n_part
  cor_func_B(i_cor) =2.*cor_func_B(i_cor)/n_part/cor_func_B(n_cor)
   write(22,'(3e15.5)') ni_time*i_cor*dt/di_time,
&       (cor_func(i_cor,i_type),i_type=1,n_type)
   write(23,'(2e15.5)') ni_time*i_cor*dt/di_time,qdot_cor(i_cor)
   write(25,'(2e15.5)') ni_time*i_cor*dt/di_time,cor_func_B(i_cor)
   write(26,'(2e15.5)') ni_time*i_cor*dt/di_time,cor_func_A(i_cor)
  end do
      end if
      if(i_run.eq.n_run) close(22)
      if(i_run.eq.n_run) close(23)

      if(f_potential.gt.1) then
       if(i_run.eq.1) open(24,file='pimd.g_r',status='unknown')
!----  count number of particles for given types
       do i_type = 1,n_type
       do j_type = 1,n_type
!----   define normalization factor
        if(number_spec(i_type)*number_spec(j_type).ne.0) then
         norm_spec(i_type,j_type) =  real(n_part) /
     &   (number_spec(i_type)*number_spec(j_type))
        else
         norm_spec(i_type,j_type) = 0.
        end if
        norm_spec(i_type,j_type) =
     &  2*(norm_spec(i_type,j_type)/(n_part/vol_1))*
     &  (real(n_grid)/r_range(i_type,j_type))/g_r(1,1,n_grid+1)
!----   symmetrize to yield better statistics
        if(i_type.lt.j_type) then
         do i_grid = 0,n_grid
          g_r(i_type,j_type,i_grid) =
     &    ( g_r(i_type,j_type,i_grid) + g_r(j_type,i_type,i_grid) ) / 2
          g_r(j_type,i_type,i_grid) = g_r(i_type,j_type,i_grid)
         end do
        end if
!----   apply normalization to g(r)
        do i_grid = 0, n_grid
         g_r(i_type,j_type,i_grid) = g_r(i_type,j_type,i_grid)
     &   * norm_spec(i_type,j_type)
        end do
       end do
       end do

!----  format r,g(r)/4{\pi}r**2
       do i_grid = 1, n_grid
        write(24,'(6e12.4)')
     &    i_grid*r_cutoff(1,1)/n_grid,g_r(1,1,i_grid)
     &  / (4*pi*(i_grid*r_cutoff(1,1)/n_grid)**2) ,
     &    i_grid*r_cutoff(1,2)/n_grid,g_r(1,2,i_grid)
     &  / (4*pi*(i_grid*r_cutoff(1,2)/n_grid)**2) ,
     &    i_grid*r_cutoff(2,2)/n_grid,g_r(2,2,i_grid)
     &  / (4*pi*(i_grid*r_cutoff(2,2)/n_grid)**2)
       end do
       if(i_run.eq.n_run) close(24)
      end if

      if (f_potential.eq.3) call bond_out
      if (f_potential.eq.5) call bond_out
      if (f_potential.eq.6) call bond_out
      if (f_potential.eq.7) call bond_out
      if (f_potential.eq.8) call bond_out
      if (f_potential.eq.9) call bond_out

  201 format(1e15.7,a)
  202 format(2e15.7,a)
  203 format(3e15.7,a)
  204 format(4e15.7,a)
  206 format(6e13.5,a)
 209  format(4e13.5,a,i3)
```

```
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine bond_out
      implicit none
      include "pimd.com.f"
      double precision dr_neigh
      dr_neigh = (r_neigh_max-r_neigh_min) / n_grid
      open(41,file='pimd.bon',status='unknown')
      do i_grid = 1, n_grid
       r_dummy = r_neigh_min + i_grid * dr_neigh
       write(41,'(5e15.6)') r_dummy,
     & hist_ang(1,i_grid) / (dr_neigh*n_silic*neigh_time) / r_dummy**2,
     & i_grid * 180. / n_grid,
     & n_grid * hist_ang(2,i_grid) / (180. * n_oxy * neigh_time), !  Si-O-Si
     & n_grid * hist_ang(3,i_grid) / (6 * 180. * n_silic * neigh_time) ! O-Si-O
      end do
      close(41)
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine check_skin
      implicit none
      include "pimd.com.f"

      integer f_update(0:n_trot-1,n_part)
      common /cf_update/ f_update

      do i_part = 1,n_part
       i_type = type(i_part)
       do i_trot = 0,n_trot-1
        r_dummy = 0.
        f_update(i_trot,i_part) = 0
        do i_dim = 1,n_dim
         delta_r(i_dim) = r_real(i_dim,i_trot,i_part)
     &    - r0_old(i_dim,i_trot,i_part)
         r_dummy = r_dummy + delta_r(i_dim)**2
        end do
        if(r_dummy.gt.(0.5*r_skin(i_type,i_type))**2) then
         f_skin = 1
         f_update(i_trot,i_part) = 1
        end if
       end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine predict
      implicit none
      include "pimd.com.f"
      double precision loc_force(n_dim)

!---- propagate only independent parts

      do j_trot = 0,n_trot-1
       i_trot = j_trot + 1
       if(i_trot.eq.2) i_trot = n_trot + 1

!----  first positions
       do j_order = 1,n_order
       do i_dim = 1,n_dim
       do i_part = 1,n_part
        u0(i_dim,i_trot,i_part) = u0(i_dim,i_trot,i_part) +
     &  predict_coef(0,j_order) * ux(j_order,i_dim,i_trot,i_part)
       end do
       end do
       end do

!----  add random force to positions and velocities
       i_dummy = (i_trot-1)/2
       do i_part = 1,n_part
        i_type = type(i_part)
!----   term stoch force
        do i_dim = 1, n_dim
         loc_force(i_dim) = 0.
         do j_dim = 1, n_dim
          loc_force(i_dim) = loc_force(i_dim) +
     &    h_mat_inv(i_dim,j_dim) * fr(j_dim,i_trot,i_part)
         end do
        end do
        do i_dim = 1,n_dim
         u0(i_dim,i_trot,i_part) = u0(i_dim,i_trot,i_part) +
     &   loc_force(i_dim) * dt_2 / 2
     &   * rf_width(i_dummy,i_type)
         ux(1,i_dim,i_trot,i_part) = ux(1,i_dim,i_trot,i_part) +
     &   loc_force(i_dim) * dt_2 * rf_width(i_dummy,i_type)
        end do
       end do

!----  then derivatives
       do i_order = 1, n_order-1
       do j_order = i_order + 1, n_order
       do i_dim = 1,n_dim
       do i_part = 1,n_part
        ux(i_order,i_dim,i_trot,i_part) =
     &  ux(i_order,i_dim,i_trot,i_part) +
     &  predict_coef(i_order,j_order) * ux(j_order,i_dim,i_trot,i_part)
       end do
       end do
       end do
       end do

      end do

      call symmetrize

! 667  continue

!---  predict box shape
      if(f_pressure.eq.1) then

       do j_order = 1, n_order
       do i_dim = 1,n_dim + e_dim
        scal_0(i_dim) = scal_0(i_dim) +
     &  predict_coef(0,j_order) * scal_x(j_order,i_dim)
       end do
       end do

       do i_dim = 1, n_dim + e_dim
        if (i_dim.le.n_dim) then
```

```
          scal_0(i_dim) = scal_0(i_dim)
     &    + rfb_width * fr_box(i_dim) * dt_2 / 2
          scal_x(1,i_dim) = scal_x(1,i_dim)
     &    + rfb_width * fr_box(i_dim) * dt_2
         else
          scal_0(i_dim) = scal_0(i_dim)
     &    + rfb_width * fr_box(i_dim) * dt_2 / 2 / sqrt2
          scal_x(1,i_dim) = scal_x(1,i_dim)
     &    + rfb_width * fr_box(i_dim) * dt_2 / sqrt2
         end if
        end do

        do i_order = 1, n_order-1
        do j_order = i_order + 1, n_order
        do i_dim = 1, n_dim + e_dim
         scal_x(i_order,i_dim) = scal_x(i_order,i_dim)
     &  + predict_coef(i_order,j_order) * scal_x(j_order,i_dim)
        end do
        end do
        end do

        call scal_0_update

       end if

       if (i_time.lt.4000) then
        i_type = type(1)
        j_type = type(n_part)
        write(51,'(10e16.7)') r_time,
     & ux(1,1,1,1) / dt / v_therm(0,i_type),
     & ux(1,1,2*n_trot-1,1) / dt / v_therm(n_trot-1,i_type),
     & ux(1,1,1,n_part) / dt / v_therm(0,j_type),
     & ux(1,1,2*n_trot-1,n_part) / dt / v_therm(n_trot-1,j_type),
     & volume / n_part, 1.*i_type, 1.*j_type
       else if (i_time.eq.4000) then
        close(51)
       end if

       end subroutine

! ! ! ! ! ! ! ! ! ! !

       subroutine correct
       implicit none
       include "pimd.com.f"
       double precision cor_max,cor_max_min,cor_max_max
       integer num_part(n_type)
       double precision avg_charge(n_type), max_charge(n_type),
     &     min_charge(n_type)
       double precision fstress(3,3)
       integer ki_dim

!---   change
!      parameter (cor_max_min=2.e-09,cor_max_max=1.e-06)
       parameter (cor_max_min=1.e-09,cor_max_max=0.5e-06)
       integer cor_max_count
       common /ccor_max_count/ cor_max_count
       integer failure(3)

       t_dynamic = 0.d0

       if(i_time.le.1) cor_max_count = 0


!---   here, f0 has meaning of difference between predicted
!      acceleration and measured acceleration

       cor_max = 0.
       do i_part = 1,n_part
        i_type = type(i_part)
        do j_trot = 0,n_trot-1
         i_trot = j_trot + 1
         if(i_trot.eq.2) i_trot = n_trot + 1
         i_dummy = (i_trot-1)/2
         do i_dim = 1,n_dim
          f0(i_dim,i_trot,i_part) = -ux(2,i_dim,i_trot,i_part)
     &    + f0(i_dim,i_trot,i_part)/mass(i_dummy,i_type)*dt_2 / 2
          r_dummy = (f0(i_dim,i_trot,i_part)*correct_coef(1)*dt/
     &    v_therm(i_dummy,i_type))**2
!---      change
          if(r_dummy.gt.cor_max) then
           cor_max = r_dummy
           failure(1) = i_part
           failure(2) = i_trot
           failure(3) = i_dim
          end if
          cor_max = max(cor_max,r_dummy)
         end do
        end do
       end do
       if (cor_max.gt.cor_max_max) then
        writE(*,'(1e12.4,3i5)') cor_max, failure
        stoP "monomer move"
       end if

!---- change

!      goto 667

       do j_trot = 0,n_trot-1
        i_trot = j_trot + 1
        if(i_trot.eq.2) i_trot = n_trot + 1
        do i_part = 1,n_part
        do i_dim = 1,n_dim
         u0(i_dim,i_trot,i_part) = u0(i_dim,i_trot,i_part)
     &  + correct_coef(0) * f0(i_dim,i_trot,i_part)
        end do
        end do
        do i_order = 1,n_order
        do i_part = 1,n_part
        do i_dim = 1,n_dim
         ux(i_order,i_dim,i_trot,i_part) =ux(i_order,i_dim,i_trot,i_part)
     &  + correct_coef(i_order) * f0(i_dim,i_trot,i_part)
        end do
        end do
        end do
       end do

       call symmetrize

!  667 continue

       t_dynamic = 0.d0
!---- term meas dynamic
       do i_part = 1,n_part
```

```
      i_type = type(i_part)
      do i_trot = 0,n_trot-1
       do i_dim = 1,n_dim
        force(i_dim,i_trot,i_part) = 0.d0
        t_dynamic = t_dynamic +
     &   mass(i_trot,i_type) * met_ten_voi(i_dim) *
     &   (ux(1,i_dim,2*i_trot+1,i_part)**2 +
     &    ux(1,i_dim,2*i_trot+2,i_part)**2 )
       end do
       if (e_dim.eq.3) then
        t_dynamic = t_dynamic
     &   + mass(i_trot,i_type) * met_ten_voi(4) *
     &    ( ux(1,1,2*i_trot+1,i_part) * ux(1,2,2*i_trot+1,i_part)
     &    + ux(1,1,2*i_trot+2,i_part) * ux(1,2,2*i_trot+2,i_part) )
     &   + mass(i_trot,i_type) * met_ten_voi(5) *
     &    ( ux(1,2,2*i_trot+1,i_part) * ux(1,3,2*i_trot+1,i_part)
     &    + ux(1,2,2*i_trot+2,i_part) * ux(1,3,2*i_trot+2,i_part) )
     &   + mass(i_trot,i_type) * met_ten_voi(6) *
     &    ( ux(1,3,2*i_trot+1,i_part) * ux(1,1,2*i_trot+1,i_part)
     &    + ux(1,3,2*i_trot+2,i_part) * ux(1,1,2*i_trot+2,i_part) )
       end if
      end do
     end do
     t_dynamic = t_dynamic / (2*dt_2)

     if (f_pressure.eq.1) then
!----  correct box shape
      do i_dim = 1, n_dim + e_dim
       force_scal(i_dim) = - scal_x(2,i_dim)
     &  + force_scal(i_dim) / mass_box * dt_2 / 2
       r_dummy = ( force_scal(i_dim)*correct_coef(1)*dt / w_therm )**2
       if (r_dummy.gt.100*cor_max_max) then
        writE(*,*) r_dummy
        stoP "volume move"
       end if
      end do
      do i_dim = 1, n_dim + e_dim
       scal_0(i_dim) = scal_0(i_dim) +
     &  correct_coef(0) * force_scal(i_dim)
      end do

      do i_order = 1,n_order
      do i_dim = 1,n_dim + e_dim
       scal_x(i_order,i_dim) = scal_x(i_order,i_dim)
     &   + correct_coef(i_order) * force_scal(i_dim)
      end do
      end do

      t_diag = 0.
      t_ndiag = 0.
!----  term meas box dyn
      do i_dim = 1, n_dim + e_dim
       if (i_dim.le.n_dim) t_diag = t_diag
     &  + mass_box * scal_x(1,i_dim)**2
       if (i_dim.gt.n_dim) t_ndiag = t_ndiag
     &  + 2 * mass_box * scal_x(1,i_dim)**2
      end do
      t_diag = t_diag / (2*dt_2)
      t_ndiag = t_ndiag / (2*dt_2)
      t_dynamic = t_dynamic + t_diag + t_ndiag
     end if
```

```
      do i_type = 1, n_type
         avg_charge(i_type)=0
         num_part(i_type)=0
      end do
      do i_part = 1, n_part
         i_type = type(i_part)
         avg_charge(i_type)=avg_charge(i_type)+charge(i_part)
         num_part(i_type)=num_part(i_type)+1
      end do
      do i_type = 1, n_type
         avg_charge(i_type)=avg_charge(i_type)/num_part(i_type)
      end do

      pol_total = 0.d0
      pol_dipoles = 0.d0
      do i_part = 1, n_part
        do i_dim =1, n_dim
         pol_total(i_dim) = pol_total(i_dim) + r_real(i_dim,0,i_part)
     &       * charge(i_part)
         pol_dipoles(i_dim) = pol_dipoles(i_dim) + dipole(i_dim,i_part)
        end do
      end do
      pol_total = pol_total + pol_dipoles

      do i_dim=1, 3
         do j_dim = 1,3
            fstress(i_dim,j_dim)=0.d0
            do k_dim = 1,3
               if (i_dim.eq.k_dim) then
                  ki_dim = i_dim
               else if (i_dim*k_dim.eq.2) then
                  ki_dim = 4
               else if (i_dim*k_dim.eq.6) then
                  ki_dim = 5
               else if (i_dim*k_dim.eq.3) then
                  ki_dim = 6
               end if
               fstress(i_dim,j_dim)=fstress(i_dim,j_dim) +
     &              h_mat(j_dim,k_dim)*force_scal_inter(ki_dim)
     &              /volume
            end do
         end do
      end do

!     goto 101
      if (mod(i_time,1).eq.0) then
       call prim_estimator
      write(60,'(36e15.7)') r_time,
!       if (mod(i_time,10).eq.9) write(60,'(30e15.7)') r_time,
     2 (t_dynamic+v_chain+v_inter+t_dynamic_charge+v_charge+
     &      v_press-v_gauge)/n_part/n_trot,
     3 t_dynamic/n_part/n_trot,
     4 v_chain/n_part/n_trot,
     5 (v_inter-v_gauge)/n_part/n_trot,
     6 v_press/n_part/n_trot,
     7 v_ewald/n_part/n_trot,
     8 volume/n_part,
     9 (scal_0(i_dim),i_dim=1,n_dim+e_dim),       ! col  9-14
     & v_realcoulomb/n_part, v_shortrange/n_part, ! col 15+16
     & v_threebody/n_part, 0.,                    ! col 17+18
     & self_energy/n_part, v_qeqself/n_part,      ! col 19+20
     & (avg_charge(i_type),i_type=1,n_type),      ! col 21+22
```

```
     & t_dynamic_charge/n_part/n_trot,            ! col 23
     & v_charge/n_part/n_trot,                    ! col 24
     & (pol_total(i_dim),i_dim=1,3),              ! col 25-27
     & (pol_dipoles(i_dim),i_dim=1,3),            ! col 28-30
     & fstress(1,1),fstress(2,2),fstress(3,3),    ! col 31-33
     & 0.5*(fstress(1,2)+fstress(2,1)),           ! col 34
     & 0.5*(fstress(2,3)+fstress(3,2)),           ! col 35
     & 0.5*(fstress(1,3)+fstress(3,1))            ! col 36
      else if(i_time.eq.n_time_scal*4000) then
       close(60)
      end if

       if( (mod(i_time,10).eq.9) .and.
     &     ((f_symmetry.eq.6).or.(f_symmetry.eq.7))  )
     &      write(61,'(4f12.6)') r_time, ord_par, ord_parB, ord_parC

      if (mod(i_time,10).eq.9) then
         do i_type = 1, n_type
            avg_charge(i_type)=0
            max_charge(i_type)=-1.e30
            min_charge(i_type)=1.e30
         end do
         do i_part = 1, n_part
            i_type = type(i_part)
            avg_charge(i_type)=avg_charge(i_type)+charge(i_part)
            if (charge(i_part).gt.max_charge(i_type))
     &           max_charge(i_type)=charge(i_part)
            if (charge(i_part).lt.min_charge(i_type))
     &        min_charge(i_type)=charge(i_part)
         end do
         avg_charge(1)=avg_charge(1)/n_silic
         avg_charge(2)=avg_charge(2)/n_oxy

         write(62,'(20e15.7)') r_time,
     2        avg_charge(1),
     3        min_charge(1),
     4        max_charge(1),
     5        avg_charge(2),
     6        min_charge(2),
     7        max_charge(2)
      end if


! 101  continue

      ieff_time = ieff_time + di_time

!---- increase time step only if on multiple integers of ni_time
      if(cor_max.lt.cor_max_min) then
       if(mod(ieff_time,ni_time).ne.0) cor_max = 2*cor_max_min
      end if

!---- change
      goto 100

!---- make sure that time step doesn't become too large/small
      if((cor_max.lt.cor_max_min).and.(di_time.eq.ni_time)) then
       cor_max = 2*cor_max_min
      else if((cor_max.gt.cor_max_max).and.(di_time.eq.1)) then
       stop "Time correl can not be measured any longer."
      end if
```

```
!---- make sure ieff_time is not too big
      if(ieff_time.le.0) stop "ieff_time is infty"

!---  change
!     write(61,*) r_time,dt,cor_max,ieff_time,failure

!----  change
!      include width of box defo moves
      if(cor_max.gt.cor_max_max) then
!----  decrease time step
       cor_max_count = 1
!----  change
!      if(i_time.lt.10000) write(61,*) r_time,dt,cor_max,ieff_time
       dt = dt/2
       di_time = di_time/2
!----  change
!      if(i_time.lt.10000) write(61,*) r_time,dt,cor_max,ieff_time
       dt_2 = dt**2
       do i_type = 1,n_type
       do i_trot = 0,n_trot-1
        rf_width(i_trot,i_type) = rf_width(i_trot,i_type) * 2.d0**0.5d0
       end do
       end do
       do i_order = 1,n_order
        r_dummy = 1./2**i_order
!       if(i_order.gt.2) r_dummy = 0.
        do i_part = 1,n_part
        do i_trot = 0,n_trot-1
        do i_dim = 1,n_dim
         ux(i_order,i_dim,2*i_trot+1,i_part) =
     &   ux(i_order,i_dim,2*i_trot+1,i_part) * r_dummy
         ux(i_order,i_dim,2*i_trot+2,i_part) =
     &   ux(i_order,i_dim,2*i_trot+2,i_part) * r_dummy
        end do
        end do
        end do
        do i_dim = 1,n_dim
         scal_x(i_order,i_dim) = scal_x(i_order,i_dim) * r_dummy
        end do
       end do
      else if(cor_max.lt.cor_max_min) then
!----  increase time step
       cor_max_count = 1
!----  change
!      if(i_time.lt.10000) write(61,*) r_time,dt,cor_max,ieff_time
       dt = dt*2
       di_time = di_time*2
!----  change
!      if(i_time.lt.10000) write(61,*) r_time,dt,cor_max,ieff_time
       dt_2 = dt**2
       do i_type = 1,n_type
       do i_trot = 0,n_trot-1
        rf_width(i_trot,i_type) = rf_width(i_trot,i_type) / 2.d0**0.5d0
       end do
       end do
       do i_order = 1,n_order
        r_dummy = 2**i_order
!       if(i_order.gt.2) r_dummy = 0.
        do i_part = 1,n_part
        do i_trot = 0,n_trot-1
        do i_dim = 1,n_dim
         ux(i_order,i_dim,2*i_trot+1,i_part) =
```

```
     &   ux(i_order,i_dim,2*i_trot+1,i_part) * r_dummy
         ux(i_order,i_dim,2*i_trot+2,i_part) =
     &   ux(i_order,i_dim,2*i_trot+2,i_part) * r_dummy
        end do
        end do
        end do
        do i_dim = 1,n_dim
         scal_x(i_order,i_dim) = scal_x(i_order,i_dim) * r_dummy
        end do
       end do
      end if
!     if(i_time.eq.1000) close(61)

 100  continue

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine symmetrize
      implicit none
      include "pimd.com.f"

      do i_trot = 1,n_trot/2-1
      do i_dim = 1,n_dim
!----  first positions
       do i_part = 1,n_part
        u0(i_dim,2*(n_trot-i_trot)+1,i_part) =
     &  u0(i_dim,2*i_trot+1,i_part)
        u0(i_dim,2*(n_trot-i_trot)+2,i_part) =
     &  -u0(i_dim,2*i_trot+2,i_part)
       end do
!----  then derivatives
       do i_order = 1,n_order
       do i_part = 1,n_part
        ux(i_order,i_dim,2*(n_trot-i_trot)+1,i_part) =
     &  ux(i_order,i_dim,2*i_trot+1,i_part)
        ux(i_order,i_dim,2*(n_trot-i_trot)+2,i_part) =
     &  -ux(i_order,i_dim,2*i_trot+2,i_part)
       end do
       end do
      end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !


      SUBROUTINE INR250(MZ,ISEED,KPTR)
C
C INITIALIZATION ROUTINE FOR R250 RANDOM NUMBER GENERATOR
C FOR 32 BIT COMPUTERS ONLY !
C USES SUN FOR STARTUP
C
      INTEGER MZ(250)
      INTEGER ISEED, KPTR
C
      INTEGER IUF(31),IGF(31)
      INTEGER IADDR1(1000),IADDR2(1000)
      DOUBLE PRECISION RRANF(2000),r
      common /cinr250/ r
C
C
      KPTR = 1
C
      DO 100 I = 1,30
        IGF(I) = 2 ** (I - 1)
        IUF(I) = 2 ** I - 1
 100  CONTINUE
C
      IGF(31) = 2**30
      IUF(31) = 2**30 + (2**30 - 1)
C
      CALL SUN(ISEED,RRANF,250)
C
C MAP REALS TO POSITIVE INTEGERS
C
      DO 200 I = 1,250
        MZ(I) = IDNINT(DBLE(RRANF(I))*DBLE(FLOAT(IUF(31))))
 200  CONTINUE
C
C SPECIAL TREATMENT OF FIRST 31 NUMBERS
C
      DO 300 I = 1,31
        MZ(I) = IOR(MZ(I),IGF(I))
 300  CONTINUE
C
      DO 400 I = 1,31
        MZ(I) = IAND(MZ(I),IUF(I))
 400  CONTINUE
C
C MIXING
C
      CALL SUN(ISEED,RRANF,2000)
C
      DO 500 I = 1,1000
        IADDR1(I) = 1. + 250. * RRANF(I)
        IADDR2(I) = 1. + 250. * RRANF(I + 1000)
 500  CONTINUE
C
      DO 600 I = 1,1000
        NTEMP         = MZ(IADDR1(I))
        MZ(IADDR1(I)) = MZ(IADDR2(I))
        MZ(IADDR2(I)) = NTEMP
 600  CONTINUE
C
C WARMING UP
C
      DO 800 I = 1,8
C
        DO 710  K = 1,147
          MZ(K) = IEOR(MZ(K),MZ(K + 103))
 710    CONTINUE
C
        DO 720 K = 148,250
          MZ(K) = IEOR(MZ(K),MZ(K - 147))
 720    CONTINUE
C
 800  CONTINUE
C
      END
C
C C C C C
```

```
C
      SUBROUTINE SUN(ISEED,RANF,N)
C
C STORES N REAL RANDOM NUMBERS IN RANF
C ISEED IS A STARTUP VALUE
C
      IMPLICIT DOUBLE PRECISION (A-H, O-Z)
C
      INTEGER ISEED
      DOUBLE PRECISION RANF(N),r
      common /cinr250/ r

C
C
C
      DATA FACTOR /41475557.0D0/, TWO28 /268435456.0D0/
C
C
C
      IF (ISEED .GE. 0) THEN
        R=DBLE(ISEED)/TWO28
        R=DMOD(R*FACTOR,1.0D0)
        ISEED=-1
      END IF
C
       DO 100 I = 1,N
        R=DMOD(R*FACTOR,1.0D0)
        RANF(I) = SNGL(R)
 100  CONTINUE
C
      END
C
C
C C C C C
C
      SUBROUTINE R250(MZ,RAN,n_dim,N,KPTR)
C
C STORES N REAL RANDOM NUMBERS IN RAN USING INTEGERS IN MZ
C RANDOM NUMBERS HAVE UNIFORM DISTRIBUTION IN INTERVAL (0,1)
C
      PARAMETER(RINV=1./2147483647.)
      INTEGER MZ(250)
      INTEGER N,KPTR
      DOUBLE PRECISION RAN(n_dim)
      INTEGER I,L
C
C
      I = 0
      L = N + KPTR - 1
C
 10   CONTINUE
C
      KMIN = KPTR
      KMAX = MIN(L,147)
      DO 100 K = KMIN,KMAX
        MZ(K)  = IEOR(MZ(K),MZ(K + 103))
        I      = I + 1
        RAN(I) = MZ(K) * RINV
 100  CONTINUE
C
      KMIN = MAX(KMIN,KMAX+1)
      KMAX = MIN(L,250)

*VOCL LOOP,NOVREC
      DO 200 K = KMIN,KMAX
        MZ(K)  = IEOR(MZ(K),MZ(K - 147))
        I      = I + 1
        RAN(I) = MZ(K) * RINV
 200  CONTINUE
C
      IF  (KMAX .EQ. 250) THEN
        KPTR = 1
        L    = L - 250
        GOTO 10
      END IF
C
      KPTR = KMAX + 1
C
      END
C
! ! ! ! ! ! ! ! !

      subroutine jacobi(dim, matrix, rot_mat, eigen, iter_max)

!     diagonalizes symmetric matrix matrix with given dimension
!     using iteration scheme of jacobi
!
!     rot_mat is matrix cointaining eigenvectors of matrix
!     matrix thus transforms from "orthogonal" space to given space
!
!     eigen is matrix in new coordinate system

      integer dim, dim_max, i_dim, j_dim, k_dim, l_dim
      parameter (dim_max=3)
      double precision matrix(dim_max,dim_max)
      double precision rot_mat(dim_max,dim_max), eigen(dim_max,dim_max)
      double precision h_mat(dim_max,dim_max), loc_ro(dim_max,dim_max)
      integer iterat, iter_max
      integer i_d_max, i_o_max, j_o_max
      double precision max_diag, max_off, tolerance
      parameter (tolerance=1.d-13)
      double precision phi

      if (dim.ne.dim_max) stoP "Allocation problem in jacobi"

      do i_dim = 1, dim_max
       do j_dim = 1, dim_max
        h_mat(i_dim,j_dim) = 0.
        rot_mat(i_dim,j_dim) = 0.
        eigen(i_dim,j_dim) = matrix(i_dim,j_dim)
       end do
       rot_mat(i_dim,i_dim) = 1.
      end do

      if (iter_max.eq.0) iter_max = 4*dim_max*dim_max
      do  100 iterat = 1, iter_max
c----  get biggest diagonal and off-diagonal element
       max_diag = abs(eigen(1,1))
       i_d_max = 1
       max_off = abs(eigen(1,2))
       i_o_max = 1
       j_o_max = 2
       do i_dim = 1, dim_max-1
        if (abs(eigen(i_dim,i_dim)).gt.max_diag) then
         max_diag = abs(eigen(i_dim,i_dim))
```

```
         i_d_max = i_dim
        end if
        do j_dim = i_dim+1, dim_max
         if (abs(eigen(i_dim,j_dim)).gt.max_off) then
          max_off = abs(eigen(i_dim,j_dim))
          i_o_max = i_dim
          j_o_max = j_dim
         end if
        end do
       end do
       if (abs(eigen(dim_max,dim_max)).gt.max_diag) then
        max_diag = eigen(dim_max,dim_max)
        i_d_max = dim_max
       end if
       if (abs(max_off/max_diag).lt.tolerance) return
!      define local rotation matrix,
!      set h_mat = 0-matrix, loc_ro = 1-matrix
       do j_dim = 1, dim_max
        do i_dim = 1, dim_max
         loc_ro(i_dim,j_dim) = 0.d0
         h_mat(i_dim,j_dim) = 0.d0
        end do
        loc_ro(j_dim,j_dim) = 1.d0
       end do
       phi = 0.5 * atan( 2*eigen(i_o_max,j_o_max) /
     & (eigen(i_o_max,i_o_max)-eigen(j_o_max,j_o_max)) )
       loc_ro(i_o_max,i_o_max) = dcos(phi)
       loc_ro(j_o_max,j_o_max) = loc_ro(i_o_max,i_o_max)
       loc_ro(i_o_max,j_o_max) = -sin(phi)
       loc_ro(j_o_max,i_o_max) = -loc_ro(i_o_max,j_o_max)
!      rotate matrix eigen
       do j_dim = 1, dim_max
       do l_dim = 1, dim_max
       if (loc_ro(l_dim,j_dim).ne.0.d0) then
       do k_dim = 1, dim_max
       if (eigen(l_dim,k_dim).ne.0.d0) then
       do i_dim = 1, dim_max
        h_mat(i_dim,j_dim) = h_mat(i_dim,j_dim)
     &  + loc_ro(k_dim,i_dim)*eigen(k_dim,l_dim)*loc_ro(l_dim,j_dim)
       end do
       end if
       end do
       end if
       end do
       end do
       do j_dim = 1, dim_max
       do i_dim = 1, dim_max
        eigen(i_dim,j_dim) = h_mat(i_dim,j_dim)
        h_mat(i_dim,j_dim) = 0.d0
       end do
       end do

       do j_dim = 1, dim_max
       do k_dim = 1, dim_max
       if (loc_ro(k_dim,j_dim).ne.0.d0) then
       do i_dim = 1, dim_max
        h_mat(i_dim,j_dim) = h_mat(i_dim,j_dim)
     &  +  rot_mat(i_dim,k_dim) * loc_ro(k_dim,j_dim)
       end do
       end if
       end do
       end do
```

```
       do i_dim = 1, dim_max
       do j_dim = 1, dim_max
        rot_mat(i_dim,j_dim) = h_mat(i_dim,j_dim)
       end do
       end do

  100 continue

       end subroutine

! ! ! ! ! ! ! ! ! ! !
       subroutine addnumtostring(string,number)
       implicit none
       integer*4 i,strlen,number,nodig,num,snum
       character(*) string

       snum=number
       do i=len(string),1,-1
         if(string(i:i)/=' ') goto 10
       enddo
    10 strlen=i

       nodig=int(log10(1.0*snum+0.1))+1
       do i=nodig,1,-1
         num=snum/10**(i-1)
         string(strlen+1:strlen+1)=char(48+num)
         strlen=strlen+1
         snum=snum-num*10**(i-1)
       enddo

       return
       end

!!!!!!!!!!!!
```

pimd.com.f

```
!---- where to pay attention when new potential gets build in
!----
!---- look for places where f_potential.eq.2 occurs
!
!---- initialize default: mass,interaction parameters,cut_off,skin
!               temperature, time step, pressure in para_init
!           ---> chose reasonable dynamic mode masses for each particle type
!           ---> calculate shifts in energy
!---- initialize default structure and default chemical composition
!---- make output file and input file compatible
!---- write new force/energy routine

       real pi,twopi,sqrt2,sqrt3
       parameter (pi=3.14159265,twopi=2*pi,sqrt2=1.41421356,
      &           sqrt3=1.73205081)

       double precision mass_amu,joule,ev,elec_prefac
       parameter (mass_amu=0.02061,joule=7.2432e22,eV=1.1605e4,
      &           elec_prefac=1.6709e5)
```

```
      integer i_dummy
      double precision r_dummy
      complex c_dummy
      character(64) char_dummy

      integer f_symmetry

      integer n_bin_x,n_bin_y,n_bin_z
      integer i_silic,n_silic,i_oxy,n_oxy

!---- f_symmetry = 1:  fcc symmetry
!     parameter (f_symmetry=1)
!     parameter (n_bin_x=5,n_bin_y=5,n_bin_z=5)
!     parameter (n_silic=n_bin_x*n_bin_y*n_bin_z,n_oxy=3*n_silic)
!---- tridymit
!     parameter (f_symmetry=2)
!     parameter (n_bin_x=4,n_bin_y=3,n_bin_z=3)
!     parameter (n_silic=8*n_bin_x*n_bin_y*n_bin_z,n_oxy=2*n_silic)
!---- cristobalit along c-axis: sometimes converts to kreatine
!      parameter (f_symmetry=3)
!      parameter (n_bin_x=4,n_bin_y=3,n_bin_z=2)
!      parameter (n_silic=12*n_bin_x*n_bin_y*n_bin_z,n_oxy=2*n_silic)
!---- cristobalit in cubic form
!     parameter (f_symmetry=4)
!     parameter (n_bin_x=3,n_bin_y=n_bin_x,n_bin_z=n_bin_x)
!     parameter (n_silic=8*n_bin_x*n_bin_y*n_bin_z,n_oxy=2*n_silic)
!---- beta quarz
!      parameter (f_symmetry=6)
!      parameter (n_bin_x=5,n_bin_y=3,n_bin_z=4)
!      parameter (n_bin_x=6,n_bin_y=4,n_bin_z=5)
!      parameter (n_silic=6*n_bin_x*n_bin_y*n_bin_z,n_oxy=2*n_silic)
!---- alpha quarz
      parameter (f_symmetry=7)
      parameter (n_bin_x=4,n_bin_y=3,n_bin_z=4)
!      parameter (n_bin_x=5,n_bin_y=3,n_bin_z=4)
!     parameter (n_bin_x=6,n_bin_y=4,n_bin_z=5)
      parameter (n_silic=6*n_bin_x*n_bin_y*n_bin_z,n_oxy=2*n_silic)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

      integer i_part,j_part,n_part
!      parameter (n_silic=1,n_oxy=2)

      parameter (n_part=n_silic+n_oxy)

      integer i_neigh,n_neigh
      parameter (n_neigh=1000)
      integer neighbor(0:n_neigh,n_part)
      common /cneighbor/ neighbor
      integer n_bin_max,n_neigh_max
      common /cn_binneigh/ n_bin_max,n_neigh_max

      integer i_trot,j_trot,n_trot,n_time_scal
      parameter (n_trot=1,n_time_scal=1)
      double precision sqrtp
      common /csqrtp/ sqrtp

      integer i_dim,j_dim,k_dim,l_dim,n_dim,e_dim
      parameter (n_dim=3,e_dim=3)

      double precision delta_r(n_dim),delta_cell(n_dim)
      common /cdelta_rc/ delta_r,delta_cell
```

```
 double precision r0(n_dim,0:n_trot-1,n_part),
&r0_old(n_dim,0:n_trot-1,n_part),r_real(n_dim,0:n_trot-1,n_part),
&force(n_dim,0:n_trot-1,n_part)
 double precision u0(n_dim,2*n_trot,n_part),force_loc(n_dim)
 common /cr0u0force/ r0,r0_old,r_real,u0,force,force_loc
 integer pbc0(n_dim,0:n_trot-1,n_part),f_skin
 common /cpbc0/ pbc0,f_skin
 double precision v_real(n_dim,2*n_trot,n_part)
 common /cvreal/  v_real
 double precision r_real_avg(n_dim,0:n_trot-1,n_part)
 common /crealavg/ r_real_avg


 integer i_order,j_order,n_order
 parameter (n_order=2)
 double precision ux(n_order,n_dim,2*n_trot,n_part),
&fr_box(n_dim+3),
&f0(n_dim,2*n_trot,n_part),fr(n_dim,2*n_trot,n_part)
 common /cuf0/ ux,f0,fr,fr_box

 integer i_time,n_relax,n_obser,n_safe
 common /citime/ i_time,n_relax,n_obser,n_safe
 double precision dt,dt_2,r_time
 common /cdtime/ dt,dt_2,r_time

 double precision tk,tkp,beta,betap
 common /ctherm/ tk,tkp,beta,betap

 integer i_type,j_type,n_type,type(n_part)
 parameter (n_type=2)
 double precision rf_width(0:n_trot-1,n_type),rfb_width
 common /ctype/ type,rf_width,rfb_width

 double precision mass(0:n_trot-1,n_type),friction(n_type),
&epsil(n_type,n_type),sigma(n_type,n_type),k_quant(n_type),
&k_eigen(0:n_trot-1,n_type),sigma_2(n_type,n_type),
&e_shift(n_type,n_type),v_therm(0:n_trot-1,n_type),
&c_vandwaals(n_type,n_type),w_therm,
&chargenp(n_part),chargenm(n_part)
 double precision charge(n_part), chargex(n_order,n_part),
&fQ(n_part)
 common /cproperties/ mass,friction,epsil,sigma,k_quant,
&k_eigen,sigma_2,e_shift,v_therm,c_vandwaals,w_therm,
&charge,chargex,
&chargenp,chargenm,
&fQ
 double precision vash_h(n_type,n_type),vash_eta(n_type,n_type),
&  vash_alpha(n_type),vash_B(n_type),vash_l(n_type),
&  vash_ctheta(n_type),vash_r0(n_type)
 common /vashishta/ vash_h,vash_eta,vash_alpha,vash_B,
&    vash_l, vash_ctheta, vash_r0
 integer n_angular, n_ang(n_type)
 parameter (n_angular=6)
 double precision triple_cos(n_part,n_angular),
&                 triple_rij(n_part,n_angular),
&                 triple_rik(n_part,n_angular),
&                 triple_deltaij(n_part,n_angular,n_dim),
&                 triple_deltaik(n_part,n_angular,n_dim)
 integer          triple_partj(n_part,n_angular),
&                 triple_partk(n_part,n_angular)
 common /threebody/ n_ang, triple_cos, triple_rij, triple_rik,
```

```
&     triple_deltaij, triple_deltaik, triple_partj, triple_partk
  double precision r_cutoff(n_type,n_type),r_skin(n_type,n_type),
&      r_cutoff_2(n_type,n_type),r_range_2(n_type,n_type),
&      r_range(n_type,n_type),r_cutoff_max
  common /ccutoffskin/ r_cutoff,r_skin,r_cutoff_2,r_range_2,
&     r_range,r_cutoff_max
  double precision trans_units,d0(n_type,n_type),
&     mszeta(n_type,n_type),
&     msr0(n_type,n_type), gamma(n_type,n_type)
   integer msn(n_type)
  common /morsestretch/ d0,msr0,gamma,msn,mszeta
  parameter (trans_units = 503.22862)
  double precision dipole(n_dim,n_part), avg_dipole(0:n_dim,n_type),
&     avg_dipole_part(0:n_dim,n_part),pol_total(n_dim),
&     pol_dipoles(n_dim)
  common /cpolarize/ dipole,avg_dipole,avg_dipole_part,
&     pol_total, pol_dipoles

  integer iseed,mz(250),kptr
  double precision random(n_dim)
  common /crandom/ mz,iseed,kptr

  double precision predict_coef(0:n_order-1,0:n_order),
&correct_coef(0:n_order)
  common /cpredict/ predict_coef,correct_coef

  double precision t_dynamic,t_dynamic_1,t_dynamic_2,t_diag,t_ndiag
&      t_diag_1,t_ndiag_1,t_dynamic_charge
  common /ctdymic/ t_dynamic,t_dynamic_1,t_dynamic_2,t_diag,t_ndiag
&      ,t_diag_1,t_ndiag_1,t_dynamic_charge

  double precision v_inter,v_inter_1,v_inter_2, v_gauge
  common /cinter/ v_inter,v_inter_1,v_inter_2, v_gauge
  double precision v_chain,prim_est,prim_est_1,prim_est_2
  common /cprim_est/ v_chain,prim_est,prim_est_1,prim_est_2
  double precision viri_est,viri_est_1,viri_est_2
  common /cviri_est/ viri_est,viri_est_1,viri_est_2
  double precision v_qeqself, v_realcoulomb, v_shortrange, v_charge
  common /cvqeq/ v_qeqself, v_realcoulomb, v_shortrange, v_charge
  double precision v_threebody
  common /vthree/ v_threebody
  double precision dipole_class(0:n_dim,12),
&     dipole_class1(0:n_dim,12),
&     dipole_class2(0:n_dim,12),dipole_class4(0:n_dim,12),
&     dipole_count
  common /cdipoleclass/ dipole_class, dipole_class1,
&     dipole_class2, dipole_class4, dipole_count

  integer i_cor,n_cor,i_measure
  parameter (n_cor=2048)
  double precision conf_centr(n_dim,n_part)
  double precision cor_obs(0:n_cor),cor_func(0:n_cor,n_type)
  double precision qdot_cor(0:n_cor), qdot0(n_dim,n_part)
  double precision cor_func_A(0:n_cor)
  common /ccor/ conf_centr,cor_obs,cor_func,i_measure
  common /ccor2/ cor_func_A,qdot_cor,qdot0
  double precision cor_func_B(0:n_cor),
&     stored_vel(n_dim,0:n_cor,n_part)
  common /ccorfuncb/ cor_func_B, stored_vel

  integer di_time,ni_time,ieff_time
  common /creal_time/ di_time,ieff_time
```

```
  parameter (ni_time=n_time_scal)

  double precision cor_ifunc(0:n_trot-1)
  common /cicor/ cor_ifunc

  double precision time_obs
  common /ctime_obs/ time_obs

  integer count_ifunc
  common /ccount_ifunc/ count_ifunc

  integer n_pbc
  common /cn_pbc/ n_pbc

  integer f_movie
  f_movie = 1; write multiple frames xyz-file
  f_movie = 0; do not
  parameter(f_movie=0)

  integer f_confmovie
  f_confmovie = 1; write multiple frames conf-file
  f_confmovie = 0; do not
  parameter(f_confmovie=0)

  integer f_e_applied
  f_e_applied = 0; no external electric field
  f_e_applied = 1; electric field in x direction
  f_e_applied = 2; electric field in y direction
  f_e_applied = 3; electric field in z direction
  parameter(f_e_applied=1)
  CAUTION:   for Scandolo potential, please set
  =======       telectric in metric.f / module
                electric_field accordingly !!!

  integer f_potential
  f_potential = 1:  Harmonic oscillator
  f_potential = 2;  Lennard Jones
  f_potential = 3;  BKS potential
  f_potential = 4;  Aziz
  f_potential = 5;  MS QEq potential
  f_potential = 6;  MS fluc-Q potential
  f_potential = 7;  MS DQEq potential
  f_potential = 8;  Vashishta potential
  f_potential = 9;  Scandolo potential
  parameter(f_potential=9)

  integer f_chargeadjust
  f_chargeadjust = 1:  constant charges
  f_chargeadjust = 2;  self consistency
  f_chargeadjust = 3;  extended lagrangian
  f_chargeadjust = 4;  matrix inversion method
  parameter(f_chargeadjust=1)

  double precision qeqelectroneg(n_type), qeqhardness(n_type),
&  qeqlambdai(n_part), qeqlambda, qeqrec(n_part),
&  qeqmass(n_type), qeqoldcharge(n_part,0:3), qeqfriction
  integer n_atomtype(n_type)
  common /qeq/ qeqelectroneg, qeqhardness,
&  qeqlambdai, qeqlambda, qeqrec, qeqoldcharge,
&  qeqmass, qeqfriction, n_atomtype
  double precision constant_charge(n_type)
  common /flucqeq/ constant_charge
```

```
 double precision force_scal(n_dim+3),force_scal_inter(n_dim+3)
&,scal_0(n_dim+3),scal_x(n_order,n_dim+3)
 double precision pressure(n_dim+3),mass_box,press,fric_box
 integer f_pressure
 common /cpress/ force_scal_inter,scal_0,scal_x,pressure,
&       force_scal,press,f_pressure
 common /cpress2/ mass_box,fric_box

 double precision h_mat(3,3),h_mat_2(3,3),h_mat_dot(3,3),
&h_mat_2d(3,3),h_mat_2_inv(3,3),g_mat_dot(3,3),h_mat_inv(3,3)
 common /ctensor/ h_mat,h_mat_2,h_mat_dot,h_mat_2d,h_mat_2_inv,
&       g_mat_dot,h_mat_inv
 double precision met_ten_voi(n_dim+3),g_voi_dot(n_dim+3)
 common /cvoigt/ met_ten_voi,g_voi_dot

 double precision scal_1(n_dim+3),volume,vol_1,vol_2,
&       scal_2(n_dim+3,n_dim+3),v_press
 common /cpress_obs/ scal_1,scal_2,v_press,volume,vol_1,vol_2

 double precision r_ref(n_dim,n_part)
 common /cr_ref/ r_ref

 integer i_grid,n_grid
 parameter (n_grid=256)
 double precision g_r(n_type,n_type,0:n_grid+1)
 common /cg_r/ g_r

 integer i_potential,n_potential
 parameter (n_potential=10000)
 double precision r_pot(n_type,n_type,0:n_potential),
&     r_pot_d(n_type,n_type,0:n_potential)
 double precision r_pot_1(n_type,n_type,0:n_potential),
&     r_pot_2(n_type,n_type,0:n_potential),
&     r_pot_2q(n_type,n_type,0:n_potential),
&     r_pot_d_1(n_type,n_type,0:n_potential),
&     r_pot_d_2(n_type,n_type,0:n_potential),
&     r_pot_d_2q(n_type,n_type,0:n_potential)
 common /cr_potd/ r_pot_1,r_pot_2,r_pot_2q,r_pot_d_1,
&     r_pot_d_2,r_pot_d_2q ,r_pot,r_pot_d
 complex tab_exp_ima(-n_potential:n_potential)
 common /ctab_exp_im/ tab_exp_ima

 integer i_run,n_run
 common /in_run/ i_run,n_run

 double precision erfcc
 double precision alpha,q_cut_2,self_energy,v_ewald,
&dq(n_dim,n_dim),v_cut,alpha_prefac
 common /cewald/ alpha,q_cut_2,self_energy,v_ewald,dq,v_cut
&,alpha_prefac

 integer nq_max(n_dim),n_rec_vec,n_rec_vec_max,i_rec_vec
 common /crec_ewald/ nq_max,n_rec_vec
 parameter (n_rec_vec_max=12*n_part)
 integer rec_vec_list(n_rec_vec_max,n_dim)
 common /crec_vec_list/ rec_vec_list

 double precision rec_force_offset(n_dim+3),rec_pot_offset
 common /rec_offset/ rec_force_offset,rec_pot_offset

 double precision r_cut_short,e_shift_short(n_type,n_type)
 parameter (r_cut_short=9.0)

 double precision ela_born(n_dim,n_dim,n_dim,n_dim),
&     ela_born_1(n_dim,n_dim,n_dim,n_dim),time_born
 common /celastic/ ela_born,ela_born_1,time_born
 double precision strain(n_dim,n_dim), strain_1(n_dim,n_dim),
&     strain_2(n_dim,n_dim,n_dim,n_dim)
 common /cstrain/ strain,strain_1,strain_2
 integer f_opt_rec
 common /cf_opt/ f_opt_rec

 double precision rot_mat(n_dim,n_dim), eigen(n_dim,n_dim)
 common /cjacobi/ rot_mat, eigen

 double precision ela_kin(n_dim,n_dim)
 common /cela_kin/ ela_kin

 double precision for_sca_mfa_cor
 common /cfor_sca_mfa_cor/ for_sca_mfa_cor

 integer i_neigh_ori(n_part,4)
 double precision hist_ang(3,n_grid)
 common /ci_neigh_ori/ hist_ang, i_neigh_ori
 double precision r_neigh_min, r_neigh_max, neigh_time
 common /cr_neigh/ r_neigh_min, r_neigh_max, neigh_time

 double precision  ord_par,ord_par_1,ord_par_2,ord_par_4,ord_c
 common /cord_par/ ord_par,ord_par_1,ord_par_2,ord_par_4,ord_c

 double precision  ord_parB,ord_parB_1,ord_parB_2,
&     ord_parB_4,ordB_c
 common /cord_parB/ ord_parB,ord_parB_1,ord_parB_2,
&     ord_parB_4,ordB_c

 double precision  ord_parC,ord_parC_1,ord_parC_2,ord_parC_4,ordC_c
 common /cordparC/ ord_parC,ord_parC_1,ord_parC_2,ord_parC_4,ordC_c
```

B.1.1. Inputfiles

Inputfile pimd.inp for BKS potential

```
           3   f_potential, information available
           1   f_chargeadjust
       10000   # of relaxation steps
      200000   # of observation steps
       10000   # of steps between configuration output
       48674   random # init

0.130910E-03   time step
3.000000e+02   temperature

0.579141E+00  0.329760E+00   masses
0.763884E+02  0.763884E+02   friction
0.464498E+01  0.763884E+02   mass + fr. of box

0.000000e+00   pressure
           1   0 = NVT; 1 = NpT
           0   1 = optimized Ewald



0.000000E+00  0.208934E+09    exp. energy parameters
0.208934E+09  0.161167E+08    exp. energy parameters

0.000000E+00  0.487318E+01   inverse exp. lenght scales
0.487318E+01  0.276000E+01   inverse exp. lenght scales

0.000000E+00  0.154971E+07   van de Waals strength
0.154971E+07  0.203088E+07   van de Waals strength

0.300000E+00  0.200000E+02   alpha, v_cut

0.450000E+00  0.450000E+00   skin radius
0.450000E+00  0.450000E+00   skin radius
```

Inputfile pimd.inp for fluctuating charge potential

```
           6   f_potential, information available
           3   f_chargeadjust
       50000   # of relaxation steps
      200000   # of observation steps
       10000   # of steps between configuration output
       48748   random # init

0.130910E-03   time step
3.000000e+02   temperature

0.579141E+00  0.329760E+00   masses
0.763884E+02  0.763884E+02   friction
0.464498E+01  0.763884E+02   mass + fr. of box

0.000000e+00   pressure
           1   0 = NVT; 1 = NpT
           0   1 = optimized Ewald

0.483676E+05  0.101435E+06   electronegativity
0.809298E+05  0.155083E+06   hardness of EN

0.400000E-01  0.400000E-01   fictitious charge mass
0.763884E+03   charge friction

0.295600E+00  0.459970E+02   MS bond strength D0
0.459970E+02  0.536300E+00   MS bond strength D0

0.341030E+01  0.161480E+01   MS bond length R0
0.161480E+01  0.378350E+01   MS bond length R0

0.117139E+02  0.880220E+01   MS force constant gamma
0.880220E+01  0.104112E+02   MS force constant gamma

0.148810E+01  0.148810E+01   slater orbital exponent zeta
0.186846E+01  0.186846E+01   slater orbital exponent zeta

           3             2   principal quantum number n

0.300000E+00  0.000000E+00   alpha, v_cut

0.450000E+00  0.450000E+00   skin radius
0.450000E+00  0.450000E+00   skin radius
```

Inputfile pimd.inp for fluctuating dipole potential

```
           9   f_potential, information available
           1   f_chargeadjust
        1000   # of relaxation steps
        6000   # of observation steps
        1000   # of steps between configuration output
        6811   random # init

 0.130910E-03   time step
 0.300000E+03   temperature

 0.579141E+00  0.329760E+00   masses
 0.763884E+02  0.763884E+02   friction
 0.464498E+01  0.763884E+02   mass + fr. of box

 0.217290E+04   pressure
           1   0 = NVT; 1 = NpT



 0.300000E+00  0.200000E+02   alpha, v_cut

 0.450000E+00  0.450000E+00   skin radius
 0.450000E+00  0.450000E+00   skin radius
```

B.2. Analysis Programs

examf60.f

Reads the output file `fort.60` and writes analysis output to standard out.
The output contains averaged box dimensions, lattice constants, stress, volume per unit cell, elastic constants, polarization, dielectric constant and piezoelectric stress and strain coefficients.
The gaussj routine from “Numerical Recipes in Fortran” [74] needs to be included in the code.

calcstrain.f

Reads the output file `fort.60` and writes analysis output to `fort.59`. Needs a reference box matrix from standard in.
The output is the strain with time, relative to a reference box h_0. The reference box is prompted at the start of the program.

examf60.f

```
      program exam_f60
      implicit none

      integer n_bin_x, n_bin_y, n_bin_z
      parameter (n_bin_x=4, n_bin_y=3, n_bin_z=4)

      integer i_relax, n_relax, n_obser
      integer i_dim, j_dim, n_dim, e_dim, iv_dim, voi_ind
      parameter (n_dim=3, e_dim=3)
      double precision vol_1
      common /cvol_1/ vol_1

      double precision x(8), scal(n_dim + e_dim), scal_1(n_dim + e_dim),
     & scal_2(n_dim + e_dim, n_dim + e_dim), temp, y(10),
     & pol_total(n_dim), pol_dipoles(n_dim), pol_fluc(n_dim,n_dim),
     & pol_avg(n_dim), polstrain(n_dim+e_dim,n_dim)
      double precision a,b,c,da,db,dc, redpol_avg(n_dim), redpol(n_dim)
      double precision h_mat(3,3), h_i(3,3), h1(3,3), h1_i(3,3)
     &     , h1_i_avg_i(3,3), h1_i_avg(3,3)
      double precision h(3,3), h0(3,3), ht(3,3), h0i(3,3), h0ti(3,3),
     &     h0t(3,3)
      double precision s1(3,3), s2(3,3), strainm(3,3),
     &     strain(6), strain_1(6), stress(6), polstress(6,3),stress_1(6)


      write(*,*) "How many steps to discard?"
      read(*,*)  n_relax
      write(*,*) "Temp."
      read(*,*) temp

! first run

      do i_dim=1, n_dim
         redpol_avg(i_dim)=0.d0
         do j_dim = 1, n_dim
            h1_i_avg(i_dim,j_dim) = 0.d0
         end do
      end do

      do i_relax = 1, n_relax
       read(60,'(36e15.7)') (x(i_dim), i_dim=1, 8),
     &  (scal(j_dim), j_dim=1, n_dim+e_dim), (y(i_dim), i_dim=1,10),
     &  (pol_total(i_dim), i_dim=1,3), (pol_dipoles(i_dim), i_dim=1,3),
     &        (stress(i_dim), i_dim=1,6)
      end do

      n_obser = 0
      do i_dim = 1, n_dim+e_dim
       scal_1(i_dim) = 0.
      end do

 901  continue
       read(60,'(36e15.7)',end=902) (x(i_dim), i_dim=1, 8),
     &  (scal(j_dim), j_dim=1, n_dim+e_dim), (y(i_dim), i_dim=1,10),
     &  (pol_total(i_dim), i_dim=1,3), (pol_dipoles(i_dim), i_dim=1,3),
     &        (stress(i_dim), i_dim=1,6)
       n_obser = n_obser + 1
       call trans_to_tens(scal, h_mat)
       call inv_mat(h_mat, h_i)
       do i_dim = 1, n_dim + e_dim
        scal_1(i_dim) = scal_1(i_dim) + scal(i_dim)
       end do
      call trans_to_tens(scal, h1)
      call inv_mat(h1, h1_i)

       do i_dim = 1, n_dim
          pol_total(i_dim) = pol_total(i_dim) - pol_dipoles(i_dim)
! uncomment following line to consider only polarization due to dipoles:
!          pol_total(i_dim) = pol_dipoles(i_dim)
       end do

       do i_dim = 1, n_dim
          do j_dim = 1, n_dim
             h1_i_avg(i_dim,j_dim) =
     &            h1_i_avg(i_dim,j_dim) + h1_i(i_dim,j_dim)
             redpol_avg(i_dim)= redpol_avg(i_dim) + h_i(i_dim,j_dim)
     &            * pol_total(j_dim)
          end do
       end do

      goto 901
 902  continue

      do i_dim = 1, n_dim
         redpol_avg(i_dim)= redpol_avg(i_dim) / n_obser
         do j_dim = 1, n_dim
            h1_i_avg(i_dim,j_dim) = h1_i_avg(i_dim,j_dim) / n_obser
         end do
      end do

      do i_dim =1 , n_dim + e_dim
         scal_1(i_dim) = scal_1(i_dim) / n_obser
      end do

      call inv_mat(h1_i_avg, h1_i_avg_i)

! calculate h0, h0^-1, h0', h0'^-1 for strain
      call trans_to_tens(scal_1,h0)
      write(*,*) "h0", h0(1,1), h0(2,2), h0(3,3),h0(1,2),h0(2,3),h0(1,3)

      call inv_mat(h0,h0i)
      call trans_mat(h0,h0t)
      call inv_mat(h0t,h0ti)

      write(*,*) "h0ti", h0ti(1,1), h0ti(2,2), h0ti(3,3)

      close(60)

! second run

      do i_relax = 1, n_relax
       read(60,'(36e15.7)') (x(i_dim), i_dim=1, 8),
     &  (scal(j_dim), j_dim=1, n_dim+e_dim), (y(i_dim), i_dim=1,10),
     &  (pol_total(i_dim), i_dim=1,3), (pol_dipoles(i_dim), i_dim=1,3),
     &  (stress(i_dim), i_dim=1,6)
      end do

      n_obser = 0
      do i_dim = 1, n_dim+e_dim
       scal_1(i_dim) = 0.
       strain_1(i_dim) = 0.
       stress_1(i_dim) = 0.
```

```
      do j_dim = 1, n_dim+e_dim
        scal_2(i_dim,j_dim) = 0.
       end do
      end do
      do j_dim=1, n_dim
         pol_avg(j_dim)=0.
         do i_dim=1,6
            polstrain(i_dim,j_dim)=0.
            polstress(i_dim,j_dim)=0.
         end do
         do i_dim=1,3
            pol_fluc(i_dim,j_dim)=0.
         end do
      end do

  100 continue
       read(60,'(36e15.7)',end=200) (x(i_dim), i_dim=1, 8),
     &  (scal(j_dim), j_dim=1, n_dim+e_dim), (y(i_dim), i_dim=1,10),
     &  (pol_total(i_dim), i_dim=1,3), (pol_dipoles(i_dim), i_dim=1,3),
     &  (stress(i_dim), i_dim=1, 6)
       n_obser = n_obser + 1
       do i_dim = 1, n_dim + e_dim
        scal_1(i_dim) = scal_1(i_dim) + scal(i_dim)
        do j_dim = 1, n_dim + e_dim
         scal_2(i_dim,j_dim) = scal_2(i_dim,j_dim) +
     &          scal(i_dim) * scal(j_dim)
        end do
       end do

       call trans_to_tens(scal, h_mat)
       call inv_mat(h_mat, h_i)
       do i_dim = 1, n_dim
          pol_total(i_dim) = pol_total(i_dim) - pol_dipoles(i_dim)
! uncomment following line to consider only polarization due to dipoles:
!         pol_total(i_dim) = pol_dipoles(i_dim)
       end do

       do i_dim = 1, n_dim
          redpol(i_dim) = 0.
          do j_dim = 1, n_dim
             iv_dim=voi_ind(i_dim,j_dim)
             redpol(i_dim) = redpol(i_dim) + h_i(i_dim,j_dim) *
     &            pol_total(j_dim)
          end do
       end do


       do i_dim = 1, n_dim
          pol_total(i_dim) = 0.
          do j_dim = 1, n_dim
             pol_total(i_dim) = pol_total(i_dim)
     &            + h1_i_avg_i(i_dim,j_dim) *
     &            (redpol(j_dim) - redpol_avg(j_dim))
          end do
! comment out following line to consider only polarization due to charges:
          pol_total(i_dim) = pol_total(i_dim) + pol_dipoles(i_dim)
       end do

       do i_dim = 1, n_dim
          pol_avg(i_dim) = pol_avg(i_dim) + pol_total(i_dim)
       end do
```

```
       do i_dim= 1, n_dim
          do j_dim=1,  n_dim
             pol_fluc(i_dim,j_dim) = pol_fluc(i_dim,j_dim) +
     &            pol_total(i_dim)*pol_total(j_dim)
          end do
       end do

       call trans_to_tens(scal,h)
       call trans_mat(h,ht)
       call mult_mat(h,h0i,s1)
       call mult_mat(ht,s1,s2)
       call mult_mat(h0ti,s2,strainm)

       do i_dim=1,3
          do j_dim=1,3
             if (i_dim.eq.j_dim)
     &          strainm(i_dim,j_dim) = strainm(i_dim,j_dim) - 1.d0
          strainm(i_dim,j_dim) = 0.5d0 * strainm(i_dim,j_dim)
          end do
       end do

       call trans_to_voigt(strainm,strain)

       do i_dim = 1, 6
          strain_1(i_dim) = strain_1(i_dim) + strain(i_dim)
          stress_1(i_dim) = stress_1(i_dim) + stress(i_dim)
       end do

       write(44,'(7f15.5)') x(1), (strain(i_dim), i_dim=1,6)

       do i_dim = 1, 6
          do j_dim = 1, 3
             polstrain(i_dim,j_dim) = polstrain(i_dim,j_dim) +
     &            strain(i_dim) * pol_total(j_dim)
          polstress(i_dim,j_dim) = polstress(i_dim,j_dim) +
     &            stress(i_dim) * pol_total(j_dim)
          end do
       end do
       goto 100
  200 continue

       write(*,*) "Obs. Steps", n_obser
       do i_dim = 1, n_dim+e_dim
        scal_1(i_dim) = scal_1(i_dim) / n_obser
        strain_1(i_dim) = strain_1(i_dim) / n_obser
        stress_1(i_dim) = stress_1(i_dim) / n_obser
       end do
       write(*,*) "strain_1", (strain_1(i_dim), i_dim=1,n_dim)
       write(*,*) "stress_1", (stress_1(i_dim), i_dim=1,n_dim)

       do i_dim = 1, n_dim
          pol_avg(i_dim) = pol_avg(i_dim) / n_obser
       end do

       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
          pol_fluc(i_dim,j_dim) = pol_fluc(i_dim,j_dim) / n_obser
          pol_fluc(i_dim,j_dim) = (pol_fluc(i_dim,j_dim) -
     &       pol_avg(i_dim)*pol_avg(j_dim))
       end do
       end do
```

```
 do i_dim = 1, 6
    do j_dim = 1 ,3
       polstrain(i_dim,j_dim)=polstrain(i_dim,j_dim) / n_obser
       polstrain(i_dim,j_dim)=(polstrain(i_dim,j_dim) -
&           strain_1(i_dim)*pol_avg(j_dim))
       polstress(i_dim,j_dim)=polstress(i_dim,j_dim) / n_obser
       polstress(i_dim,j_dim)=(polstress(i_dim,j_dim) -
&           stress_1(i_dim)*pol_avg(j_dim))
    end do
 end do

 do i_dim = 1, n_Dim + e_dim
  do j_dim = 1, n_dim + e_dim
   scal_2(i_dim,j_dim) = scal_2(i_dim,j_dim) / n_obser
   scal_2(i_dim,j_dim) = ( scal_2(i_dim,j_dim) -
&  scal_1(i_dim) * scal_1(j_dim) )
  end do
 end do

 call calc_vol(scal_1,vol_1)

 call piezo_coeff(pol_avg,scal_2,scal_1,temp,polstrain,polstress)
 call dielectric_const(pol_fluc,pol_avg,scal_2,scal_1,temp)


 write(*,*) "Avg. Polarization"
 do i_dim = 1, 3
   write(*,*) pol_avg(i_dim)/vol_1,"+-",
&        sqrt(pol_fluc(i_dim,i_dim))/vol_1
 end do

 write(*,*) "Avg. Scal. Variab."
 do i_dim = 1, n_dim + e_dim
   write(*,'(f15.9af15.9)') scal_1(i_dim), " +-",
&        sqrt(scal_2(i_dim,i_dim))
 end do
 a=scal_1(1)/n_bin_x
 b=scal_1(2)/sqrt(3.)/n_bin_y
 c=scal_1(3)/n_bin_z
 da= sqrt(scal_2(1,1))/n_bin_x
 db= sqrt(scal_2(2,2))/sqrt(3.)/n_bin_y
 dc= sqrt(scal_2(3,3))/n_bin_z
 write(*,*) "a ",a," +- ",da
 write(*,*) "b ",b," +- ",db
 write(*,*) "c ",c," +- ",dc
 write(*,*) "c/a ", 1. / (sqrt(a*b ) / c)," +- ",sqrt(
&   (.25*((da/a)**2+(db/b)**2) +dc**2)/(a*b))
 write(*,*) "vol ", sqrt(3.)/2*a*b*c," +- ",sqrt(.75*
&     ((da*b*c)**2 + (a*db*c)**2 + (a*b*dc)**2))
 write(*,'(af15.9af15.9af15.9af15.9af15.9af15.9a)')
&     "h = {",scal_1(1),",",scal_1(2),",",
&     scal_1(3),",",scal_1(4),",",scal_1(5),",",scal_1(6),"}"
 write(*,'(af15.9af15.9af15.9af15.9af15.9af15.9a)')
&     "dh = {",sqrt(scal_2(1,1)),",",sqrt(scal_2(2,2)),","
&     ,sqrt(scal_2(3,3)),",",sqrt(scal_2(4,4)),",",
&     sqrt(scal_2(5,5)),",",sqrt(scal_2(6,6)),"}"
 write(*,'(af15.9af15.9af15.9af15.9af15.9af15.9a)')
&     "P = {",pol_avg(1)/vol_1,",",pol_avg(2)/vol_1,",",
&     pol_avg(3)/vol_1,"}"
 write(*,'(af15.9af15.9af15.9af15.9af15.9af15.9a)')
&     "dP = {",sqrt(pol_fluc(1,1))/vol_1,",",
&     sqrt(pol_fluc(2,2)/vol_1),",",sqrt(pol_fluc(3,3)/vol_1),"}"
```

```
      write(*,*) "Avg. Fluc. Tens."
      do i_dim = 1, n_Dim + e_dim
       do j_dim = 1, n_dim + e_dim
        scal_2(i_dim,j_dim) = scal_2(i_dim,j_dim) / temp
       end do
       write(*,'(6e13.5)')
     & ( scal_2(i_dim,j_dim), j_dim = 1, n_dim + e_dim )
      end do

      call calc_strain(scal_2, scal_1)
      call calc_elco(scal_2, scal_1)

      end program

! ! ! ! ! ! ! ! ! !

      subroutine calc_vol(scal_1,vol_1)
      double precision scal_1, vol_1, h_mat(3,3)

      call trans_to_tens(scal_1, h_mat)
      vol_1 = determ(h_mat)

      end subroutine


! ! ! ! ! ! ! ! ! !

      subroutine dielectric_const(pol_fluc,pol_avg, scal_2,scal_1,temp)
      integer i_dim, j_dim, n_dim, e_dim, iv_dim, voi_ind
      integer k_dim, l_dim, ij_dim, kl_dim
      parameter (n_dim=3, e_dim=3)
      double precision pol_fluc(n_dim,n_dim), dk(n_dim + e_dim)
      double precision vol_1, r_dummy, pol_avg(n_dim), temp
      double precision polcorrect(n_dim,n_dim), smat(6,6),
     &     scal_1(n_dim + e_dim), scal_2(n_dim + e_dim, n_dim + e_dim)
      common /cvol_1/ vol_1

      call calc_smat(scal_2, scal_1, smat)

      do i_dim = 1, n_dim
      do k_dim = 1, n_dim
         polcorrect(i_dim,k_dim) = 0.
      do j_dim = 1, n_dim
      do l_dim = 1, n_dim
         ij_dim = voi_ind(i_dim,j_dim)
         kl_dim = voi_ind(k_dim,l_dim)
         polcorrect(i_dim,k_dim) = polcorrect(i_dim,k_dim) +
     &      smat(ij_dim, kl_dim) * pol_avg(j_dim) * pol_avg(l_dim)
      end do
      end do
      end do
      end do


      do i_dim = 1, n_dim
         write(*,*) (polcorrect(i_dim,k_dim), k_dim=1,3)
      end do


      write(*,*) "dk_ij"
```

```
      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
         iv_dim = voi_ind(i_dim,j_dim)
! (angstrom * e)**2/(epsilon0 angstrom**3 kB K) = 2099841
         dk(iv_dim) = (pol_fluc(i_dim, j_dim))
     &        *2099841/vol_1/temp
         write(*,*) i_dim, j_dim, (pol_fluc(i_dim, j_dim)
     &        -polcorrect(i_dim, j_dim))
     &        *2099841/vol_1/temp
         if (iv_dim.le.3) dk(iv_dim) = dk(iv_dim) +1
      end do
      end do

      write(*,*) "Avg. dielectric constant"
      write(*,'(6f13.5)') (dk(iv_dim), iv_dim = 1, 6)

      end subroutine

! ! ! ! ! ! ! ! !

      subroutine piezo_coeff(temp,polstrain, polstress)
      integer j_dim, k_dim
      parameter (n_dim=3, e_dim=3)
      double precision polstrain(n_dim+e_dim,n_dim),
     &     temp, polstress(6,3)

      write(*,*) "Avg. piezoelectric stress Coefficients [Cb/m^2]"
      do k_dim=1,3
         write(*,'(6f13.5)')
     &        (polstress(j_dim,k_dim)*16.02177 / temp,j_dim=1,6)
      end do


      write(*,*) "Avg. piezoelectric strain Coefficients [10^-12 m/V]"
      do k_dim=1,3
         write(*,'(6f13.5)')
     &        (polstrain(j_dim,k_dim)*1.16045d-6*1.d12/ temp,j_dim=1,6)
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine calc_strain(scal_2, scal_1)
      implicit none
      integer i_dim, j_dim, k_dim, l_dim, n_dim, jv_dim, iv_dim,
     &     a_dim, voi_ind
      parameter (n_dim=3)
      double precision  scal_2(6,6), h_mat(3,3), scal_1(6),
     &  h_i(3,3), strain(6), s_2(3,3,3,3)

      call trans_to_tens(scal_1, h_mat)
      call inv_mat(h_mat, h_i)

      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
       iv_dim = voi_ind(i_dim,j_dim)
       do k_dim = 1, n_dim
       do l_dim = 1, n_dim
        jv_dim = voi_ind(k_dim,l_dim)
        s_2(i_dim, j_dim, k_dim, l_dim) = scal_2(iv_dim,jv_dim)
       end do
       end do
      end do
      end do

      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
       iv_dim = voi_ind(i_dim,j_dim)
       strain(iv_dim) = 0.
       do a_dim = 1, n_dim
        strain(iv_dim) = strain(iv_dim) + (
     &  sqrt(s_2(i_dim,a_dim,i_dim,a_dim))*h_i(a_dim,j_dim) +
     &  sqrt(s_2(a_dim,i_dim,a_dim,i_dim))*h_i(j_dim,a_dim))/2.
       end do
      end do
      end do

      write(*,*) "strain"
      write(*,'(6e19.11)') (strain(iv_dim),iv_dim=1,6)

      end subroutine


! ! ! ! ! ! ! ! ! ! !

      subroutine trans_to_voigt(matrix,vector)
      implicit none
      double precision matrix(3,3), vector(6)
      vector(1) = matrix(1,1)
      vector(2) = matrix(2,2)
      vector(3) = matrix(3,3)
      vector(4) = 2 * matrix(1,2)
      vector(5) = 2 * matrix(2,3)
      vector(6) = 2 * matrix(3,1)
      end subroutine


! ! ! ! ! ! ! ! !

      subroutine trans_to_tens(vector,matrix)
      implicit none
      double precision matrix(3,3), vector(6)
      matrix(1,1) = vector(1)
      matrix(2,2) = vector(2)
      matrix(3,3) = vector(3)
      matrix(1,2) = vector(4)
      matrix(2,1) = vector(4)
      matrix(2,3) = vector(5)
      matrix(3,2) = vector(5)
      matrix(3,1) = vector(6)
      matrix(1,3) = vector(6)
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine inv_mat(mat,mat_inv)
      implicit none
      integer i_dim, j_dim, sign
      double precision mat(3,3), mat_inv(3,3), det_mat
      real determ
      double precision vol_1
      common /cvol_1/ vol_1
```

```
      det_mat = determ(mat)
      vol_1 = det_mat

      do i_dim = 1, 3
      do j_dim = 1, 3
       mat_inv(i_dim,j_dim)  = 0.d0
      end do
      end do
      mat_inv(1,1) = mat(2,2)*mat(3,3) - mat(2,3)**2
      mat_inv(2,2) = mat(3,3)*mat(1,1) - mat(3,1)**2
      mat_inv(3,3) = mat(1,1)*mat(2,2) - mat(1,2)**2
      mat_inv(1,2) = - (mat(2,1)*mat(3,3) - mat(2,3)*mat(3,1))
      mat_inv(2,3) = - (mat(3,2)*mat(1,1) - mat(3,1)*mat(1,2))
      mat_inv(1,3) = + (mat(1,2)*mat(2,3) - mat(1,3)*mat(2,2))
      do i_dim = 1, 3
      do j_dim = i_dim, 3
       mat_inv(i_dim,j_dim) = mat_inv(i_dim,j_dim) / det_mat
       mat_inv(j_dim,i_dim) = mat_inv(i_dim,j_dim)
      end do
      end do

      end subroutine


! ! ! ! ! ! ! ! ! ! ! !

      subroutine trans_mat(mat,mat_trans)
      implicit none
      integer i_dim, j_dim
      double precision mat(3,3), mat_trans(3,3)

      do i_dim=1, 3
         do j_dim=1,3
            mat_trans(i_dim,j_dim)=mat(j_dim,i_dim)
         end do
      end do

      end subroutine


! ! ! ! ! ! ! ! ! ! ! !

      real function determ(mat)
      implicit none
      double precision mat(3,3)
      determ = mat(1,1)*mat(2,2)*mat(3,3) + 2*mat(1,2)*mat(2,3)*mat(3,1)
     & -mat(1,1)*mat(2,3)**2 -mat(2,2)*mat(3,1)**2 -mat(3,3)*mat(1,2)**2
      end function

! ! ! ! ! ! ! ! ! ! ! !

      subroutine mult_mat(mat1,mat2,mult)
      implicit none
      integer i_dim, j_dim, k_dim
      double precision mat1(3,3), mat2(3,3), mult(3,3)

      do i_dim=1, 3
         do j_dim=1,3
            mult(i_dim,j_dim)=0.d0
            do k_dim=1,3
               mult(i_dim,j_dim) = mult(i_dim,j_dim)
     &              + mat1(i_dim,k_dim) * mat2(k_dim,j_dim)
            end do
         end do
      end do

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_elco(scal_2, scal_1)
      implicit none
      integer i_dim, j_dim, k_dim, l_dim, m_dim, n_dim
      parameter (n_dim=3)
      double precision scal_2(6,6), h_1(6,6), scal_1(6), s_2(6,6,6,6)
      double precision vol_1, rot_6(6,6),  eig_6(6,6), smat(6,6)
      common /cvol_1/ vol_1
      real c_11, c_12, c_13, c_14, c_33, c_44, c_66

      call calc_smat(scal_2, scal_1, smat)
      write(*,*) "s_matrix"
      do i_dim = 1, 6
       do j_dim = 1, 6
       scal_2(i_dim,j_dim) = vol_1 * smat(i_dim,j_dim)
       end do
       write(*,'(6e13.5)') (scal_2(i_dim,j_dim),j_dim=1,6)
      end do
      call gaussj(scal_2)
      write(*,*) "El. Const. in GPa"
      do i_dim = 1, 6
       do j_dim = 1, 6
        if (i_dim.gt.3) scal_2(i_dim,j_dim) = scal_2(i_dim,j_dim) / 2
        if (j_dim.gt.3) scal_2(i_dim,j_dim) = scal_2(i_dim,j_dim) / 2
       end do
       write(*,'(6e13.5)') (scal_2(i_dim,j_dim)/72.43,j_dim=1,6)
      end do
      write(*,*) "# c_11,  c_12,  c_13,  c_14,  c_33,  c_44,  c_66
     &( Quartz ) "
      c_11 = ( scal_2(1,1) + scal_2(2,2) ) / 72.43  / 2
      c_12 = scal_2(1,2) / 72.43
      c_13 = ( scal_2(1,3) + scal_2(2,3) ) / 72.43 / 2
      c_14 = ( scal_2(1,5) - scal_2(2,5) + scal_2(4,6) ) / 72.43 / 3
      c_33 = scal_2(3,3) / 72.43
      c_44 = ( scal_2(5,5) + scal_2(6,6) ) / 72.43 / 2
      c_66 = scal_2(4,4) / 72.43
      write(*,"(7f7.1)") c_11, c_12, c_13, c_14, c_33, c_44, c_66

      end subroutine

! ! ! ! ! ! ! ! ! ! ! !

      subroutine calc_smat(scal_2, scal_1, smat)
      implicit none
      integer i_dim, j_dim, k_dim, l_dim, n_dim, iv_dim, jv_dim,
     &voi_ind, a_dim, b_dim
      parameter (n_dim=3)
      double precision tk, scal_2(6,6), h_mat(3,3), scal_1(6),
     &s_2(3,3,3,3), h_i(3,3), e_2(3,3,3,3), smat(6,6)

      call trans_to_tens(scal_1, h_mat)
      call inv_mat(h_mat, h_i)

      do i_dim = 1, n_dim
      do j_dim = 1, n_dim
```

```
        iv_dim = voi_ind(i_dim,j_dim)
        do k_dim = 1, n_dim
        do l_dim = 1, n_dim
         jv_dim = voi_ind(k_dim,l_dim)
         s_2(i_dim, j_dim, k_dim, l_dim) = scal_2(iv_dim,jv_dim)
        end do
        end do
       end do
       end do

       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
       do k_dim = 1, n_dim
       do l_dim = 1, n_dim
        e_2(i_dim, j_dim, k_dim, l_dim) = 0.
        do a_dim = 1, n_dim
        do b_dim = 1, n_dim
         e_2(i_dim,j_dim,k_dim,l_dim) =
     &   e_2(i_dim,j_dim,k_dim,l_dim) +
     &   s_2(i_dim,a_dim,b_dim,l_dim)*h_i(a_dim,j_dim)*h_i(b_dim,k_dim) +
     &   s_2(i_dim,a_dim,k_dim,b_dim)*h_i(a_dim,j_dim)*h_i(b_dim,l_dim) +
     &   s_2(a_dim,j_dim,b_dim,l_dim)*h_i(a_dim,i_dim)*h_i(b_dim,k_dim) +
     &   s_2(a_dim,j_dim,k_dim,b_dim)*h_i(a_dim,i_dim)*h_i(b_dim,l_dim)
        end do
        end do
        e_2(i_dim, j_dim, k_dim, l_dim) =
     & e_2(i_dim, j_dim, k_dim, l_dim) / 4
       end do
       end do
       end do
       end do

       do i_dim = 1, n_dim
       do j_dim = 1, n_dim
        iv_dim = voi_ind(i_dim,j_dim)
        do k_dim = 1, n_dim
        do l_dim = 1, n_dim
         jv_dim = voi_ind(k_dim,l_dim)
         smat(iv_dim,jv_dim) = e_2(i_dim, j_dim, k_dim, l_dim)
        end do
        end do
       end do
       end do

       end subroutine

! ! ! ! ! ! ! ! ! ! !

       integer function voi_ind(i_dim,j_dim)
       integer i_dim, j_dim
       if (i_dim.eq.j_dim) then
        voi_ind = i_dim
       else if (i_dim*j_dim.eq.2) then
        voi_ind = 4
       else if (i_dim*j_dim.eq.6) then
        voi_ind = 5
       else if (i_dim*j_dim.eq.3) then
        voi_ind = 6
       else
        stoP "Ind. in voi_ind w.o. sense"
       end if
       end function
```

calcstrain.f

```
      program calcstrain
      implicit none

      integer i_dim, j_dim, n_dim, e_dim
      parameter (n_dim=3, e_dim=3)

      double precision h(3,3), h0(3,3), ht(3,3), h0i(3,3), h0ti(3,3),
     &     h0t(3,3)
      double precision s1(3,3), s2(3,3), strain(3,3)
      double precision x(8), scal(n_dim + e_dim), y(10),
     &     pol_total(n_dim), pol_dipoles(n_dim)

      write(*,*) "h0(1,1) ?"
      read(*,*) h0(1,1)
      write(*,*) "h0(2,2) ?"
      read(*,*) h0(2,2)
      write(*,*) "h0(3,3) ?"
      read(*,*) h0(3,3)
      write(*,*) "h0(1,2) ?"
      read(*,*) h0(1,2)
      write(*,*) "h0(2,3) ?"
      read(*,*) h0(2,3)
      write(*,*) "h0(1,3) ?"
      read(*,*) h0(1,3)

      h0(2,1) = h0(1,2)
      h0(3,2) = h0(2,3)
      h0(3,1) = h0(1,3)

      call inv_mat(h0,h0i)
      call trans_mat(h0,h0t)
      call inv_mat(h0t,h0ti)

  100 continue
      read(60,'(30e15.7)',end=200) (x(i_dim), i_dim=1, 8),
     &  (scal(j_dim), j_dim=1, n_dim+e_dim), (y(i_dim), i_dim=1,10),
     &  (pol_total(i_dim), i_dim=1,3), (pol_dipoles(i_dim), i_dim=1,3)

      call trans_to_tens(scal,h)
      call trans_mat(h,ht)
      call mult_mat(h,h0i,s1)
      call mult_mat(ht,s1,s2)
      call mult_mat(h0ti,s2,strain)

      do i_dim=1,3
         do j_dim=1,3
            if (i_dim.eq.j_dim)
     &           strain(i_dim,j_dim) = strain(i_dim,j_dim) - 1.d0
            strain(i_dim,j_dim) = 0.5d0 * strain(i_dim,j_dim)
         end do
      end do

      write(59,'(30e15.7)') x(1), strain(1,1), strain(2,2),
     &     strain(3,3), strain(1,2), strain(2,3), strain(1,3)
```

```
      goto 100
  200 continue

      end program

! ! ! ! ! ! ! ! !

      subroutine trans_to_tens(vector,matrix)
      implicit none
      double precision matrix(3,3), vector(6)
      matrix(1,1) = vector(1)
      matrix(2,2) = vector(2)
      matrix(3,3) = vector(3)
      matrix(1,2) = vector(4)
      matrix(2,1) = vector(4)
      matrix(2,3) = vector(5)
      matrix(3,2) = vector(5)
      matrix(3,1) = vector(6)
      matrix(1,3) = vector(6)
      end subroutine

! ! ! ! ! ! ! ! ! ! !

      subroutine trans_mat(mat,mat_trans)
      implicit none
      integer i_dim, j_dim
      double precision mat(3,3), mat_trans(3,3)

      do i_dim=1, 3
         do j_dim=1,3
            mat_trans(i_dim,j_dim)=mat(j_dim,i_dim)
         end do
      end do

      end subroutine


! ! ! ! ! ! ! ! ! ! !

      real function determ(mat)
      implicit none
      double precision mat(3,3)
      determ = mat(1,1)*mat(2,2)*mat(3,3) + 2*mat(1,2)*mat(2,3)*mat(3,1)
     & -mat(1,1)*mat(2,3)**2 -mat(2,2)*mat(3,1)**2 -mat(3,3)*mat(1,2)**2
      end function

! ! ! ! ! ! ! ! ! ! !

      subroutine mult_mat(mat1,mat2,mult)
      implicit none
      integer i_dim, j_dim, k_dim
      double precision mat1(3,3), mat2(3,3), mult(3,3)

      do i_dim=1, 3
         do j_dim=1,3
            mult(i_dim,j_dim)=0.d0
            do k_dim=1,3
               mult(i_dim,j_dim) = mult(i_dim,j_dim)
     &              + mat1(i_dim,k_dim) * mat2(k_dim,j_dim)
            end do
         end do
      end do

      end subroutine


! ! ! ! ! ! ! ! ! ! !

      subroutine inv_mat(mat,mat_inv)
      implicit none
      integer i_dim, j_dim, sign
      double precision mat(3,3), mat_inv(3,3), det_mat
      real determ
      double precision vol_1
      common /cvol_1/ vol_1

      det_mat = determ(mat)
      vol_1 = det_mat

      do i_dim = 1, 3
      do j_dim = 1, 3
       mat_inv(i_dim,j_dim)  = 0.d0
      end do
      end do
      mat_inv(1,1) = mat(2,2)*mat(3,3) - mat(2,3)**2
      mat_inv(2,2) = mat(3,3)*mat(1,1) - mat(3,1)**2
      mat_inv(3,3) = mat(1,1)*mat(2,2) - mat(1,2)**2
      mat_inv(1,2) = - (mat(2,1)*mat(3,3) - mat(2,3)*mat(3,1))
      mat_inv(2,3) = - (mat(3,2)*mat(1,1) - mat(3,1)*mat(1,2))
      mat_inv(1,3) = + (mat(1,2)*mat(2,3) - mat(1,3)*mat(2,2))
      do i_dim = 1, 3
      do j_dim = i_dim, 3
       mat_inv(i_dim,j_dim) = mat_inv(i_dim,j_dim) / det_mat
       mat_inv(j_dim,i_dim) = mat_inv(i_dim,j_dim)
      end do
      end do

      end subroutine


! ! ! ! ! ! ! ! ! ! !

      integer function voi_ind(i_dim,j_dim)
      integer i_dim, j_dim
      if (i_dim.eq.j_dim) then
       voi_ind = i_dim
      else if (i_dim*j_dim.eq.2) then
       voi_ind = 4
      else if (i_dim*j_dim.eq.6) then
       voi_ind = 5
      else if (i_dim*j_dim.eq.3) then
       voi_ind = 6
      else
       stoP "Ind. in voi_ind w.o. sense"
      end if
      end function
```

List of Figures

Bibliography

[1] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford Science Publications, Oxford, 1987.

[2] D. Andrault, G. Fiquet, F. Guyot, and M. Hanfland. *Science*, 282:720, 1998.

[3] J. C. Axe and G. Shirane. *Phys. Rev. B*, 1:342, 1970.

[4] J. P. Bachheimer and G. Dolino. *Phys. Rev. B*, 11:3195, 1975.

[5] E. J. Banda, R. A. Craven, R. D. Parks, P. M. Horn, and M. Blume. *Solid State Comm.*, 17:11, 1975.

[6] A. G. Beda. *Soviet Physics - Solid State*, 9:5, 1967.

[7] M. Benoit and W. Kob. *Europhys. Lett.*, 60:269, 2002.

[8] K. Binder. *Phys. Rev. Lett.*, 47:693, 1981.

[9] W. Bragg and R. E. Gibbs. *Proc. Roy. Soc. London*, A109:414, 1925.

[10] W. G. Cady. *Piezoelectricity*. Dover Publications, New York, 1964.

[11] R. Car and M. Parrinello. *Phys. Rev. Lett.*, 55:2471, 1985.

[12] M. A. Carpenter, R. J. Hemley, and H. kwang Mao. *J. Geophys. Res.*, 105:10807, 2000.

[13] M. A. Carpenter, E. K. H. Salje, A. Graeme-Barber, B. Wrucki, M. T. Dove, and K. S. Knight. *Am. Mineral.*, 83:2, 1998.

[14] E. C. T. Chao, J. J. Fahey, J. Littler, and D. J. Milton. *J. Geophys. Res.*, 67:419, 1962.

[15] R. E. Cohen. *Geophys. Res. Lett.*, 14:37, 1987.

[16] B. Coluzzi and P. Verrocchio. *J. Chem. Phys.*, 116:3789, 2002.

[17] R. K. Cook and P. G. Weissler. *Phys. Rev.*, 80:4, 1950.

[18] D. W. J. Cruickshank. *J. chem. Soc.*, 1077:5486, 1961.

[19] E. Demiralp, 2001. private communication.

[20] E. Demiralp, T. Cagin, and W. A. Goddard III. *Phys. Rev. Lett.*, 82:1708, 1999.

[21] T. Demuth, Y. Jeanvoine, J. Hafner, and J. G. Angyan. *J. Phys.: Condens. Matter*, 11:3833, 1999.

[22] G. Dolino, B. Berge, M. Vallade, and F. Moussa. *J. de Physique I*, 2:1461, 1992.

[23] W. A. Dollase. *Acta Cryst.*, 23:617, 1967.

[24] M. T. Dove, D. A. Kreen, A. C. Hannon, and I. P. Swainson. *Phys. Chem. Miner.*, 24:311, 1997.

[25] F. Ercolessi and J. B. Adams. *Europhys. Lett.*, 26:583, 1994.

[26] P. P. Ewald. *Ann. Phys.*, 64:253, 1921.

[27] P. W. Fowler and P. A. Madden. *Phys. Rev. B*, 31:5443, 1985.

[28] D. Frenkel and B. Smit. *Understanding Molecular Simulation*. Academic Press, London, 2nd edition, 2002.

[29] J. D. Gale. *Phil. Mag. B*, 73:3, 1996.

[30] J. D. Gale. *J. Chem. Soc. Faraday Trans.*, 93:629, 1997.

[31] J. D. Gale. *Mol. Simul.*, 29:291, 2003.

[32] F. Galeener. *Phys. Rev. B*, 19:4292, 1979.

[33] C. W. Gear. *Numerical Initial Value Problems in ordinary Differential Equations*. Prentice Hall, Englewood Cliffs, NJ, 1971. Chap. 9.

[34] G. V. Gibbs, M. M. Hamil, S. J. Louisnathan, L. S. Bartell, and H. Yow. *Amer. Mineral.*, 57:1578, 1972.

[35] H. Grimm and B. Dorner. *J. Phys. Chem. Solids*, 36:407, 1975.

[36] J. M. Haile. *Molecular Dynamics Simulation: Elementary Methods*. John Wiley and Sons, Inc., New York, wiley professional paperback series edition, 1997.

[37] J. Haines, O. Cambon, D. A. Keen, M. G. Tucker, and M. T. Dove. *Appl. Phys. Lett.*, 81:2968, 2002.

[38] J. Haines, J. M. Léger, F. Gorelli, and M. Hanfland. *Phys. Rev. Lett.*, 87:155503, 2001.

[39] P. J. Heaney. Structure and chemistry of the low-pressure silica polymorphs. In P. J. Heaney, C. T. Prewitt, and G. V. Gibbs, editors, *Silica - Physical Behaviour, Geochemistry, and Materials Applications*, volume 29 of *Rev. Mineral.*, page 1. Mineral. Soc. of Am., Washington D. C., 1994.

[40] R. H. Hemley, A. P. Jephcoat, H. K. Mao, L. C. Ming, and M. H. Manghnani. *Nature*, 334:52, 1988.

[41] R. J. Hemley. Pressure dependence of raman spectra in sio_2 polymorphs: α-quartz, coesite, and stishovite. In M. H. Manghnani and Y. Syono, editors, *High Pressure Research in Mineral Physics*, volume 39 of *Geophys. Monogr. Ser.*, page 347. AGU, Washington D. C., 1987.

[42] R. J. Hemley, M. D. Jackson, and R. G. Gordon. *Eos Trans. AGU*, 66:357, 1985.

[43] R. J. Hemley, C. T. Prewitt, and K. J. Kingma. High-pressure behaviour of silica. In P. J. Heaney, C. T. Prewitt, and G. V. Gibbs, editors, *Silica - Physical Behaviour, Geochemistry, and Materials Applications*, volume 29 of *Rev. Mineral.*, page 41. Mineral. Soc. of Am., Washington D. C., 1994.

[44] R. J. Hill and G. V. Gibbs. *Acta Cryst.*, B35:25, 1979.

[45] U. T. Höchli and J. F. Scott. *Phys. Rev. Lett.*, 26:1627, 1971.

[46] J. Horbach and W. Kob. *Phys. Rev. B*, 60:3169, 1999.

[47] J. B. Jones and E. R. Segnit. *J. Geol. Soc. Australia*, 18:419, 1972.

[48] P. Jund and R. Jullien. *Phys. Rev. Lett.*, 83:2210, 1999.

[49] K. Kihara. *Eur. J. Miner.*, 2:63, 1990.

[50] K. Kihara. *Phys. Chem. Min.*, 19:492, 1993.

[51] K. J. Kingma, R. E. Cohen, R. J. Hemley, and H. K. Mao. *Nature*, 374:243, 1995.

[52] K. J. Kingma, R. J. Hemley, H. K. Mao, and D. R. Veblen. *Phys. Rev. Lett.*, 70:3927, 1993.

[53] K. J. Kingma, H. K. Mao, and R. J. Hemley. *High Press. Res.*, 14:363, 1996.

[54] Klein and Hurlbut. *Manual of Mineralogy, p. 527*. John Wiley & Sons, 21st edition, 1962.

[55] M. B. Kruger and R. Jeanloz. *Science,* 249:647, 1990.

[56] D. J. Lacks and R. G. Gordon. *J. Geophys. Res.,* 98:22147, 1993.

[57] A. Laio, S. Bernard, G. L. Chiarotti, S. Scandolo, and E. Tosatti. *Science,* 287:1027, 2000.

[58] D. P. Landau and K. Binder, editors. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press, Cambridge, 2000.

[59] S. J. Louisnathan and G. V. Gibbs. *Amer. Mineral.,* 57:1614, 1972.

[60] M. H. M. Wilson, P. A. Madden and C. A. Angell. *Phys. Rev. Lett.,* 77:4023, 1996.

[61] C. H. Macgillavry, G. D. Rieck, and K. Lonsdale, editors. *International Tables of X-Ray Crystallography*. Kynoch Press, Birmingham, 1962.

[62] H. K. Mao, J. Shu, J. Hu, and R. J. Hemley. *Eos Trans. AGO, Fall Meet. Suppl.,* 75(44):662, 1994.

[63] R. M. Martin. *Phys. Rev. B,* 5:1607, 1972.

[64] L. McNeil and M. Grimsditch. *Phys. Rev. Lett.,* 68:83, 1992.

[65] M. H. Müser. Average structure vs. real structure: Molecular dynamics studies of silica. In K. A. Gernoth and M. L. Ristig, editors, *Particle Scattering, X-Ray Diffraction, and Microstructure of Solids and Liquids,* volume 610 of *Lecture Notes in Physics,* pages 75–100. Springer, Berlin, 2003.

[66] M. H. Müser and K. Binder. *Phys. Chem. Miner.,* 28:746, 2001.

[67] L. Nagel and M. O'Keeffe. *Mater. Res. Bull.,* 6:1317, 1971.

[68] E. L. Nave, H. E. Stanley, and F. Sciortino. *Phys. Rev. Lett.,* 88:035501, 2002.

[69] S. Nose. *J. Chem. Phys.,* 81:511, 1984.

[70] R. G. Parr and W. Yang. *Density-Functional Theory of Atoms and Molecules*. Oxford Science, Oxford, 1989.

[71] M. Parrinello and A. Rahman. *Phys. Rev. Lett.,* 45:1196, 1980.

[72] M. Parrinello and A. Rahman. *J. Chem. Phys.,* 76:2662, 1982.

[73] A. Pasquarello and R. Car. *Phys. Rev. Lett.,* 80:5145, 1998.

[74] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in Fortran*. Cambridge University Press, Cambridge, 2nd edition, 1992.

[75] D. L. Price and J. M. Carpenter. *J. Non-Cryst. Solids*, 92:153, 1987.

[76] A. K. Rappé, 2001. private communication.

[77] A. K. Rappé and W. A. Goddard III. *J. Phys. Chem.*, 95:3358, 1991.

[78] S. W. Rick, S. J. Stuart, and B. J. Berne. *J. Chem. Phys.*, 101:6141, 1994.

[79] G. Roma, Y. Limoge, and S. Baroni. *Phys. Rev. Lett.*, 86:4564, 2001.

[80] N. L. Ross, J. F. Shu, and R. M. Hazen. *Am. Min.*, 75:739, 1990.

[81] E. K. H. Salje, A. Ridgwell, B. Güttler, B. Wruck, M. T. Dove, and G. Dolino. *J. Phys. Cond. Mat.*, 4:571, 1992.

[82] R. T. Sanderson. *Science*, 114:670, 1951.

[83] P. Schöffel. Diplomarbeit, Johannes Gutenberg-Universität Mainz, 2000.

[84] D. Sebastiani. PhD thesis, Universität Stuttgart, 2001. http://elib.uni-stuttgart.de/opus/volltexte/2001/792.

[85] J. Sefcik, E. Demiralp, T. Cagin, and W. A. Goddard III. *J. Comput. Chem.*, 23:1507, 2002.

[86] G. S. Smith. *Acta Cryst.*, 16:542, 1963.

[87] D. R. Spearing, I. Farnan, and J. F. Stebbins. *Phys. Chem. Miner.*, 19:307, 1992.

[88] M. Sprik and M. L. Klein. *J. Chem. Phys.*, 89:7556, 1988.

[89] S. M. Stishov and S. V. Popova. *Geochem. (USSR)*, 10:923, 1961.

[90] P. Tangney and S. Scandolo. *J. Chem. Phys.*, 117:8898, 2002.

[91] S. N. Taraskin and S. R. Elliot. *Phys. Rev. B*, 56:8605, 1997.

[92] Y. Tezuka, S. Shin, and M. Ishigame. *Phys. Rev. Lett.*, 66:2356, 1991.

[93] M. P. Tosi. *Solid St. Phys.*, 16:1, 1964.

[94] A. Toukmaji, C. Sagui, J. Board, and T. Darden. *J. Chem. Phys.*, 113:10913, 2000.

[95] J. S. Tse and D. D. Klug. *Phys. Rev. Lett.*, 67:3559, 1991.

[96] J. S. Tse and D. D. Klug. *J. Chem. Phys.*, 95:9176, 1991.

[97] J. S. Tse, D. D. Klug, and Y. L. Page. *Phys. Rev. Lett.*, 69:3647, 1992.

[98] Y. Tsuchida and T. Yagi. *Nature*, 340:217, 1989.

[99] Y. Tsuchida and T. Yagi. *Nature*, 347:267, 1990.

[100] S. Tsuneyuki, H. Aoki, M. Tsukada, and Matsui. *Phys. Rev. Lett.*, 64:776, 1990.

[101] B. van Beest, G. Kramer, and R. van Santen. *Phys. Rev. Lett.*, 64:1955, 1990.

[102] W. Voigt. *Lehrbuch der Kristallphysik*. Teubner, Berlin, 1910.

[103] K. Vollmayr, W. Kob, and K. Binder. *Phys. Rev. B*, 54:15808, 1996.

[104] K. Vollmayr, J. D. Reger, M. Scheucher, and K. Binder. *Z. Phys. B*, 91:113, 1993.

[105] R. M. Wentzcovitch, C. da Silva, J. R. Chelikowsky, and N. Binggeli. *Phys. Rev. Lett.*, 80:2149, 1998.

[106] M. Wilson and P. A. Madden. *J. Phys.: Condens. Matt.*, 5:2687, 1993.

[107] A. F. Wright and M. S. Lehmann. *J. Solid. State. Chem.*, 36:372, 1981.