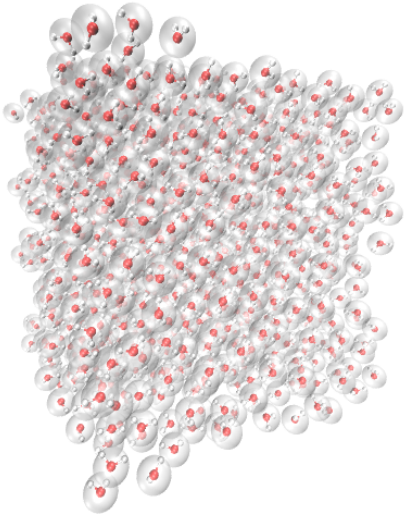
Stochastic and Deterministic Parametrization of Coarse Graining models for Molecular Systems



University of Crete, Department of Mathematics and Applied Mathematics

Anthony Chazirakis

Committee
Associate Prof. Vagelis Harmandaris
Prof. George Zouraris
Dr. Evangelia Kalligiannaki

October 2017

Abstract

Molecular dynamics (MD) simulations study the physical movement of atoms, providing valuable insight for a variety of physical systems. Despite of the modern advances to the available computational resources, the length and time scales of the physical systems under study are still limited. The only remedy is the use of mathematical methods in order to reduce the dimensionality of the physical system under study. The most often used methodology for doing that is Coarse Graining (CG). Coarse graining methodologies have both mathematical as well as computational and numerical challenges. We experiment with three different methods of coarse graining, the Iterative Boltzmann Inversion (IBI) method, the Force Matching (FM or MSCG) method and the Relative Entropy (RE) method, for a variety of physical systems and compare the results. The IBI method finds the CG potential that reproduces the atomistic pair correlation function. The FM method finds the CG potential that minimizes the mean distance between the atomistic and the CG forces. The RE method finds the potential that minimizes the difference between the atomistic and CG coordinate probability distributions. The numerical treatment of the IBI and FM methods is deterministic, although it involves calculation of expectations. The RE method not only involves calculation of expectations, but also performs minimization using the stochastic Robins-Monro scheme.

Acknowledgements

I would like to thank Associate Prof. Vageli Harmandari for his ongoing support and guidance, as well as Dr. Evangelia Kalligiannaki for all the help she provided. I would also like to thank Dr. Tsourti Anastasio for the data he shared.

Finally I would like to thank all the many people that helped, or just were there for a nice talk, during the course of my master's studies.

Contents

Chapter 1

Molecular Dynamics

1.1 Introduction

Molecular dynamics (MD) simulations study the physical movement of atoms and molecules. They do so by treating the problem as an N-body problem. If the interaction potentials between atoms of the system are properly modeled, then the equations of motion can be numerically integrated to acquire the evolution of the system through time.

Classical molecular dynamics methods disregard the fluctuations of the electron state of the atoms and use properly parametrized force fields to model the occurring forces. This approach is based on the Born-Oppenheimer approximation and usually the ground state potential energy is represented in the force field. Molecular dynamics simulations have the advantage of speed as the interactions of the vast amount of electrons is not studied but approximated. In our simulations the classical molecular dynamics method is used. The energy potentials used in atomistic MD simulations though, come from quantum mechanical ab-initio calculations.

Molecular dynamics are used to study phenomena that cannot otherwise be studied, or that are expensive or dangerous to study by other means. The detailed atomistic view acquired from such simulations cannot be acquired by other experimental means and valuable insight on the details of the dynamics and the arrangement of matter can be gained. This aspect is especially important at studies of biological and organic matter.

Molecular dynamics simulations can be used to predict thermodynamic and structural properties of materials and substances, for example polymers and drugs, and are thus considered a valuable tool in material and drug design. Apart from the flexibility offered by such simulated experiments, where one can test arbitrarily hypothetical configurations, the cost of research is also significantly lower, as simulating a multitude of

experiments is usually cheaper than having to actually perform them. Another advantage is that one can perform virtual experiments involving very dangerous substances without having to physically handle them, or impose extremely harsh environmental conditions without having to manifest them.

Molecular dynamics suffer from the complexity of the calculations involved, and are thus limited regarding the size of the simulated systems, and the duration of the simulated experiment. Typical simulations contain less than one million atoms and the physical time simulated is less than a few microseconds. The continuous growth of available computing power, most notably the common availability of clusters of parallel processing units, multiprocessors and programmable graphics processing units, has enabled the simulation of much larger systems. Largely parallel computing is mandatory in molecular dynamics simulations, when large physical systems are to be examined, and thankfully it has been available during the past few years and is becoming all the more available.

Still there is the problem of scale in time and length. One often needs to simulate for times much longer, or for systems much larger than the typical MD scales. Such systems are usually biological systems where characteristic times and lengths are large. Multiscale MD tackles this problem by removing degrees of freedom from the system, through approximations, that hopefully capture the important physical phenomena. Reducing the number of particles of a simulation has a beneficial side effect on the time scale of the simulation. Aggregate atoms vibrate slower than their constituting parts, allowing a greater integration time step. The overall gain in both length and time scales is of orders of magnitude. In this work we experiment with three different methods of systematically reducing degrees of freedom in a MD simulation, the Iterative Boltzmann Inversion method, the Force Matching method and the Relative Entropy method.

1.2 Equations of motion

The molecular dynamics simulations [9] we perform is the numerical solution of the equations of motion of an N-body system. The system is comprised of N "particles" which are treated as solid point masses.

The interaction between particles of the system may originate from short range var der Waals interactions, from long range electrostatic interactions, or from intrarmolecular interactions due to chemical bonds.

The equations of motion of an N-body system can be derived using the Lagrangian formalism. If the generalized coordinates describing the atom positions are

$$\boldsymbol{q}(t) = \{\boldsymbol{q}_1(t), \ldots, \boldsymbol{q}_N(t)\}$$

Then the equations of motion for the system are derived by the following set of differential equations,

$$\frac{\partial L}{\partial \boldsymbol{q}_i} = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\boldsymbol{q}}_i}\right) \quad ,$$

where $\dot{\boldsymbol{q}}_i = \partial \boldsymbol{q}_i / \partial t$ is the generalized velocity of particle i, and L is the Lagrangian of the system, defined by the total kinetic energy T, and the total potential energy U as

$$L = L(\boldsymbol{q}, \dot{\boldsymbol{q}}, t) = T - U \quad .$$

The generalized momenta are defined as

$$\boldsymbol{p}_i = \frac{\partial L}{\partial \dot{\boldsymbol{q}}_i} \quad , \quad i = 1, ..., N \quad .$$

The value of the Lagrangian formulation is that one can easily include constraints in the Lagrangian of the system, and then derive the equations of motion of the constrained system. Such constraints may be fixed bond lengths, or fixed system temperature.

The Hamiltonian formalism can also be used to derive the equations of motion, where generalized momenta are used in place of generalized velocities. The Hamiltonian of the system is defined as

$$H(\boldsymbol{q}, \boldsymbol{p}) = \sum_i \boldsymbol{p}_i(\dot{\boldsymbol{q}}_i) - L$$

The generalized coordinates $\boldsymbol{q}_i$ and momenta $\boldsymbol{p}_i$ satisfy Hamilton's equations

$$\dot{\boldsymbol{q}}_i = \partial H / \partial \boldsymbol{p}_i \quad , \quad \dot{\boldsymbol{p}}_i = -\partial H / \partial \boldsymbol{q}_i \quad .$$

When the potential U of a system is independent of velocities and has no explicit dependence of time, the Hamiltonian of a system equals the total energy, kinetic and potential, of the system.

$$H = T(\boldsymbol{p}) + U(\boldsymbol{q})$$

In Cartesian coordinates the equations of Hamilton produce Newton's equations of motion.

$$\dot{\boldsymbol{q}}_i = \boldsymbol{v}_i = \boldsymbol{p}_i / m_i \quad , \quad \dot{\boldsymbol{p}}_i = -\nabla_{\boldsymbol{q}_i} U = -\partial U / \partial \boldsymbol{q}_i = \boldsymbol{f}_i \quad ,$$

that give,

$$m_i \ddot{\boldsymbol{q}}_i = \boldsymbol{f}_i \quad ,$$

where $\boldsymbol{f}_i$ is the force acting on particle i and m_i is its mass.

Overall, the equations of motion can be derived in various ways, which can be useful if some constraints need to be applied to the system, but they more or less reduce to the well known Newton's equations of motion. For a system of N unconstrained particles the Hamiltonian equations of motion are $6N$ first order differential equations, that need to be integrated to acquire the system evolution through time. Additional terms may be introduced to the Hamiltonian, representing a thermostat that scales velocities (Nose-Hoover thermostat [25, 14]), or a barostat that scales volume (Andersen [4], Nose-Hoover [15], Martyna-Tuckerman-Klein [21] barostats). We shall not go into such details.

1.3 Molecular Force Field

1.3.1 Introduction

What needs to be defined firstly, is the potential energy of the system (see discussion in the previous section) and thus the force applied to each particle in the system. The force is the potential gradient in respect to the particle generalized coordinates.

The forces applied to each particle are of three different kinds, the short range non-bonded forces, the long range electrostatic forces and the forces due to bonds. The short range non-bonded forces include both the attractive interactions due to the instantaneously induced dipole moments of the electron clouds, and the repulsive interactions due to Pauli's exclusion principle. The long range electrostatic forces are the forces exerted due to each particle's net charge. The bonded forces are of three types, the bond stretching forces that occur when the distance between two bonded particles changes, the bond bending forces that occur when the angle between three bonded particles changes, and the torsional bond forces that occur when four bonded particles are rotated about the central bond. The classification of the bonded forces to three types does not reflect naturally occurring physical forces, but rather a modeling of the total bond forces exerted due to the variance of the geometry of the bond. For a more detailed presentation of the molecular force fields see chapter four of [20].

1.3.2 Electrostatic Non Bonded Forces

The electrostatic forces are long range in nature and are modeled by,

$$U_c^{ij}(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{Q_i \cdot Q_i}{r_{ij}}$$

where U_c^{ij} is the electrostatic potential energy of the particle pair, Q_i and Q_j is the charge of particle i and j respectively, r_{ij} is the distance between the two particles and ϵ_0 is the permittivity of free space.

There is not much to say here, other than that to calculate the electrostatic forces in a molecular system, one has to resort to techniques like the Ewald summation, that screens the charges so as to separate the electrostatic forces to short range forces and long range forces. Short range forces are calculated like the non electrostatic short range forces (discussion at section 1.3.3). Long range forces are calculated by sums at the reciprocal Fourier transformed space. Computationally the Particle Mesh Ewald variant of the algorithm is usually preferred.

1.3.3 Van der Waals Non Bonded Forces

The non electrostatic short range non bonded forces are very often modeled using the Lennard-Jones potential (see figure 1.1), which is of the form,

$$U_{LJ}(r) = \epsilon \left[\left(\frac{r_m}{r} \right)^{12} - 2 \left(\frac{r_m}{r} \right)^{6} \right]$$

where ϵ is the well depth, i.e. the strength of the interaction and r_m is the location of the minimum, i.e. the equilibrium distance.

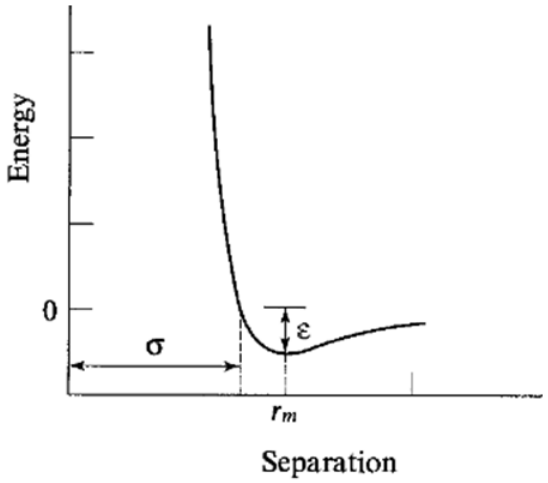


Figure 1.1: The Lennard Jones potential

The attractive term r^{-6} describes the long range van der Waals force. It has a clear physical justification, arising as a result of instantaneously induced dipole interactions. The repulsive term r^{-12} describes the short range Pauli repulsion due to overlapping electron orbitals. It is a good approximation of the exponential form suggested by quantum mechanical calculations and is chosen in favour of computational efficiency, as it can be calculated as the square of the attractive term.

Different pairs of atoms have different sets of parameters (ϵ,r_m) for the LJ potential modeling their interaction. A system of N_{types} different types of atoms requires at least $N_{types}(N_{types} - 1)/2$ sets of LJ parameters to be modeled. Furthermore, as the electrostatic distribution of the atom electrons may vary depending on the bonds it has formed, a physical atom type may be treated as a multitude of distinct atom types in the simulation.

All non-bonded interactions are usually taken to be pair interactions. Three or more body interactions should be included for a more detailed simulation, but that would

significantly increase the computational power required. Alternatively one can use appropriate parameterizations of the two body interactions, not necessarily of an LJ form, to take into account a significant proportion of the many body effects. Such potentials are called effective pair potentials and can be used for more accurate simulations.

To reduce computational effort, a cutoff distance is used for the short range potentials. As the potential falls rapidly to zero, at relatively short distances, one can ignore contributions from interactions of particles that are separated enough. Although this approach does induce some numerical errors, its benefits are of great value. Most of the computation time in the simulation is spent on calculating the non-bonded pairwise interactions. The number of pairs in a system scales as the square of the number of particles in it, so for larger systems the number of pairs varies non-linearly. By using a cutoff distance the number of pairs grows linearly with the number of atoms in the system, so apart from speeding up a simulation of a given size, the cutoff distance enables us to run simulations of much larger systems.

1.3.4 Bond Stretching Forces

The bond stretching forces tend to keep pairs of bonded atoms at a fixed distance. By Taylor expanding an arbitrary potential around its minima it can be shown that a good approximation is in the form of a harmonic oscillator. So the bond stretching potential is typically modeled using the following form,

$$U(l) = \frac{k}{2}(l - l_0)^2 \ ,$$

where l is the particle separation, l_0 is the natural bond length and k defines the strength of the bond.

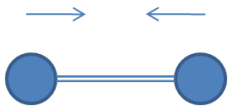


Figure 1.2: A 2-body stretching bond

1.3.5 Bond Bending Forces

The bond bending forces tend to keep the angle θ formed by a triplet of bonded atoms at a fixed value. The bond bending potential is typically modeled using the following form.

$$U(\theta) = \frac{k}{2}(\theta - \theta_0)^2 \ ,$$

where θ_0 is the natural bond angle and k defines the strength of the bond.

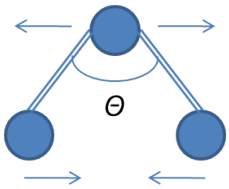


Figure 1.3: A 3-body bending bond

1.3.6 Torsional Bond Forces

Torsional bond forces tend to keep quadruplets of bonded atoms close to equilibrium rotational angles around the central bond. Torsional potentials are typically expressed as cosine series expansions of the following form,

$$U(\omega) = \sum_{n=1}^{N} C_n cos(\omega)^n \ ,$$

where ω is the torsion angle. We have N C_n parameters that define each torsional bond type. N is most often five or nine. Note that there may be more than one torsional potential contributions around a single rotation, for example nine torsional potentials contribute to ethane ($H_3C - CH_3$).

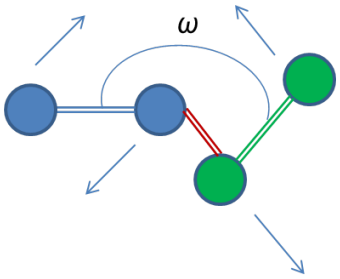


Figure 1.4: A 4-body torsional bond

1.4 Numerical integration of the equations of motion

Having defined a way to model the potential of the system and subsequently the forces applied to each particle, we can integrate the equations of motion to acquire the system's state through time (book [5] §2, book [31] §3.8). The simplest methods used to establish an integration scheme are based on the Taylor series expansion of the positions $\boldsymbol{q}(t)$, velocities $\boldsymbol{v}(t)$, accelerations $\dot{\boldsymbol{v}}(t)$ and so forth of the particles of the system.

1.4.1 Verlet scheme

By Taylor expanding the positions at $t + dt$ and $t - dt$ we get,

$$\boldsymbol{q}(t + dt) = \boldsymbol{q}(t) + dt\ \ \boldsymbol{v}(t) + \frac{dt^2}{2}\ddot{\boldsymbol{q}}(t) + \frac{dt^3}{6}\dddot{\boldsymbol{q}}(t) + O(dt^4)$$
$$\boldsymbol{q}(t - dt) = \boldsymbol{q}(t) - dt\ \ \boldsymbol{v}(t) + \frac{dt^2}{2}\ddot{\boldsymbol{q}}(t) - \frac{dt^3}{6}\dddot{\boldsymbol{q}}(t) + O(dt^4)\ \ .$$

Adding these together we get,

$$\boldsymbol{q}(t + dt) = 2\boldsymbol{r}(t) - \boldsymbol{q}(t - dt) + dt^2\ddot{\boldsymbol{q}}(t) + O(dt^4)\ \ ,$$

where $\ddot{\boldsymbol{q}}(t) = \boldsymbol{f}(t)/m$, $\boldsymbol{f}(t)$ is the force acting of the particle and m is the particle's mass. So if we have the positions at the previous time step and the forces and the positions at the current time step, we can calculate the positions at the next time step. The velocities can be estimated at the half-step

$$\boldsymbol{v}(t + \frac{dt}{2}) = \frac{\boldsymbol{q}(t + dt) - \boldsymbol{q}(t)}{dt}$$

The Verlet scheme has a couple of disadvantages. First of all the velocity is only implicitly obtained. Then it is not a self starting scheme, as the positions at two time steps must be known to start the integration.

1.4.2 Velocity Verlet scheme

The Velocity Verlet scheme overcomes the problem of the definition of the velocities and positions at different time steps. The positions and velocities are obtained from the formulae below,

$$\boldsymbol{q}(t + dt) = \boldsymbol{q}(t) + dt\boldsymbol{v}(t) + \frac{dt^2}{2}\ddot{\boldsymbol{q}}(t)$$
$$\boldsymbol{v}(t + dt) = \boldsymbol{v}(t) + \frac{dt}{2}[\ddot{\boldsymbol{q}}(t) + \ddot{\boldsymbol{q}}(t + dt)]\ \ .$$

Since

$$\boldsymbol{v}(t + \frac{dt}{2}) = \boldsymbol{v}(t) + \frac{dt}{2}\ddot{\boldsymbol{q}}(t)\ \ ,$$

then

$$\boldsymbol{v}(t + dt) = \boldsymbol{v}\left(t + \frac{dt}{2}\right) + \frac{dt}{2}\ddot{\boldsymbol{q}}(t + dt)\ \ .$$

And thus the velocity Verlet integration is a three step process. First the new positions are obtained from the current positions, velocities and forces. Then the velocities at the half time step are obtained from the starting velocities and forces. Finally the new

velocities are obtained from the velocities at the half time step and the forces at the new time step. At the next iteration the forces at the current time step are known, so the stage of the force calculation occurs once in each time step, after the new positions and before the new velocities have been calculated. The velocity Verlet scheme is a second order scheme. It is efficient and accurate enough for molecular dynamics simulations, where the statistical nature of their results does not require very high accuracy. Most importantly it is time reversible, guaranteeing the long time energy conservation, which is much needed in lengthy simulations.

1.5 Maintaining Constant Temperature

The works of this document concern molecular systems at equilibrium at a constant temperature T_0. Various methods exist that keep the simulated system at a constant temperature. The simplest approach applied to realistic simulations is the Berendsen thermostat. The Berendsen thermostat scales the velocities at each time step, so that the rate of change of temperature T is proportional to the temperature difference.

$$\frac{dT}{dt} = \frac{T_0 - T}{\tau} \ ,$$

where τ is the coupling parameter, specifying how tightly the system is coupled to the heat bath. The system reaches the desired temperature exponentially.

$$T = T_0 - C \ e^{-t/\tau} \ .$$

To implement the Berendsen thermostat, the velocities at each time step have to be rescaled by

$$\lambda = \sqrt{1 + \frac{dt}{\tau}\left(\frac{T_0}{T} - 1\right)} \ .$$

A disadvantage of the Berendsen thermostat is that it does not conform to the canonical ensemble. That means that it does not accurately sample the constant temperature system states. Nevertheless it is frequently used. An alternative method of keeping a constant temperature, that accurately samples the constant temperature ensemble, is the Nose-Hoover thermostat. We used the Nose-Hoover thermostat in our works (book [5] §8). We do not present it here as it both tedious and far off topic. We only present the Berendsen barostat, so that the reader has a sense of how things work in a MD simulation and because all simulated systems in this work are in the NVT ensemble, i.e. under equilibrium at constant temperature.

1.6 Computational Approach

In this work we used a variety of MD simulators, a custom in-house parallel C++ code [1], LAMMPS [26] and GROMACS [2]. A typical molecular dynamics simulation begins by defining the molecular system to be simulated. The user has to define all atom types and the functional forms of the non bonded interactions between them. He or she also has to define all bonds present in the molecular system along with their functional forms. The user has also to define the initial placement of the atoms in space, as well as the dimensions of the simulated volume. These are often called the topology of the system.

Having described the constitution of the molecular system, the user then decides under which ensemble he wishes to simulate. That may be under constant volume and energy (NVE), constant volume and temperature (NVT), or constant pressure and temperature (NPT). There are some relevant parameters that have to be defined, like the target temperature and the heat bath coupling parameter. The user also defines the length in time of the integration time step dt and the number of steps that should be integrated.

Finally the user defines what data should be captured throughout the simulation and how often should they be captured. Such data may be scalars like the energy, pressure or temperature of the system, or they may be bulk data like the coordinates and the velocities of the simulated atoms.

All of the above are the necessary input data to an MD simulator. The simulator is then executed, usually in a parallel manner, where the computational load is shared among many processors. The real time length of an MD run is typically of the order of days or weeks and the gathered data is of the order of tenths of gigabytes. After the simulation completes, a thorough analysis of the extracted data follows, in order to reach to any physical conclusions.

A typical MD flowchart for the NVT ensemble is displayed at figure 1.5.

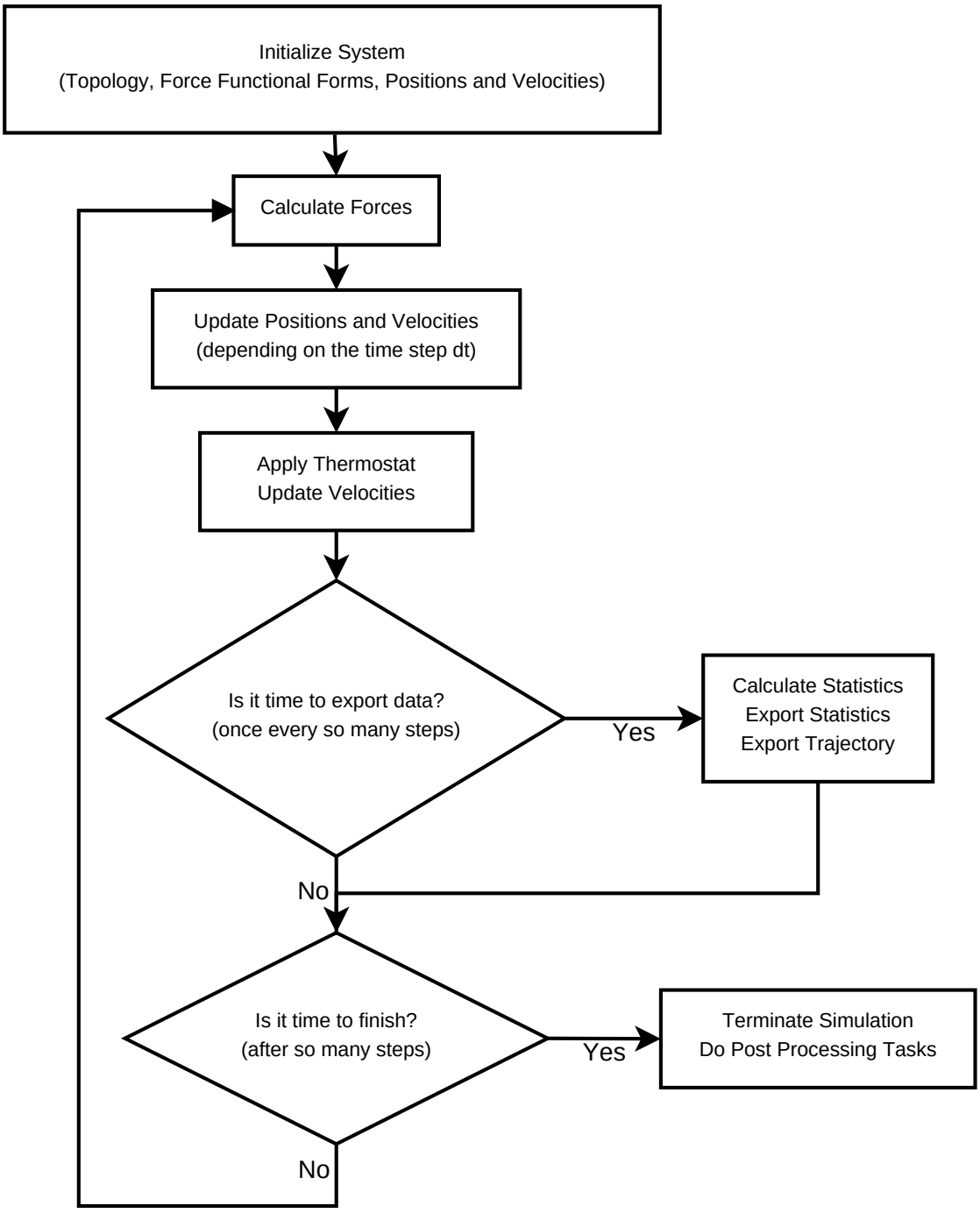


Figure 1.5: A typical MD flowchart

Chapter 2

Multiscale Dynamics of Molecular Systems

2.1 Introduction

Multiscale dynamics refers to studying the dynamics of a system at different time and length scales. During a molecular dynamics simulation of an atomistic system, there is a limit to how large the time step dt can be. This is so that the numerical integration of the equations of motions is accurate. This time step is typically a few femptosecods. Thus we are restricted to simulating short time intervals, given the available computational resources.

The actual time, that a simulation spends in calculating the forces of a single time step, can be reduced by performing parallel MD simulations. Still it cannot be reduced under a limit where the simulation becomes inefficient. Typically a processor can handle as few as half a thousand atoms before the overhead nullifies the gain.

For massive numbers of atoms massive parallelism can be utilized. Simple biological systems, like a single red blood cell, are typically considered huge systems. Although the length scale can be handled by a supercomputer, the time scale cannot, as biological time scales are orders of magnitude larger than atomistic ones. The same goes for large non biological systems of course. For example the relaxation time of long polymer chains occurs in physically long time scales, longer than average MD simulations can handle.

In order to overcome these limits in length and time scales, degrees of freedom must be removed from the simulated systems. New smaller systems, with considerably fewer particle numbers, are derived from the full detail systems. The derived systems must approximate the original ones to a satisfying degree. If this is the case then the benefit is twofold. The integration becomes computationally cheaper due to the reduced number of particles. Also, the time length dt of a single integration step becomes larger, as the

characteristic times of heavier particles are larger than those of lighter ones. As a result one may simulate orders of magnitude longer times and scales. Finally, the decrease of degrees of freedom often simplifies the simulated model and reveals simpler physical relations in it. Different methods exist towards reducing the degrees of freedom. We refer to two of them below, Langevin Dynamics in section 2.2 and Coarse Graining in section 2.3.

Figure 2.1 displays the different scales relevant to the study of molecular systems. Conceptually four scales define four distinct but related approaches. The first scale, the quantum mechanical, is where the derivation of the MD potentials is made. People doing MD are usually different than those providing the all-atom MD potentials. The second scale is where all atom simulations are performed. Out of all atom simulations, effective potentials for systems with aggregate atoms are derived. The third scale is where aggregate atoms are simulated, in order to extract physical information for large and complex systems. The fourth scale is a the continuum level. There, the physical properties of matter retrieved from the third scale are used to model matter. Examples of studies at the fourth scale are fluid dynamics and continuum mechanics. People doing MD are usually different than those working at the continuum scale.

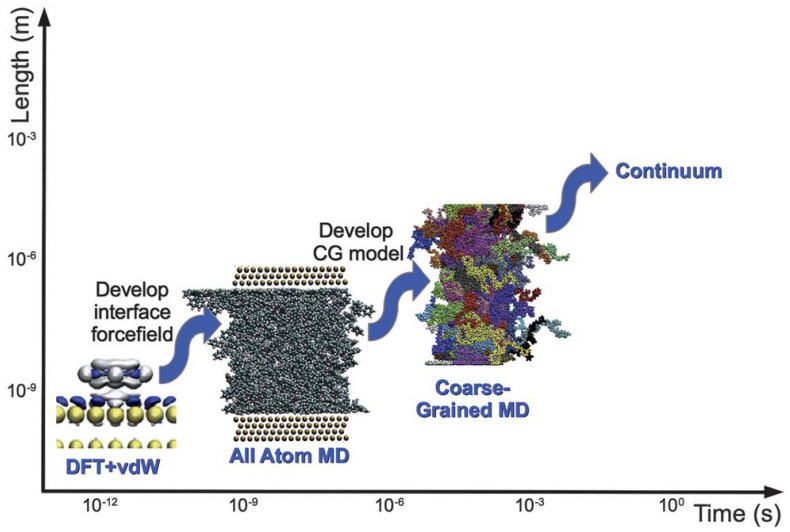


Figure 2.1: Relation of time and length scales of molecular dynamics and related methods.

2.2 Langevin Dynamics

In Langevin dynamics we replace the microscopic details of a system with a friction term and a stochastic term. Langevin dynamics are typically used to model particles

in a solvent. They are based on the homonymous stochastic differential equation. The Langevin equation is,

$$\boldsymbol{M}\ddot{\boldsymbol{Q}} = -\nabla U(\boldsymbol{Q}) - \gamma\dot{\boldsymbol{Q}} + \sqrt{2\gamma k_B T}\boldsymbol{R}(t) \ ,$$

where $\boldsymbol{Q}$ represents the positions of the particles, $\boldsymbol{M}$ is the mass matrix, k_B is Boltzmann's constant, T is the target temperature and $U(\boldsymbol{Q})$ is the many-body potential. The stationary measure of the process $\boldsymbol{Q}(t)$ is the canonical measure with,

$$P(\boldsymbol{Q}) = \frac{1}{Z} \ e^{-U(\boldsymbol{Q})/k_B T}$$
$$Z = \int d\boldsymbol{Q} e^{-U(\boldsymbol{Q})/k_B T} \ .$$

The dumping coefficient is γ and the friction term is $-\gamma\dot{\boldsymbol{Q}}$. The last term acts as a thermostat and also as a source of noise that models the occasional collisions of the system's particles. The simulation is performed at the NVT ensemble and $\boldsymbol{R}(t)$ is a Gaussian process (white noise) satisfying

$$\langle \boldsymbol{R}(t) \rangle = 0$$
$$\langle \boldsymbol{R}(t)\boldsymbol{R}(t') \rangle = \delta(t - t')$$

For molecular systems that can be modeled by Langevin dynamics there is a huge increase to the length and time scales of the simulations that can be performed. Unfortunately most systems do not offer this possibility, since in general the complexity of the molecular systems is beyond that of a solvent and solute.

An example case that can be modeled by Langevin dynamics is that of Brownian motion. This is the motion of usually large particles suspended in a fluid or gas. An example is the motion of dust particles that collide with their surrounding air particles. Another is the motion of a single water molecule that collides with its surrounding water molecules. The trajectory of a single water molecule performing Brownian motion is displayed at figure 2.2.

Notice that Langevin dynamics may be used in conjunction with other methodologies that reduce the degrees of freedom. That means that apart from the removal of the solvent atoms in a simulation, the modeled non solvent atoms may themselves be aggregate atoms.

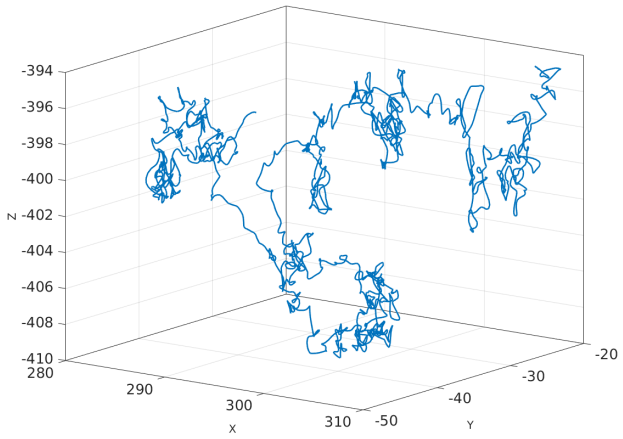


Figure 2.2: Brownian motion of a water molecule.

2.3 Coarse Graining

2.3.1 Introduction

Coarse Graining (CG) is a process used to reduce the degrees of freedom of a molecular system. Several atoms are grouped together and represented by a point particle called the "superatom". This particle's mass and charge is the sum of the contributing atom masses and charges respectively. The CG superatom is usually located at the center of mass of its comprising atoms. After grouping the atoms, various methodologies are used to derive a molecular force field for the CG system, so that it best approximates the original all-atom molecular system.

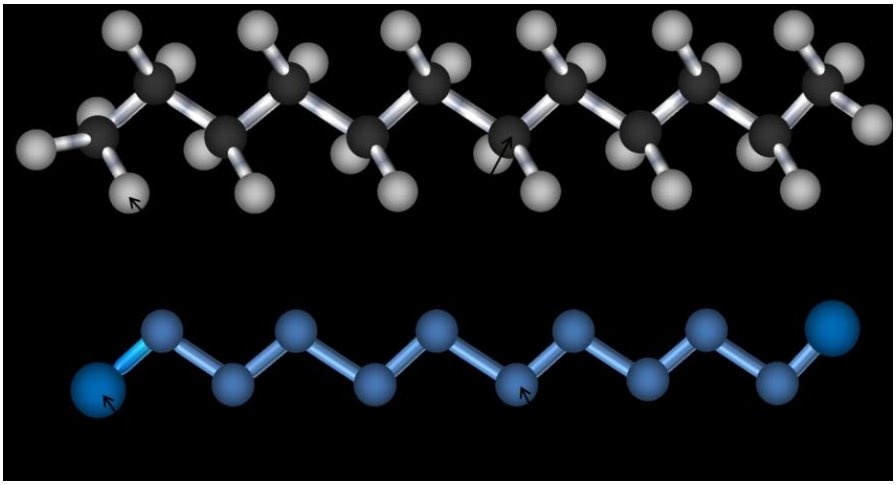


Figure 2.3: All Atom Dodecane vs United Atom Dodecane

A typical CG example, found in the simulation of polymers, is the United Atom (UA) model (as opposed to the All Atom model). The UA model merges all the hydrogens with their bonded carbons, to create the respective CH, CH_2, CH_3 or CH_4 united atoms. The decrease of degrees of freedom is by a factor of 2 to 3. The high symmetry of the carbon-hydrogen groups, and the rigidity of the hydrogen bonds, result in very accurate CG force fields that span large temperature ranges. The relation of the AA model to the UA model is illustrated at figure 2.3.

Another CG example is that of polysterene presented here. The aromatic rings become heavy CG atoms. The CH_2's along with the two halves of the CH's surrounding them become lighter CG atoms. The head CH_3 and the tail CH_2 become two other types of CG atoms, that occur only once. Four types of CG atoms are defined in total as shown in figure 2.4. The peculiar splitting of the CH in two halves helps reduce the mass differences of the CG atoms, which results in a more natural CG system, reducing approximation errors. More CG examples may be found at [30] and [22].

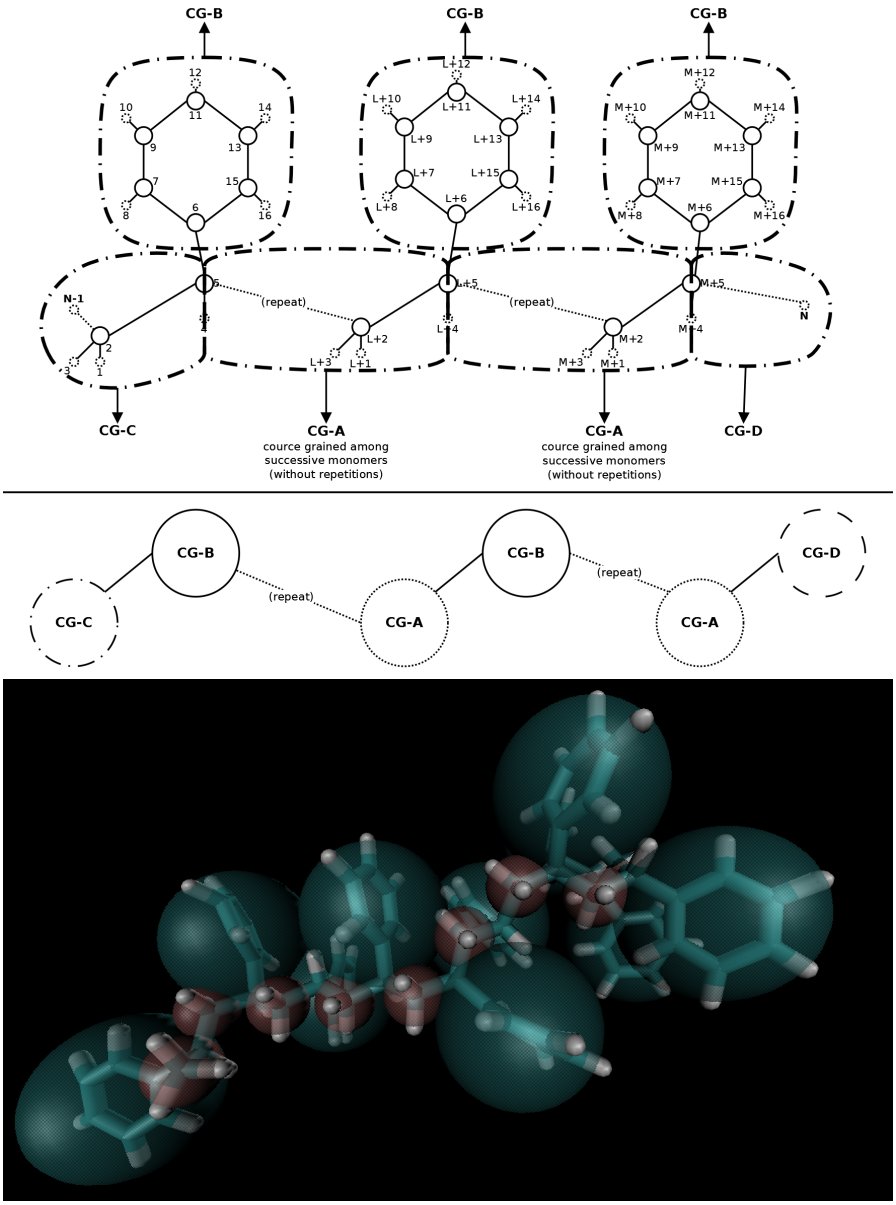


Figure 2.4: Coarse grained all-atom polysterene

We will present three rigorous methods of deriving the CG force field. These are the Iterative Botzmann Invertion method, the Force Matching method and the Relative Entropy method. All of them are defined for molecular systems at equillibrium, under constant temperature, i.e. in the NVT statistical ensemble.

2.3.2 CG Mapping

The way a system's atoms are combined to form a CG system is called the CG map. Let the number of atoms of the detailed atomistic system be N and the coordinates of the atoms be

$$\boldsymbol{q}_i \quad , \quad i = 1, ..., N \ .$$

The number of superatoms of the CG system is M, with $M < N$. Their coordinates are defined by the CG map $\boldsymbol{M} : \boldsymbol{q} \mapsto \boldsymbol{Q}$. For linear $\boldsymbol{M}$

$$\boldsymbol{Q}_I = \boldsymbol{M}_I(\boldsymbol{q}) = \sum_i c_{Ii} \boldsymbol{q}_i \quad , \quad I = 1, ..., M$$

$$\boldsymbol{M}(\boldsymbol{q}) = \{\boldsymbol{M}_1(\boldsymbol{q}), ..., \boldsymbol{M}_M(\boldsymbol{q})\} \ . \tag{2.1}$$

The factors c_{Ii} are usually such that the superatoms are located at the center of mass of their comprising atoms.

2.3.3 Potential of Mean Force

For an atomistic system that is at the NVT ensemble, the probability distribution of the coordinates is,

$$p(\boldsymbol{q}) = \frac{1}{Z_A} e^{-U(\boldsymbol{q})/k_B T} \ , \tag{2.2}$$

where U is the atomistic potential and Z_A is the atomistic partition function,

$$Z_A = \int d\boldsymbol{q} \ \ e^{-U(\boldsymbol{q})/k_B T} \ . \tag{2.3}$$

The probability distribution of the CG system is,

$$P(\boldsymbol{Q}) = \frac{1}{Z_{CG}} e^{-W(\boldsymbol{Q})/k_B T} \ , \tag{2.4}$$

where W is the CG potential and Z_{CG} is the CG partition function,

$$Z_{CG} = \int d\boldsymbol{Q} \ \ e^{-W(\boldsymbol{Q})/k_B T} \ . \tag{2.5}$$

We require that the two distributions are consistent, i.e. the expectation of a quantity $f(\boldsymbol{Q})$ must be the same with respect to both probabilities,

$$\int_{\mathbb{R}^M} f(\boldsymbol{Q}) P(\boldsymbol{Q}) d\boldsymbol{Q} = \int_{\mathbb{R}^N} f(\boldsymbol{M}(\boldsymbol{q})) p(\boldsymbol{q}) d\boldsymbol{q} \ .$$

In order for this to be true, the probability of a CG state must be the integral of the probabilities of the matching atomistic states,

$$P(\boldsymbol{Q}) = \int d\boldsymbol{q} p(\boldsymbol{q}) \delta\left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q}\right) \ , \tag{2.6}$$

where

$$\delta\left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q}\right) = \prod_{I=1}^{M} \delta\left(\boldsymbol{M}_I(\boldsymbol{q}) - \boldsymbol{Q}_I\right) \ ,$$

and δ denotes the Dirac delta.

So then,

$$\begin{aligned} e^{-W(\boldsymbol{Q})/k_B T} &= \frac{Z_{CG}}{Z_A} \int d\boldsymbol{q} e^{-U(\boldsymbol{q})/k_B T} \delta\left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q}\right) \Rightarrow \\ W(\boldsymbol{Q}) &= -k_B T \ ln\left(\int d\boldsymbol{q} e^{-U(\boldsymbol{q})/k_B T} \delta\left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q}\right)\right) - ln\left(\frac{Z_{CG}}{Z_A}\right) \Rightarrow \\ W(\boldsymbol{Q}) &= -k_B T \ ln\left(\int d\boldsymbol{q} e^{-U(\boldsymbol{q})/k_B T} \delta\left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q}\right)\right) + const \ . \end{aligned}$$

$W(\boldsymbol{Q})$ is the many-body potential of mean force (PMF), or effective potential. The PMF can reproduce structural and thermodynamic properties of the atomistic system, but not dynamical properties. Although its analytical form is known, its calculation is not possible due to the high dimentionality of the integral. In principle there are many-body contributions to the PMF, but it is usually approximated by two-body terms. This approximation makes it useful in MD simulations, where pair potentials are used for the sake of performance.

2.4 Direct Boltzmann Inversion

2.4.1 Radial Distribution Function

The radial distribution function (RDF) (book [8] §7.2) is related to the structural properties of a system. The RDF is symbolized as $g(r)$. It is a measure of the probability of finding an atom at distance r from another, or equivalently of how the atomic density varies from the mean density at distance r.

For an isotropic system the simplest definition of the radial distribution function is

$$g(r) = \frac{\rho(r)}{\rho} = \frac{N_{dr}/V_{dr}}{N/V} \ ,$$

where N is the total number of atoms, V the volume of the system, V_{dr} is the volume of a thin spherical shell around r, and N_{dr} is the number of atoms in the spherical shell (see figure 2.5).

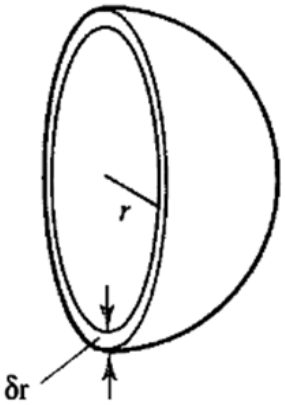


Figure 2.5: Spherical shell of thickness δr, used for the definition of g(r)

The relation of the RDF to the probability that an atom is at distance r from another is

$$g(r) = P(r)/\rho^2$$

Where $P(r)$ is the probability and ρ is the average density of the system.

Note that the two body RDF presented here is a case of the more general n-body correlation function,

$$g^{(n)}(\boldsymbol{q}_1, ..., \boldsymbol{q}_n) = \frac{V^n N!}{N^n (N-n)!} \int P(\boldsymbol{q}_1, ..., \boldsymbol{q}_N) d\boldsymbol{q}_{n+1} ... d\boldsymbol{q}_N \ .$$

A typical plot of an RDF of soft matter is shown at figure 2.6. The first peak is at the distance of the "first neighbours" around the central atom. A less dense area follows until we find the second peak at the distace of the "second neighbours". Such oscillations smooth out until we reach the bulk area, where there is no more structure information. For systems with multiple atom types multiple RDF's are defined, one for each pair type.

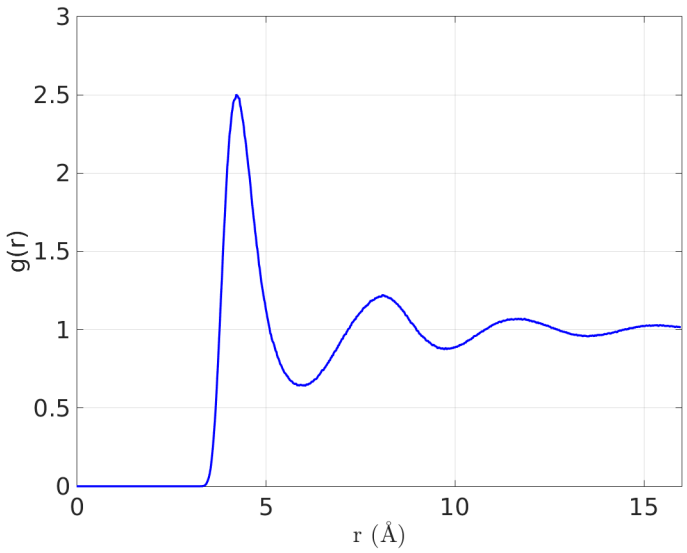


Figure 2.6: A typical fluid Radial Distribution Function (methane at T=100K)

2.4.2 Direct Boltzmann Invertion

The Direct Boltzmann Invertion (DBI) method is a first attempt to approximate the PMF. The reversible work theorem ([8] §7.3) states that the two-body pair PMF (PPMF), which is analogous but distinct from the many-body PMF, is related to the RDF by,

$$W_2(r) = -k_B T \ ln \ g(r) \ ,$$

where $g(r)$ is calculated from the atomistic MD simulation. A similar expression relates the n-body PMF to the n-body correlation function.

A typical intermolecular potential calculated with the DBI method is shown at figure 2.7. The DBI potential is accurate when the CG interactions are isolated and not coupled. Thus, it is useful for nonbonded pair potentials if the CG sites are dilute and for bonded potentials if the CG bonds are stiff [22].

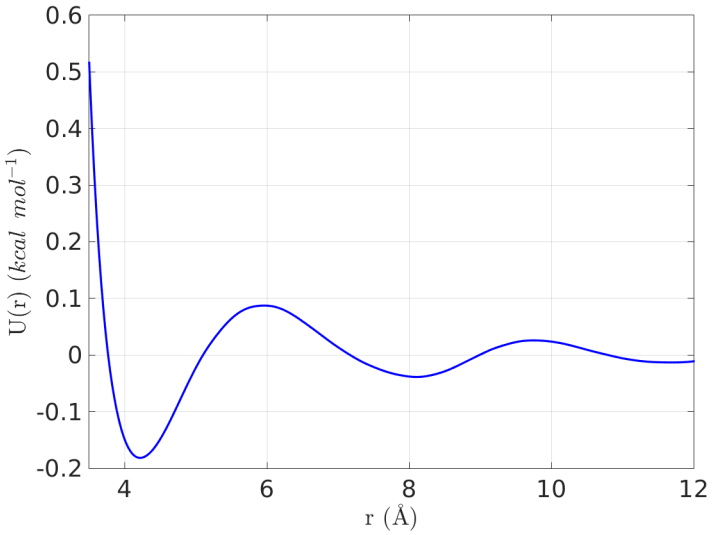


Figure 2.7: A typical intermolecular DBI potential (methabe at T=100K)

For complex CG molecules there are more than one potentials defining its internal structure [30]. For simplicity let's assume there is only one bond type, one angle type and one dihedral type. We denote the bond length with l, the bond angle with θ and the dihedral angle with ω. The probability distribution governing the molecule's internal structure is $P(l, \theta, \omega)$. We assume the probabity distributions factorize (which is a severe approximation),

$$P(l, \theta, \omega) = P(l)P(\theta)P(\omega) \ .$$

These separate probability distributions can be measured by atomistic MD or Monte Carlo (MC) simulations. At the NVT ensemble they should be the Boltzmann distributions,

$$\begin{aligned} P(l) &\sim e^{-U^l(l)/k_BT} \ , \\ P(\theta) &\sim e^{-U^\theta(\theta)/k_BT} \ , \\ P(\omega) &\sim e^{-U^\omega(\omega)/k_BT} \ . \end{aligned}$$

So then the intramolecular potentials are

$$\begin{aligned} U^l(l) &= -k_BT \ \ ln \ \ P(l) + const. \\ U^\theta(\theta) &= -k_BT \ \ ln \ \ P(\theta) + const. \\ U^\omega(\omega) &= -k_BT \ \ ln \ \ P(\omega) + const. \end{aligned}$$

Note that since the normalization of the bond length and bond angle probabilities relies on the integral

$$\int P(l)P(\theta)l^2 sin(\theta) dl d\theta d\phi = 1 \ ,$$

we have to adjust the MD or MC measured probabilities $P'(l)$ and $P'(\theta)$ so that

$$P(l) \sim P'(l)/l^2$$
$$P(\theta) \sim P'(\theta)/sin(\theta) \ .$$

The torsional probability does not require such adjustment. A typical bond length DBI potential is shown at figure 2.8.

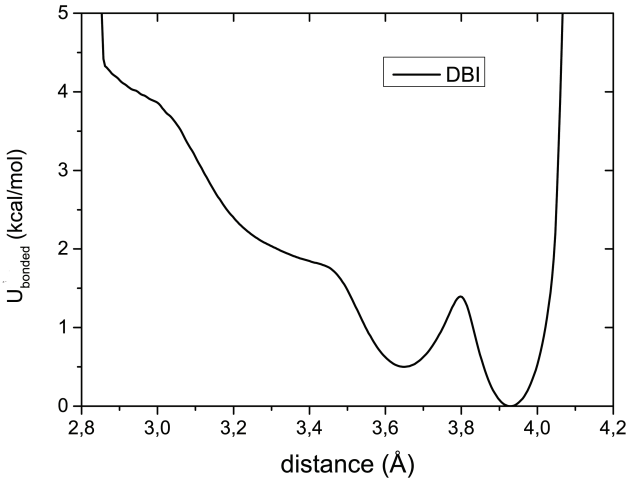


Figure 2.8: CG Hexane (3:1 mapping) bonded DBI potential (taken from [17])

2.5 Iterative Boltzmann Invertion

Recall that for a consistent CG scheme, the CG and atomistic probability distributions must match (equation 2.6). DBI potentials produce CG probability distributions that are close but not equal to the atomistic ones. Bonded probability distributions are much closer than non-bonded ones. The differences occur due to the many-body contributions to the PMF.

The Iterative Boltzmann Inversion (IBI) method [29] starts with the DBI potential and iteratively applies corrections to it, until the atomistic and CG probability distributions match. The target RDF for CG superatoms of type α and β is first measured

from an atomistic MD run. The first CG potential is the DBI potential,

$$U^0_{\alpha\beta}(r) = -k_B T \ ln \ g^T_{\alpha\beta}(r) \ .$$

Then a CG MD run is performed using $U^0_{\alpha\beta}$. From it the $g^0_{\alpha\beta}(r)$ RDF is calculated. The potential at every r is corrected by,

$$\Delta U^0_{\alpha\beta}(r) = k_B T \ ln \left(g^0_{\alpha\beta}(r) / g^T_{\alpha\beta}(r) \right) \ .$$

So the new CG potential is,

$$U^1_{\alpha\beta}(r) = U^0_{\alpha\beta}(r) + k_B T \ ln \left(g^0_{\alpha\beta}(r) / g^T_{\alpha\beta}(r) \right) \ .$$

A new CG MD run is performed using $U^1_{\alpha\beta}(r)$ and $g^1_{\alpha\beta}(r)$ is recalculated. A new correction $\Delta U^1_{\alpha\beta}(r)$ is applied to $U^1_{\alpha\beta}(r)$. In general for $i = 1, 2, ...$

$$U^{i+1}_{\alpha\beta}(r) = U^i_{\alpha\beta}(r) + k_B T \ ln \left(g^i_{\alpha\beta}(r) / g^T_{\alpha\beta}(r) \right) \ . \tag{2.7}$$

This process is repeated until $g^i_{\alpha\beta} \approx g^T_{\alpha\beta}$, i.e.

$$\int dr \left(g^i_{\alpha\beta}(r) - g^T_{\alpha\beta}(r) \right)^2 < \epsilon \ , \tag{2.8}$$

for some tolerance ϵ.

For bonded interactions the IBI scheme is generalized as

$$W^0_\mu(\mu) = -k_B T \ ln \ P(\mu)$$
$$W^{i+1}_\mu(\mu) = W^i_\mu(r) + k_B T \ ln \left(P^i_\mu(\mu) / P^T_\mu(\mu) \right)$$

where μ stands for either the bond distance l, bond angle α or torsional angle θ. A typical $g(r)$ evolution during the execution of the IBI scheme is shown at figure 2.9.

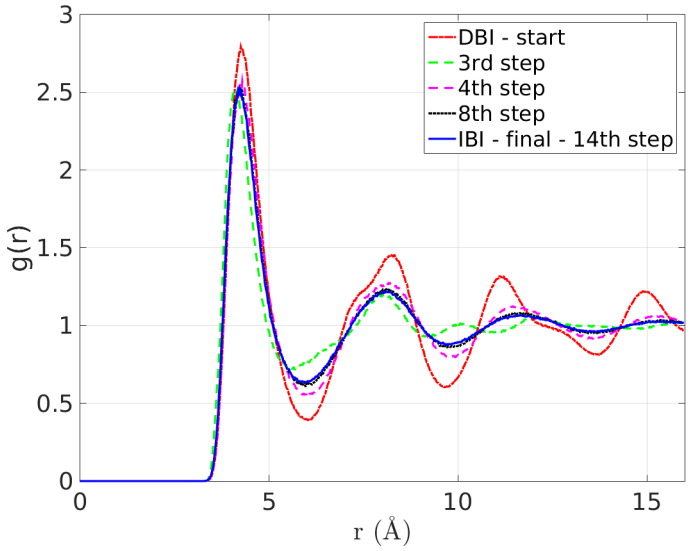


Figure 2.9: A typical $g(r)$ evolution during the execution of the IBI scheme (bulk methane at T=100K)

The uniqueness of the IBI potential that reproduces the atomistic RDF is proven in [13]. The reason that the IBI procedure converges is that an increase in the pair potential function will always cause an overall decrease in the $g(r)$ and vise versa [29]. The uniqueness theorem for the $g(r)$ ([10] page 178) states that for two pair potentials $U^0(r)$ and $U^1(r)$ that differ more than a constant

$$\int dr \left[U^1(r) - U^0(r)\right] \left[g^1(r) - g^0(r)\right] < 0 \ , \tag{2.9}$$

where $g^1(r)$ and $g^2(r)$ are the RDFs occurring from $U^1(r)$ and $U^2(r)$. We can write the total RDF as the sum of the pairwise RDF and the many-body RDF.

$$g(r) = g^p(r) + g^m(r) \tag{2.10}$$

$$g^p(r) = e^{-U(r)/k_BT} \tag{2.11}$$

$$g^m(r) = g^p(r)\left(e^{w(r)/k_BT} - 1\right)$$

where $w(r)$ is the difference between the pair potential and the potential of mean force. From eq. 2.11 we get that

$$\Delta g^p(r)\Delta U(r) < 0$$

Substituting this and eq. 2.10 into eq. 2.9 we get that

$$\int dr\Delta \left[g^p(r) + g^m(r)\right] \Delta U(r) < 0 \tag{2.12}$$

Equation 2.12 now shows that an increase in the pair potential function will always cause an overall decrease in the $g(r)$ and vise versa.

2.6 Force Matching

The force matching approach (or MS-CG approach) [16, 24, 18] is seeking for an effective CG potential by minimizing the average difference between the atomistic and the corresponding CG forces. It approximates the many-body PMF and in the limit where the trial potential is an N-body potential, like the PMF is, this method produces the exact PMF[23, 7]. In practice the method is used to produce pairwise approximations to the PMF.

Recall the symbolism of the atomistic coordinates $\boldsymbol{q}$ and the CG coordinates $\boldsymbol{Q}$. To emphasize on the size of the system we denote in the sequel $\boldsymbol{q}^N$ the number of elements N and in the sequel $\boldsymbol{Q}^M$ the number of elements M, with $M < N$. We shall denote the CG potential with a capital U. The charge Q_I of each CG site I is equal to the sum of charges of the respective atoms. The electrostatic contribution to the CG potential is

$$U^C(\boldsymbol{Q}^M) = \sum_{I \neq J} \frac{1}{4\pi\epsilon_0} \frac{Q_I Q_J}{|\boldsymbol{Q}_I - \boldsymbol{Q}_J|}$$

And the total CG potential is

$$U(\boldsymbol{Q}^M) = U^C(\boldsymbol{Q}^M) + \sum_{\zeta i \gamma} U_{\zeta i}\left(\chi_\zeta\left(\{\boldsymbol{Q}_\gamma\}\right)\right)$$

We need now define some symbols,

ζ : Index of the interaction type, one of: non-bonded, direct bond, angular bond, torsional bond.

i : Index of the functional form of interaction (see definition of $U_{\zeta i}(x)$ below).

γ : Index to the set of CG sites of type ζ with the functional form i. It may be an index to the set of non-bonded CG pairs, or the set of bonded CG pairs, or the set of angle CG atom triplets, or the set of the dihedral CG atom quads

χ_ζ : The scalar variable for the interaction type ζ. It can be a distance, an angle or a torsional angle.

$\{\boldsymbol{Q}\}_\gamma$: The coordinates of all the CG sites that participate in the set γ.

$\chi_\zeta\left(\{\boldsymbol{Q}\}_\gamma\right)$: The scalar variable for the interaction ζ as a function of the relevant CG site positions. It can be a distance, an angle or a torsional angle.

$U_{\zeta i}(\chi)$: The contribution of the potential of the scalar variable χ. It may be like $k(l - l_0)^2$ for direct bonds, $k(\theta - \theta_0)^2$ for angular bonds or $\sum_n C_n cos(\omega)^n$ for torsional bonds.

We also define $F_{\zeta i}$ to be

$$F_{\zeta i} = -\frac{dU_{\zeta i}}{d\chi} \ ,$$

then the CG force on particle I is

$$\boldsymbol{F}_I\left(\boldsymbol{Q}^M\right) = \boldsymbol{F}_I^C\left(\boldsymbol{Q}^M\right) + \sum_{\zeta i\gamma} F_{\zeta i}\left(\chi_\zeta\left(\left\{\boldsymbol{Q}\right\}_\gamma\right)\right)\frac{\partial\chi_\zeta\left(\left\{\boldsymbol{Q}\right\}_\gamma\right)}{\partial\boldsymbol{Q}_I} \ ,$$

where $\boldsymbol{F}_I^C$ is the electrostatic contribution to the force.

We proceed by defining basis functions (see appendix A were linear and cubic spline basis functions are defined) so that the $U_{\zeta i}(x)$ potential is expressed as a linear combination of them,

$$U_{\zeta i}(\chi) = \sum_d \phi_{\zeta id} u_{\zeta id}(\chi) \ , \tag{2.13}$$

where $u_{\zeta id}$ is the basis function and $\phi_{\zeta id}$ is the coefficient. We also derive the set of basis functions for the force,

$$f_{\zeta id}(x) = -\frac{du_{\zeta id}}{d\chi} \ . \tag{2.14}$$

So,

$$F_{\zeta i}(\chi) = \sum_d \phi_{\zeta id} f_{\zeta id}(\chi) \ , \tag{2.15}$$

and the CG force on particle I becomes

$$\boldsymbol{F}_I\left(\boldsymbol{Q}^M\right) = \boldsymbol{F}_I^C\left(\boldsymbol{Q}^M\right) + \sum_{\zeta id} \phi_{\zeta id}\boldsymbol{\mathcal{G}}_{I;\zeta id}\left(\boldsymbol{Q}^M\right) \ ,$$

where

$$\boldsymbol{\mathcal{G}}_{I;\zeta id}\left(\boldsymbol{Q}^M\right) = \sum_\gamma f_{\zeta id}\left(\chi_\zeta\left(\left\{\boldsymbol{Q}\right\}_\gamma\right)\right)\frac{\partial\chi_\zeta\left(\left\{\boldsymbol{Q}\right\}_\gamma\right)}{\partial\boldsymbol{Q}_I} \ .$$

To simplify this expression we replace the multiple index ζid with a single index D running over all basis functions for all interaction types. We denote the total number of basis functions with N_D. Then,

$$\boldsymbol{F}_I\left(\boldsymbol{Q}^M\right) = \boldsymbol{F}_I^C\left(\boldsymbol{Q}^M\right) + \sum_{D=1}^{N_D} \phi_D\boldsymbol{\mathcal{G}}_{I;D}\left(\boldsymbol{Q}^M\right) \ .$$

We define $\boldsymbol{f}_I(\boldsymbol{q}^N)$ to be the sum of the atomistic forces acting on the CG site I. Recall we denote with $\boldsymbol{M}(\boldsymbol{q}^N)$ the operator that maps the atomistic coordinates to CG

coordinates. We denote with $\boldsymbol{\phi}$ the set of CG force field parameters, $\boldsymbol{\phi} = \{\phi_1, ..., \phi_D\}$. We sample the molecular system by atomistic (and not CG) MD simulations. We denote with n_t the number of MD sample configurations. We seek to minimize the total error of the CG force field approximation.

$$\chi^2 = \frac{1}{3M} \left\langle \sum_{I=1}^{M} \left| \boldsymbol{f}_I(\boldsymbol{q}^N) - \boldsymbol{F}_I(\boldsymbol{M}(\boldsymbol{q}^N); \boldsymbol{\phi}) \right|^2 \right\rangle_t \tag{2.16}$$

$$= \frac{1}{3Mn_t} \sum_{t=1}^{n_t} \sum_{I=1}^{M} \left| \tilde{\boldsymbol{f}}_I(\boldsymbol{q}_t^N) - \sum_{D=1}^{N_D} \phi_D \boldsymbol{\mathcal{G}}_{I;D}(\boldsymbol{M}(\boldsymbol{q}_t^N)) \right|^2 , \tag{2.17}$$

where,

$$\tilde{\boldsymbol{f}}_I(\boldsymbol{q}_t^N) = \boldsymbol{f}_I(\boldsymbol{q}_t^N) - \boldsymbol{F}_I^C(\boldsymbol{M}(\boldsymbol{q}_t^N)) \ .$$

So we first run an atomistic MD simulation getting n_t sample configurations. Then we map the atomistic configurations to CG configurations. We calculate the atomistic forces and the CG forces acting on CG particles and seek to minimize their difference. Equation 2.17 shows that when we use functions linearly dependent on the parameters to define the CG potential, e.g. linear splines or the Lennard Jones potential, then we arrive to a linear least squares minimization problem [1].

We want to reformulate equation 2.17 in matrix form. We define $\underline{\tilde{\boldsymbol{f}}}$, $\underline{\boldsymbol{\mathcal{G}}}$ and ϕ as

$$\underline{\tilde{\boldsymbol{f}}} = \begin{bmatrix} \tilde{f}_{1x}(\boldsymbol{q}_1^N) \\ \tilde{f}_{1y}(\boldsymbol{q}_1^N) \\ \tilde{f}_{1z}(\boldsymbol{q}_1^N) \\ \vdots \\ \tilde{f}_{Mx}(\boldsymbol{q}_1^N) \\ \tilde{f}_{My}(\boldsymbol{q}_1^N) \\ \tilde{f}_{Mz}(\boldsymbol{q}_1^N) \\ \tilde{f}_{1x}(\boldsymbol{q}_2^N) \\ \vdots \\ \tilde{f}_{Mx}(\boldsymbol{q}_{n_t}^N) \\ \tilde{f}_{My}(\boldsymbol{q}_{n_t}^N) \\ \tilde{f}_{Mz}(\boldsymbol{q}_{n_t}^N) \end{bmatrix} \tag{2.18}$$

[1] We could have not used linear basis functions and still define a valid non linear minization scheme with equation 2.16. More on that later.

$$\underline{\underline{\boldsymbol{\mathcal{G}}}} = \begin{bmatrix} \mathcal{G}_{1x;1}(\boldsymbol{M}(\boldsymbol{q}_1^N)) & \cdots & \mathcal{G}_{1x;D}(\boldsymbol{M}(\boldsymbol{q}_1^N)) \\ \mathcal{G}_{1y;1}(\boldsymbol{M}(\boldsymbol{q}_1^N)) & \cdots & \mathcal{G}_{1y;D}(\boldsymbol{M}(\boldsymbol{q}_1^N)) \\ \mathcal{G}_{1z;1}(\boldsymbol{M}(\boldsymbol{q}_1^N)) & \cdots & \mathcal{G}_{1z;D}(\boldsymbol{M}(\boldsymbol{q}_1^N)) \\ \mathcal{G}_{2x;1}(\boldsymbol{M}(\boldsymbol{q}_1^N)) & \cdots & \mathcal{G}_{2x;D}(\boldsymbol{M}(\boldsymbol{q}_1^N)) \\ \vdots & \vdots & \vdots \\ \mathcal{G}_{Mx;1}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) & \cdots & \mathcal{G}_{Mx;D}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) \\ \mathcal{G}_{My;1}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) & \cdots & \mathcal{G}_{My;D}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) \\ \mathcal{G}_{Mz;1}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) & \cdots & \mathcal{G}_{Mz;D}(\boldsymbol{M}(\boldsymbol{q}_{n_t}^N)) \end{bmatrix} \tag{2.19}$$

$$\underline{\phi} = \begin{bmatrix} \phi_1 \\ \vdots \\ \phi_D \end{bmatrix} .$$

Then

$$\chi^2 = \frac{1}{3n_t M} \left| \underline{\tilde{\boldsymbol{f}}} - \underline{\underline{\boldsymbol{\mathcal{G}}}}\,\underline{\phi} \right|^2 = \frac{1}{3n_t M} \left(\underline{\tilde{\boldsymbol{f}}} - \underline{\underline{\boldsymbol{\mathcal{G}}}}\,\underline{\phi} \right)^T \left(\underline{\tilde{\boldsymbol{f}}} - \underline{\underline{\boldsymbol{\mathcal{G}}}}\,\underline{\phi} \right) \tag{2.20}$$

If the matrix $\underline{\underline{\boldsymbol{\mathcal{G}}}}$ is full rank then there is a unique $\underline{\phi}$ that minimizes χ^2. The matrix $\underline{\underline{\boldsymbol{\mathcal{G}}}}$ is not full rank when the sampling is not adequate for some distances or angles. Then some columns will be zero and some ϕ_i parameters cannot be calculated. While choosing the basis functions care must be taken so that there are not zero or undersampled columns of $\underline{\underline{\boldsymbol{\mathcal{G}}}}$.

The minimization of equation 2.20 can be performed by QR decomposition or SVD decomposition or even by some iterative technique. An obstacle might be the size of $\underline{\underline{\boldsymbol{\mathcal{G}}}}$ when large systems with long trajectories are handled with great detail. In such cases the amount of computer memory might not be enough. A solution is to group the rows of the matrix in many smaller sets of rows and calculate $\underline{\phi}$ for each one of them, taking the average to be the overall $\underline{\phi}$. This process is called the block averaging approximation [24].

Another approach for large systems is to use the normal equation, i.e. define

$$\underline{b} = \underline{\underline{\boldsymbol{\mathcal{G}}}}\,\underline{\tilde{\boldsymbol{f}}} \ , \tag{2.21}$$

$$\underline{\underline{G}} = \underline{\underline{\boldsymbol{\mathcal{G}}}}^T \underline{\underline{\boldsymbol{\mathcal{G}}}} \ , \tag{2.22}$$

then equation 2.20 becomes

$$\chi^2 = \underline{\phi}^T \underline{\underline{G}}\,\underline{\phi} - 2\underline{b}^T \underline{\phi} + \underline{\tilde{\boldsymbol{f}}}^T \underline{\tilde{\boldsymbol{f}}} \tag{2.23}$$

$\underline{\underline{G}}$ is symmetric and normal, with N_D rows and columns. The elements of $\underline{b}$ and $\underline{\underline{G}}$ are given by

$$\underline{b}_D = \frac{1}{3M} \left\langle \sum_{I=1}^{M} \boldsymbol{\mathcal{G}}_{I;D}(\boldsymbol{M}(\boldsymbol{q}_t^N)) \cdot \left[\boldsymbol{f}_I(\boldsymbol{q}_t^N) - \boldsymbol{F}_I^C(\boldsymbol{M}(\boldsymbol{q}_t^N)) \right] \right\rangle_t \tag{2.24}$$

$$\underline{\underline{G}}_{DD'} = \frac{1}{3M} \left\langle \sum_{I=1}^{M} \boldsymbol{\mathcal{G}}_{I;D}(\boldsymbol{M}(\boldsymbol{q}_t^N)) \cdot \boldsymbol{\mathcal{G}}_{I;D'}(\boldsymbol{M}(\boldsymbol{q}_t^N)) \right\rangle_t \tag{2.25}$$

In this way one can minimize 2.23 by iterative methods or by finding the stationary point where $\partial\chi^2/\partial\phi_D = 0$ for all D. Finding the stationary point is equivalent to solving

$$\underline{\underline{G}}\underline{\phi} = \underline{b} \ , \tag{2.26}$$

which can be done by Gaussian elimination or by LU decomposition. Using equations 2.24 and 2.25 dramatically reduces the computer memory requirements. Still the condition number of $\underline{\underline{G}}$ is the square of the condition number of $\underline{\boldsymbol{\mathcal{G}}}$, so numerical methods using $\underline{\underline{G}}$ might be less accurate. Preconditioning might address this problem.

2.7 Relative Entropy

2.7.1 Relative Entropy

The relative entropy (or the Kullback-Leibler divergence)[28, 6, 12, 19] is a measure of distance between two probability distributions. The aforementioned quantity is defined as

$$S_{rel}(\hat{U}) = \left\langle ln \frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \right\rangle_{\boldsymbol{Q}} = \int d\boldsymbol{Q} \ P(\boldsymbol{Q}) ln \frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \tag{2.27}$$

Where $P(\boldsymbol{Q})$ is the CG probability distribution that stems from the atomistic one (equation 2.6).

$$P(\boldsymbol{Q}) = \int d\boldsymbol{q} p(\boldsymbol{q}) \delta \left(\boldsymbol{M}(\boldsymbol{q}) - \boldsymbol{Q} \right) \ ,$$

$\hat{U}(\boldsymbol{Q})$ is the trial CG potential, and $P_{\hat{U}}(\boldsymbol{Q})$ is the probability distribution that stems from the trial CG potential (equation 2.4) ,

$$P_{\hat{U}}(\boldsymbol{Q}) = \frac{1}{\hat{Z}} e^{-\hat{U}(\boldsymbol{Q})/k_B T} \ ,$$

where

$$\hat{Z} = \int d\boldsymbol{Q} e^{-\hat{U}(\boldsymbol{Q})/k_B T} \ .$$

The expectation $\langle \cdot \rangle_{\boldsymbol{Q}}$ is calculated with respect to the probability $P(\boldsymbol{Q})$. It can be converted (see appendix B) to be written as an average with respect to atomistic probability distribution (equation 2.2)

$$\begin{aligned} S_{rel}(\hat{U}) &= \left\langle ln \frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} \right\rangle_{\boldsymbol{q}} + S_{map}(\boldsymbol{M}) \\ &= \int d\boldsymbol{q} \ p(\boldsymbol{q}) ln \frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} + S_{map}(\boldsymbol{M}) \ , \end{aligned}$$

where $\boldsymbol{M}$ is the coarse graining map (equation 2.1) and S_{map} is the relative entropy that stems from the degeneracy of states of the CG mapping and it is independent from the trial CG potential $\hat{U}$.

In order to best match the atomistic and the CG probability distributions we minimize the relative entropy. Since S_{map} is independent of the trial CG potential we only need to minimize

$$\mathcal{F}(\hat{U}) = \left\langle ln \frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} \right\rangle_{\boldsymbol{q}} \ . \tag{2.28}$$

In principle one could minimize $S_{rel}(\hat{U})$ that includes $S_{map}(\boldsymbol{M})$ in order to choose the best among different CG mappings $\boldsymbol{M}$. We will not examine this case.

Typically we would parametrize the trial CG potential with a set of numbers

$$\boldsymbol{\phi} = \{\phi_1, ..., \phi_m\} \ .$$

Note that these parameters are equivalent to the $\boldsymbol{\phi}$ parameters we used to parametrize the FM potential (see relation 2.13).

We seek to find the optimal $\boldsymbol{\phi}^*$

$$\boldsymbol{\phi}^* = \underset{\boldsymbol{\phi}}{argmin} \ \ \mathcal{F}(\hat{U}(\cdot; \boldsymbol{\phi})) \tag{2.29}$$

To proceed with some numerical minimization scheme, such as the Newton-Raphson scheme, we need to calculate the Jacobian and the Hessian of $\mathcal{F}$ with respect to $\boldsymbol{\phi}$. First we work a bit with $\mathcal{F}(\hat{U})$.

$$\mathcal{F}(\hat{U}) = \int d\boldsymbol{q} \ p(\boldsymbol{q}) \ ln \frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} = \int d\boldsymbol{q} \ p(\boldsymbol{q}) \ ln \frac{Z^{-1} e^{-U(\boldsymbol{q})/k_B T}}{\hat{Z}^{-1} e^{-\hat{U}(\boldsymbol{M}(\boldsymbol{q}))/k_B T}}$$

$$
\begin{aligned}
&= \int d\boldsymbol{q}\ \ p(\boldsymbol{q}) ln \frac{\hat{Z}}{Z} + \int d\boldsymbol{q}\ \ p(\boldsymbol{q})\ \ ln\ \ e^{\left(\hat{U}(\boldsymbol{M}(\boldsymbol{q}))-U(\boldsymbol{q})\right)/k_BT} \\
&= ln\ \ \hat{Z} - ln\ \ Z + \frac{1}{k_BT} \int d\boldsymbol{q}\ \ p(\boldsymbol{q}) \left(\hat{U}(\boldsymbol{M}(\boldsymbol{q})) - U(\boldsymbol{q})\right) \\
&= \frac{1}{k_BT} \left\langle \hat{U}(\boldsymbol{M}(\boldsymbol{q})) - U(\boldsymbol{q}) \right\rangle_{\boldsymbol{q}} + ln\ \ \hat{Z} - ln\ \ Z
\end{aligned}
$$

The Jacobian then is $J(\boldsymbol{\phi}) = \nabla_{\boldsymbol{\phi}} \mathcal{F} = (J_1(\boldsymbol{\phi}), ..., J_m(\boldsymbol{\phi}))$, where $J_i(\boldsymbol{\phi})$ is,

$$
\begin{aligned}
J_i(\boldsymbol{\phi}) = \frac{\partial \mathcal{F}}{\partial \phi_i} &= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} + \frac{\partial}{\partial \phi_i} ln\ \ \hat{Z} \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} + \frac{1}{\hat{Z}} \frac{\partial \hat{Z}}{\partial \phi_i} \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{r}} + \frac{1}{\hat{Z}} \frac{\partial}{\partial \phi_i} \int d\boldsymbol{Q}\ \ e^{-\hat{U}(\boldsymbol{Q};\boldsymbol{\phi})/K_bT} \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} + \frac{1}{\hat{Z}} \int d\boldsymbol{Q}\ \ \frac{\partial}{\partial \phi_i} e^{-\hat{U}(\boldsymbol{Q};\boldsymbol{\phi})/k_BT} \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} + \frac{1}{\hat{Z}} \int d\boldsymbol{Q}\ \ e^{-\hat{U}(\boldsymbol{Q};\boldsymbol{\phi})/k_BT} \left(\frac{-1}{k_BT}\right) \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} - \frac{1}{k_BT} \int d\boldsymbol{Q}\ \ \frac{1}{\hat{Z}} e^{-\hat{U}(\boldsymbol{Q};\boldsymbol{\phi})/k_BT} \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) \\
&= \frac{1}{k_BT} \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} - \frac{1}{k_BT} \int d\boldsymbol{Q}\ \ P_{\hat{U}}(\boldsymbol{Q}|\boldsymbol{\phi}) \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) \\
&= \frac{1}{k_BT} \left(\left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi}) \right\rangle_{\boldsymbol{q}} - \left\langle \frac{\partial}{\partial \phi_i} \hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \right)
\end{aligned}
$$

Where the first expectation $\langle \cdot \rangle_{\boldsymbol{q}}$ is with respect to the atomistic probability distribution $p(\boldsymbol{q})$ and the second $\langle \cdot \rangle_{\boldsymbol{Q}|\boldsymbol{\phi}}$ is with respect to the trial CG probability distribution ($P_{\hat{U}}(\boldsymbol{Q}|\boldsymbol{\phi}) = \hat{Z}^{-1} exp(-\hat{U}(\boldsymbol{Q};\boldsymbol{\phi})/k_BT)$).

Similarly the Hessian has elements

$$
\begin{aligned}
H_{ij} = \frac{\partial^2 \mathcal{F}(\hat{U})}{\partial \phi_i \partial \phi_j} &= \frac{1}{k_BT} \left(\left\langle \frac{\partial^2 \hat{U}(\boldsymbol{M}(\boldsymbol{q}), \boldsymbol{\phi})}{\partial \phi_i \partial \phi_j} \right\rangle_{\boldsymbol{q}} - \left\langle \frac{\partial^2 \hat{U}(\boldsymbol{Q}, \boldsymbol{\phi})}{\partial \phi_i \partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \right) \\
&+ \left(\frac{1}{k_BT}\right)^2 \left(\left\langle \frac{\partial \hat{U}(\boldsymbol{Q}, \boldsymbol{\phi})}{\partial \phi_i} \frac{\partial \hat{U}(\boldsymbol{Q}, \boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} - \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_i} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \right)
\end{aligned}
$$

The second part of the Hessian is the covariance matrix

$$C_{ij} = Cov_{\boldsymbol{Q}|\boldsymbol{\phi}} \left[\frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_i}, \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_j} \right]$$
$$= \left(\left\langle \frac{\partial \hat{U}(\boldsymbol{Q},\boldsymbol{\phi})}{\partial \phi_i} \frac{\partial \hat{U}(\boldsymbol{Q},\boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} - \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_i} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \right)$$

In this work we examine CG systems that have only pairwise interactions. We parametrize the trial CG potential as a linear combination of basis functions $U_{\zeta i}$,

$$\hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) = \sum_{I \neq J} \sum_{\zeta i} U_{\zeta i}(Q_{IJ}) \ , \tag{2.30}$$

where the outer sum is over all pairs of atoms, Q_{IJ} is the distance of the pair I, J and $U_{\zeta i}$ is exactly the same as in the FM method (equation 2.13), i.e.,

$$U_{\zeta i}(Q_{IJ}) = \sum_{d} \phi_{\zeta id} u_{\zeta id}(Q_{IJ}) \ .$$

The CG potential then is

$$\hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) = \sum_{I \neq J} \sum_{\zeta id} \phi_{\zeta id} u_{\zeta id}(Q_{IJ}) \ ,$$

and if we choose to replace the ζid multiple index with a single index $K = 1, ..., N_D$ then,

$$\hat{U}(\boldsymbol{Q};\boldsymbol{\phi}) = \sum_{K=1}^{N_D} \phi_K \sum_{I \neq J} u_K(Q_{IJ}) = \sum_{K=1}^{N_D} \phi_K \bar{u}_K(\boldsymbol{Q}) \ ,$$

where

$$\bar{u}_K(\boldsymbol{Q}) = \sum_{I \neq J} u_K(Q_{IJ}) \ .$$

The form of the basis functions u_K is similar to the one given in appendix A. The slight difference is that now we define the linear and cubic spline basis functions for the potential instead of the force.

The first part of the Hessian vanishes due to the double differentiation and the Hessian becomes

$$H_{ij}(\boldsymbol{\phi}) = \left(\frac{1}{k_B T} \right)^2 C_{ij}$$

$$= \left(\frac{1}{k_B T}\right)^2 \left(\left\langle \frac{\partial \hat{U}(\boldsymbol{Q},\boldsymbol{\phi})}{\partial \phi_i} \frac{\partial \hat{U}(\boldsymbol{Q},\boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} - \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_i} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \left\langle \frac{\partial \hat{U}(\boldsymbol{Q};\boldsymbol{\phi})}{\partial \phi_j} \right\rangle_{\boldsymbol{Q}|\boldsymbol{\phi}} \right)$$

The Hessian now is positive semidefinite and symmetric. So if $\phi \in \Phi$ and Φ is convex, then the minimization problem of equation 2.29 is convex as well. If the Hessian is positive definite the minimization problem is strictly convex and has a unique minimum.

The averages of the Jacobian and the Hessian are calculated by sampling either atomistic or CG MD simulations. We denote the number of atomistic samples with S_{at} and the number of CG samples with S_{CG}. Then the Jacobian elements become,

$$J_i(\boldsymbol{\phi}) = \frac{1}{k_B T} \left(\frac{1}{S_{at}} \sum_{s=1}^{S_{at}} \bar{u}_i(\boldsymbol{M}(\boldsymbol{q}_s)) - \frac{1}{S_{CG}} \sum_{s=1}^{S_{CG}} \bar{u}_i(\boldsymbol{Q}_s|\boldsymbol{\phi})) \right) \quad , \tag{2.31}$$

and the Hessian elements become,

$$H_{ij}(\boldsymbol{\phi}) = \left(\frac{1}{k_B T}\right)^2 \left(\frac{1}{S_{CG}} \sum_{s=1}^{S_{CG}} \bar{u}_i(\boldsymbol{Q}_s|\boldsymbol{\phi}) \bar{u}_j(\boldsymbol{Q}_s|\boldsymbol{\phi}) - \left(\frac{1}{S_{CG}} \sum_{s=1}^{S_{CG}} \bar{u}_i(\boldsymbol{Q}_s|\boldsymbol{\phi}) \right) \left(\frac{1}{S_{CG}} \sum_{s=1}^{S_{CG}} \bar{u}_j(\boldsymbol{Q}_s|\boldsymbol{\phi}) \right) \right) \quad .$$

Minimization of the relative entropy with some (iterative) numerical scheme starts with an initial $\boldsymbol{\phi}^0$ and proceeds iteratively at better $\boldsymbol{\phi}^k$'s until convergence. In order to calculate the Jacobian we must once calculate the first term of equation 2.31, which is the contribution of the atomistic MD simulation to it. The second term of the Jacobian must be recalculated at each $\boldsymbol{\phi}^k$ by resampling a new CG MD simulation. The Hessian must be recalculated at each step like the Jacobian. It is important that the atomistic contribution to the Jacobian is adequately sampled, i.e. S_{at} is large enough, since it produces a constant bias to the total Jacobian.

2.7.2 Modified Robins Monro minimization

We want to solve the minimization problem defined at 2.29. Since we have the Jacobian and the Hessian of $\mathcal{F}$ (equation 2.28) we could proceed with the Newton Raphson procedure to find where the Jacobian vanishes. Specifically we could start at some $\boldsymbol{\phi}_0$ and proceed as below until convergence,

$$\boldsymbol{\phi}_{k+1} = \boldsymbol{\phi}_k - \boldsymbol{H}^{-1}(\boldsymbol{\phi}_k)\boldsymbol{J}(\boldsymbol{\phi}_k) \quad , \quad k = 0, 1, ...$$

The problem is that the Jabobian and the Hessian are very noisy, i.e. it is difficult to acquire accurate expectations. Due to the noise the Hessian can become singular and

non invertible. If we want to overcome this we must use a very large number of samples (S_{CG}). This is not efficient and it poses a severe problem for medium to large molecular systems. An alternative is to use the Robins-Monro [27] stochastic optimization method.

$$\boldsymbol{\phi}_{k+1} = \boldsymbol{\phi}_k - \alpha_k \boldsymbol{J}(\boldsymbol{\phi}_k) \quad , \quad k = 1, 2, ...$$

where α_k is a sequence of real numbers such that $\sum_{k=1}^{\infty} \alpha_k = +\infty$ and $\sum_{k=1}^{\infty} \alpha_k^2 < +\infty$. Intuitively these conditions for α_k mean that the sequence of steps may span as large distance as needed to find the root, while maintaining a convergent behavior for large k. A typical such sequence is $\alpha_k = 1/k$.

Still another problem persists. The Robins Monro scheme, as opposed to the Newton-Raphson scheme, does not use curvature information to determine an iteration's step size. This causes problems as the initial step size might be too large. The sequence α_k could be modified to reduce the step size. Even then the Robins Monro step size could be inadequately small at flat regions.

The modified Robins-Monro scheme [6] addresses these problems. The problem of the Newton-Raphson scheme occurs when solving $\boldsymbol{H}(\boldsymbol{\phi}_k)\boldsymbol{x} = \boldsymbol{J}(\boldsymbol{\phi}_k)$, as $\boldsymbol{H}$ might be singular. If instead of solving this system we perform a few conjugate gradient steps, we get some curvature information without having to deal with the singularity. The number of conjugate gradient steps should not be too large, since that would be equivalent to solving the initial problem. The optimal number of steps remains an open problem, but the algorithm is stable for a wide range of selections. The number of steps we empirically used in this work is half the dimension of $\boldsymbol{x}$.

To sum up, the modified Robins-Monro scheme consists of starting from zero $\boldsymbol{p}_k$ and performing several conjugate gradient steps to determine $\boldsymbol{p}_k$ from,

$$\boldsymbol{H}(\boldsymbol{\phi}_k)\boldsymbol{p}_k = \boldsymbol{J}(\boldsymbol{\phi}_k) \ ,$$

and then performing the Robins-Monro step with $\boldsymbol{p}_k$ instead of the Jacobian.

$$\boldsymbol{\phi}_{k+1} = \boldsymbol{\phi}_k - \alpha_k \boldsymbol{p}_k \tag{2.32}$$

This method is a hybrid of the Newton-Raphson and Robins-Monro schemes and apart from solving the instability problems that the plain Robins-Monro scheme does, it is also much more efficient.

2.8 Relation of the three methods

The Relative Entropy and Force Matching methods theoretically [7, 23, 18, 22] produce the exact PMF (equation 2.4), provided that the trial CG potential spans all possible interactions. When the PMF is approximated with a pair PMF these methods are not

expected to produce the same results. The IBI method should produce the same results with the RE method when using a pair PMF, since they both match the atomistic pair correlation functions [7]. In practice, as we shall later see, the three methods produce different results for all but the simplest cases. This is primarily because the pair PMF's from the FM and RE methods are distinct projections of the many body PMF [18]. Also the numerical properties of the three algorithms are different.

- The IBI method
  - Seeks to minimize the distance of the atomistic and CG RDF's (equation 2.8).
  - Samples the atomistic RDF from an initial MD simulation.
  - Starts with the DBI potential.
  - Iteratively resamples the CG RDF by MD simulation of the CG system, making corrections to the trial potential.
- The FM method
  - Seeks to minimize the average atomistic and CG force differences (equation 2.16).
  - Samples once the atomistic forces and trajectory, from an initial MD simulation.
  - Linearly parametrizes the trial CG potential.
  - Solves the linear least squares problem of minimizing the mean atomistic and CG force differences.
- The RE method
  - Seeks to minimize the distance of the atomistic and the CG positional probability distributions (equation 2.29).
  - Samples once the atomistic Jacobian, from an initial MD simulation.
  - Starts with the DBI potential.
  - Iteratively resamples the CG Jacobian and Hessian by MD simulation of the CG system, making corrections to the trial potential.
  - Proceeds with the stochastic modified Robins-Monro scheme.

Chapter 3

Implementation Details

3.1 Iterative Boltzmann Inversion

The IBI method was implemented in Python, while the required MD runs were performed by a custom MPI + openmp parallel C++ MD code. Excerpts of code of the IBI method are given in C.1. The RDF calculations where performed by a custom SIMD + openmp parallel C++ code. The calls from Python to the external programs was made using Python's *subprocess* package.

For all studied systems we first obtained a single atomistic MD trajectory from which we calculated the RDF (section 2.4.1). The calculated RDF was first processed with a narrow Savitzky-Golay filter to make it smoother. Then the DBI potential (section 2.4.2) was calculated from it. RDF's values below a tolerance of 10^{-3} were zeroed out to avoid irregular bumps of the DBI. The DBI potential at high energies (short distances) was extrapolated by fitting $a\ exp(-br) + c$ at the first $0.25 Angstroms$.

The potential $U(r)$ was intentionally sampled at relatively few distances r (≈ 60 knots). The values between these nodal points were given by a *InterpolatedUnivariateSpline* from the *scipy.interpolate* package. The corrections to the potential at each step of the procedure were made only at these nodal points. This has proved to provide stability to the iterations.

Each iteration involves an MD run of 0.5 to $2ns$ and samples about 1000 configurations in order to calculate the current RDF. The corrections to the potential's nodal points are made (equation 2.7) and the new MD potential is calculated. The potential is again extrapolated as before at the high energy distances. The iterations stop when the maximum correction to the potential is below $5 \cdot 10^{-3} Kcal/mol$.

3.2 Force Matching

The Force Matching method was implemented in Matlab. Excerpts of code of the FM method are given in C.2. For all studied systems we first obtained a single atomistic MD trajectory, the same atomistic trajectory used in IBI, from which we calculated the mapped coordinates $\boldsymbol{M}(\boldsymbol{r}_t^n)$ and the CG forces $\tilde{\boldsymbol{f}}_I(\boldsymbol{r}_t^n)$ (see eq. 2.17).

We created a *Basis* class that is extended by *LinearBasis*, *CubicSplineBasis* and *LJBasis* to implement specific basis functions. The different basis classes define $f_{\zeta id}(x)$ and $u_{\zeta id}(x)$ (equations 2.14 and 2.13), where $u_{\zeta id}(x)$ is defined to be the integral of $f_{\zeta id}(x)$. The *Basis* class can calculate $F_{\zeta i}(x)$ from a set of $\phi_{\zeta id}$'s (equation 2.15). We created basis functions for non-bonded potentials only, as such were the cases we studied. The x range of the basis functions was evenly spaced in all cases. The smallest x was determined by examining where the RDF had it's first nonzero values, in order to avoid singularities.

Having defined the basis functions we calculated $\underline{\underline{\boldsymbol{\mathcal{G}}}}$ (equation 2.19) in parallel using Matlab's *parfeval*. The calculation of $\underline{\tilde{\boldsymbol{f}}}$ (equation C.2) was trivial as we did not have Coulombic interactions. We calculated $\underline{\underline{G}}$ and $\underline{b}$ as in equations 2.21 and 2.22 and then solved the normal equations problem of equation 2.26 in order to calculate $\underline{\phi}$. We chose not to use equations 2.24 and 2.25, as we wanted to compare the numerical accuracy of minimizing using the form 2.20 by a singular value decomposition (SVD), in relation to minimize by solving the normal equations 2.26. Where we made such tests we noticed no benefit of the SVD approach.

For the non linear Morse potential case, we directly calculated (again in parallel using *parfeval*) $\tilde{\boldsymbol{f}}_I(\boldsymbol{q}_t^N) - \sum_{D=1}^{N_D} \phi_D \boldsymbol{\mathcal{G}}_{I;D}(\boldsymbol{M}(\boldsymbol{q}_t^N))$ from equation 2.16 and fed it to Matlab's *lsqnonlin* in order to determine the Morse CG potential parameters.

The CG potentials at high energies (low distances) were again extrapolated by fitting $a\ exp(-br) + c$ at the first $0.25 Angstroms$

3.3 Relative Entropy

The relative entropy method was implemented in Python, while the required MD runs where performed initially by *Gromacs* [2] and later by *LAMMPS* [26]. Excerpts of code of the RE method are given in C.3. The calculations of the Jacobian and the Hessian were first implemented in Mathlab and later in MPI parallel C++ in favor of speed (30x to 70x speedup on a typical i7), using *Eigen* [11] for then matrix/vector operations. The calls from Python to the external programs was made using Python's *subprocess* package.

The iterative processes that we primarily performed is the Modified Robins-Monro 2.7.2. The conjugate gradients steps performed were half of the dimension of the linear basis. The α_k sequence of equation 2.32 was $1/k$. For the water system we used the Newton-Raphson iterative process.

We started the iterative process from the DBI potential (section 2.4.2). The calculated RDF was first processed with a narrow Savitzky-Golay filter to make it smoother. Then the DBI potential was calculated from it. The potential at high energies (low distances) was extrapolated by a power law of the form $a/r^{12} + b$, requiring that the potential and its derivative are continuous at the beginning of the extrapolation.

We expressed the potential on a linear basis like the one used in the Force Matching procedure. The difference is that the basis is now linear to the potential and not to the force. The initial coefficients relative to the linear basis where calculated by a non linear least squares fit to the DBI potential.

At each step of the procedure we need to calculate the force that originates from the potential. Since we used linear basis functions for the potential, the force would not be smooth. To avoid that we smoothed the potential at each step with a narrow Savitzky-Golay filter, before performing the MD run.

The iterative process stopped after a fixed number of 100 to 150 iterations. That is because after a few tenths of iterations the Jacobian norm would no longer decrease. We wanted to monitor this process so we kept the number of steps to a safe high value. More details on this issue can be found at the results section.

Chapter 4

Results

4.1 Molecular Systems

We experimented with several molecular systems and compared the results of the three CG methods, Iterative Boltzmann Inversion, Force Matching and Relative Entropy. The molecular systems we experimented with is a two methane (CH_4) system, bulk methane at temperatures of 80K and 100K, and water (H_2O) at temperature of 300K.

4.1.1 Two Methane System

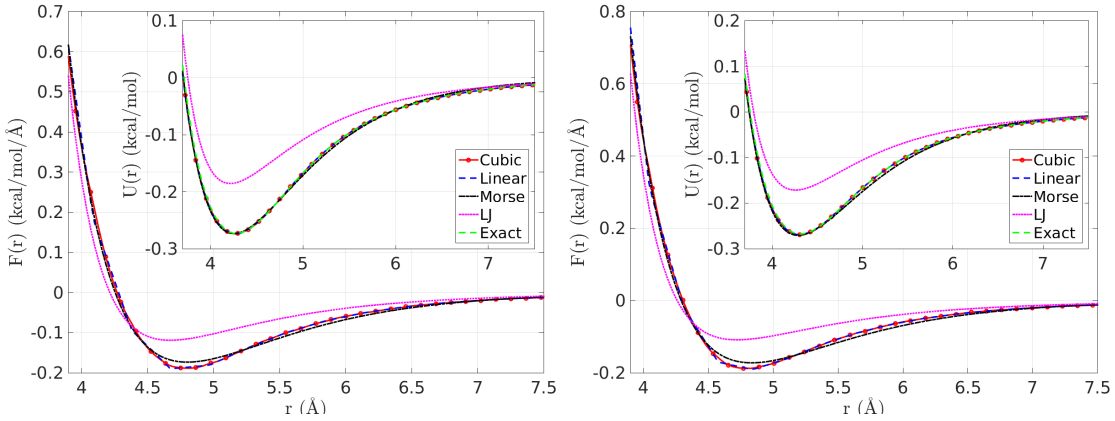
The first system we worked with is a model system of two methane (CH_4) molecules. We obtained two Langevin MD trajectories of the two methane system at temperatures of 80K and 100K. Langevin integration was used in order to avoid entrapment at minimal energy configurations of the two methane systems. We performed the Force Matching procedure using these trajectories. The data for these trajectories were obtained from [3].

We compared the results with an "exact" potential calculated by a geometric direct calculation [3]. The geometric averaged constrained two-body effective potential, is obtained by rotating the two CH_4 molecules around their centers of mass, through their Eulerian angles and taking account of all the possible orientations. The molecules are treated as rigid bodies, that is bond lengths and bond angles are kept fixed. This calculation produces the exact PMF which in this case is a two body PMF. All of the above calculations have been performed using the all-atom Dreiding force field.

The force matching procedure was performed for a variety of basis functions. With the linear basis function approach we used linear splines, cubic splines and Lennard

Jones basis functions. With the non linear residual minimization approach we used the Morse potential ($U(r) = D_e(1 - e^{-a(r-r_e)})^2$).

We present the results for the two methane system at figures 4.1a and 4.1b. The results for the linear basis and for the cubic basis are identical and they match the exact PMF. The results of the non linear Morse approach are almost identical to the above. The Lennard Jones basis though fails to capture the exact PMF. We conclude that for this simple system the FM approach indeed produces the exact PMF. We also note, observing the Lennard Jones case, that in general, care should be taken so that the trial CG potential is flexible enough to capture the PMF.



(a) 80K (b) 100K

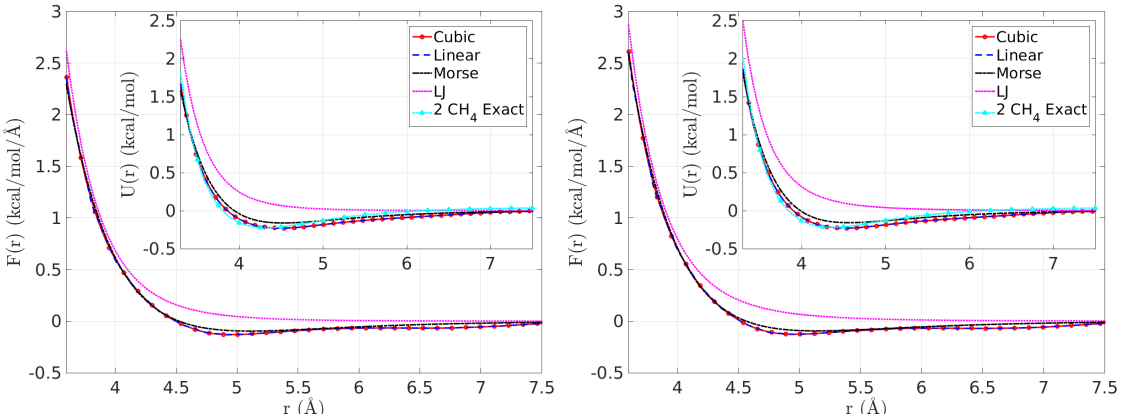
Figure 4.1: Force Matcing results for the two methane system.

4.1.2 Bulk Methane

The second system we worked with is bulk methane at two different temperatures, 80K and 100K. We obtained two atomistic MD trajectories of several nanoseconds (ns) at constant temperature (NVT conditions) at T=80K and T=100K. 512 CH_4 molecules were modeled, whereas the density was calculated after equilibrating the system in the NPT ensemble for 5 ns (ρ = 0.38 g/cm3). The time step was 0.5 fs and a cut-off distance of 10 Angstroms was used. The calculations have been performed using the all-atom Dreiding force field. For the coarse-grained representation of CH_4, we have used a one-site representation with a pair potential. We performed the various CG procedures using these trajectories.

We first tested the Force Matching method with various basis functions for the linear case and with the Morse potential for the non linear case. We see at figures 4.2a and 4.2b that the linear splines and the cubic splines approaches both yield the same results. The Morse approach results are also very close to them, but the Morse functional form

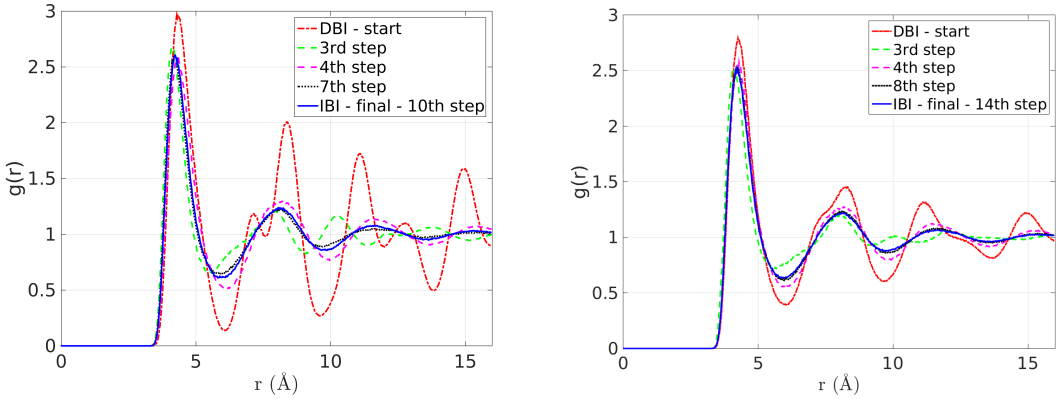
fails to capture the double minima of the CG force. The Lennard Jones basis completely fails to approximate the CG potential. We compare with the "exact" PMF of the two isolated methane system and observe that the bulk PMF is close to the two methane PMF.



(a) 80K (b) 100K

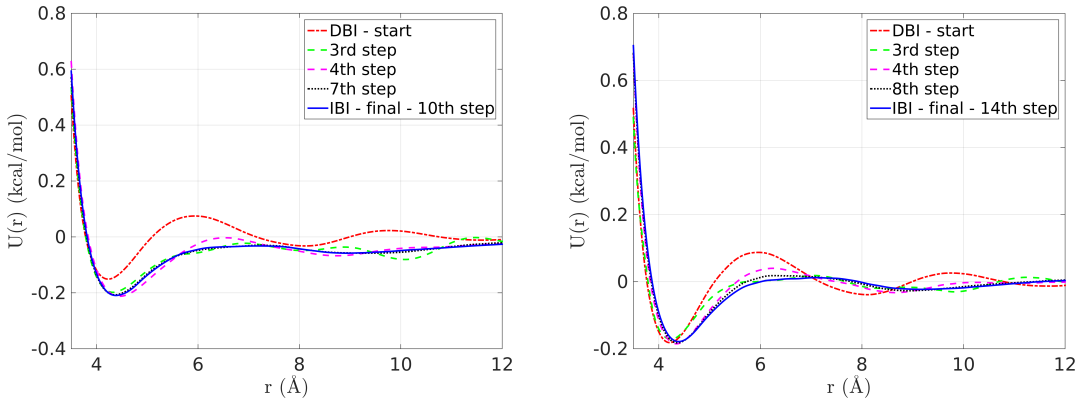
Figure 4.2: Force Matcing results for the bulk methane system.

We also tested the IBI method for the bulk methane at both temperatures. We see at figures 4.3a and 4.3b that relatively few iterations are needed for convergence, 10 for the system at 80K and 14 for the system at 100K. The evolution of the IBI potential is displayed at figures 4.4a and 4.4b.



(a) 80K (b) 100K

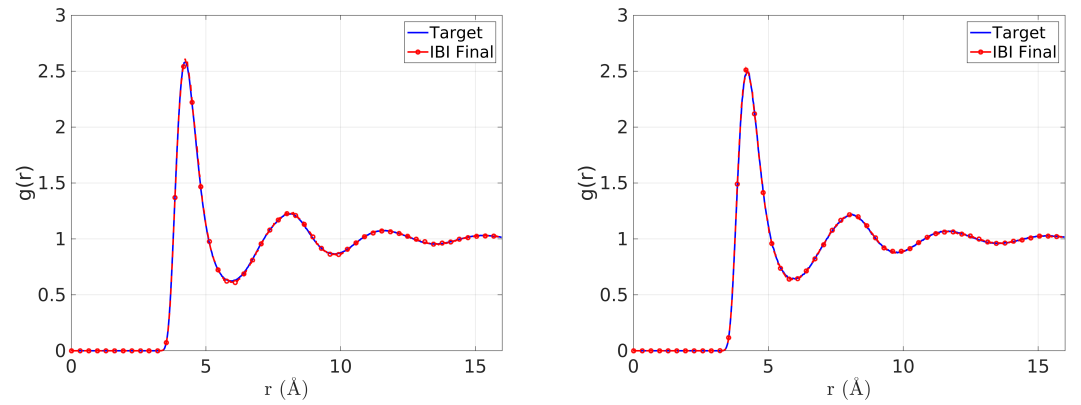
Figure 4.3: Evolution of the RDF during the IBI iterations for the bulk methane system.



(a) 80K (b) 100K

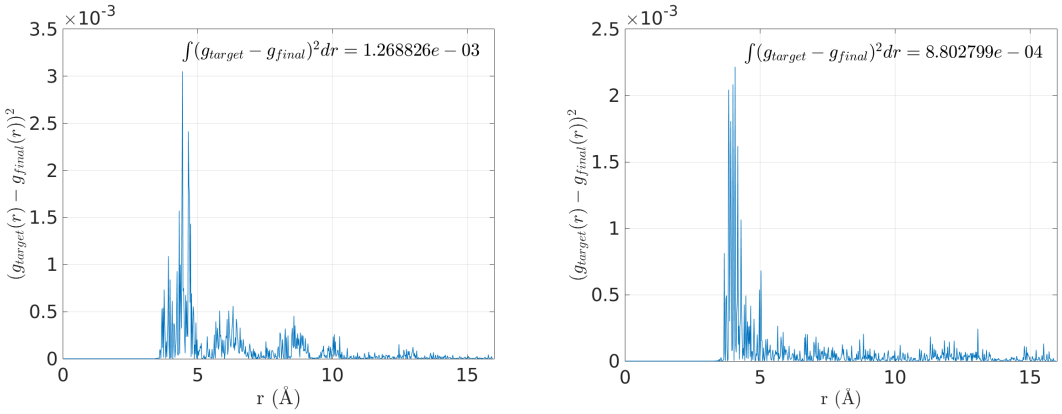
Figure 4.4: Evolution of the potential during the IBI iterations for the bulk methane system.

We present the target RDF along with the results of the IBI iterations, at figures 4.5a and 4.5b, so as to display their good matching. We also present at figures 4.6a and 4.6b the difference from the target RDF and calculate it's squared integral, in order to measure the quality of the match. We notice that the overall error is of the order of 10^{-3} and that most error comes from the RDF at high energies (low distances). This is the area where we have little data and have to rely on extrapolation of the potential.



(a) 80K (b) 100K

Figure 4.5: IBI RDF results for the bulk methane system.



(a) 80K (b) 100K

Figure 4.6: IBI RDF quality results for the bulk methane system.

We applied the Relative Entropy approach to the bulk methane at 100K temperature. The minimization was performed with the modified Robins-Monro scheme. A general observation we made is that the Jacobian norm does not reach zero, but it remains at a value close to zero, regardless of the number of iterations performed. This probably has to do with the quality of the sampling, i.e. the noise of the Jacobian and the Hessian. We performed two runs with a different MD sample size. We observe in figure 4.7 that by quadrupling the CG MD sample size, the minimum Jacobian norm halves. In any case the Jacobian does not improve after 40 iterations.

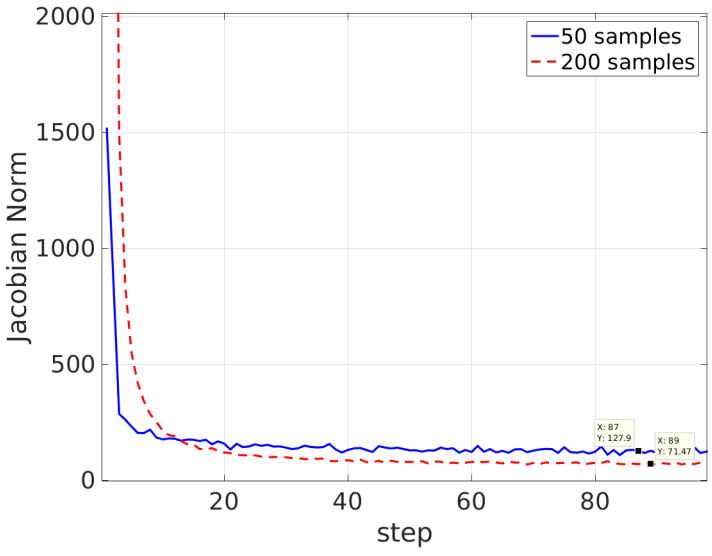
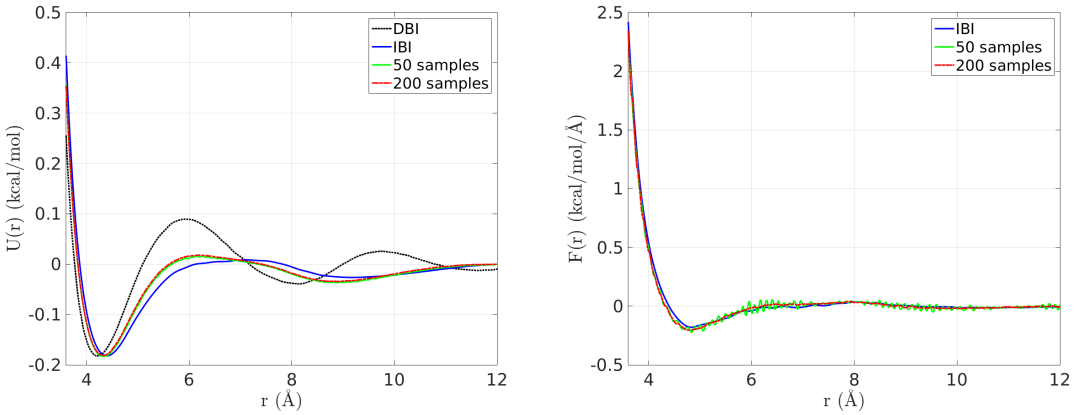


Figure 4.7: The effect of better sampling to the minimum of the Jacobian norm.

Still the results in both cases are very close to the IBI results, as they theoretically should. We see at figures 4.8a and 4.8b that the potentials and the forces from the IBI and the two RE calculations are nearly the same.

It is also evident from the forces plot that the potential derivative slightly oscillates. The width of the oscillations decreases for the "200 sample" case, secondarily because of the better sampling and primarily because there we used fewer basis functions, 80 linear splines instead of 140 linear splines used in the "50 sample" case. The oscillations could be treated by smoothing but it would probably be adequate and better, to use a cubic spline basis with wider distance intervals in addition to better MD sampling. Another direction of improvement in the Modified Robins-Monro RE scheme, would be to use an adaptive and increasing MD sample size as the Jacobian norm lowers. The final steps of the procedure could be replaced by Newton-Raphson steps if the sample size is adequate. We believe that these improvements would both lower the minimum Jacobian norm and treat any force oscillations.



(a) Potential (b) Force

Figure 4.8: Relative entropy results for the bulk methane system at 100K.

Concluding, for the bulk methane system at 100K, we compare the potentials and the forces from the three methods at figure 4.9 and find that the results are nearly identical from all methods. Specifically the results from IBI and RE are exactly the same, as they should. The results from FM are very close, as expected. This similarity of the FM results has to do with the simplicity and symmetry of the methane molecule. We shall later see that at the water system this is not the case.

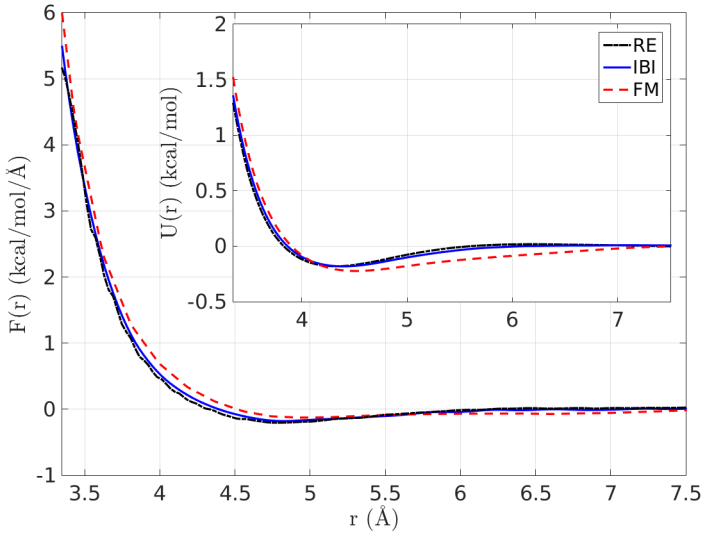


Figure 4.9: Comparison of the results of all methods for the bulk methane at 100K.

We also applied the Relative Entropy approach to the bulk methane at 80K temperature. The minimization was performed with the modified Robins-Monro scheme. Again the Jacobian does not reach zero and it now stabilizes slower at somewhat higher values than the 100K case. The evolution of the Jacobian is plotted at figure 4.10. We stopped at 150 iterations.

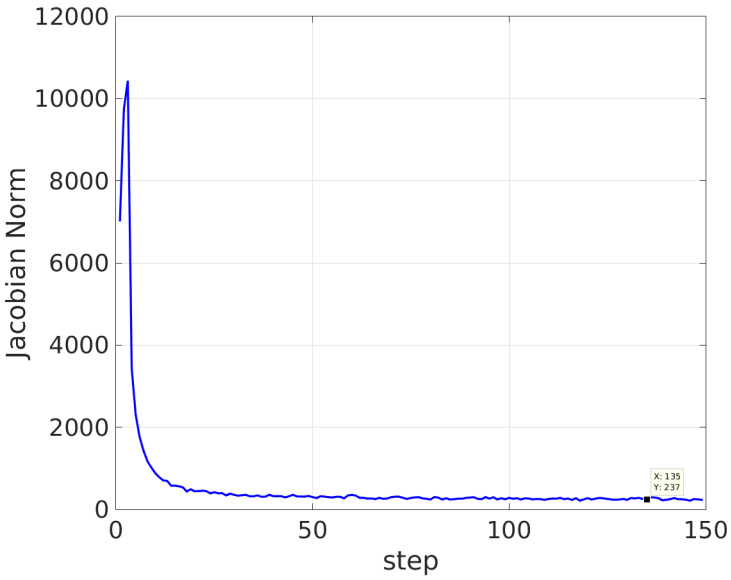
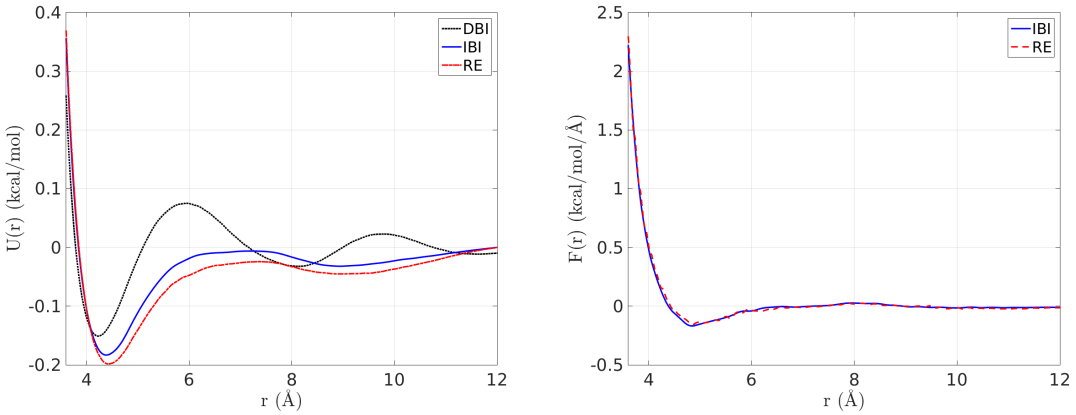


Figure 4.10: RE evolution of the Jacobian for methane at 80K

The results are very close to the IBI results. We see at figures 4.11a and 4.11b that the potentials mostly differ by a constant and that the forces from the IBI and the RE calculations are very close. As one can see from the forces plot, the potential derivative slightly oscillates. Still it's local mean is very close to the IBI. Notice that the oscillations are much smaller than these of the ill "50 samples" case of the 100K system. This happens because we used much less basis functions for the 80K system, 80 linear splines instead of 140 linear splines, increasing the distance interval of each basis function. This helps avoid local noise. The comments regarding treating the oscillations and the non zero Jacobian norm, are the same as these on the 100K system.



(a) Potential (b) Force

Figure 4.11: Relative entropy results for the bulk Methane system at 80K.

Concluding for the bulk methane system at 80K, we compare the potentials and the forces from the three methods at figure 4.12 and find that the results are nearly identical from all methods.

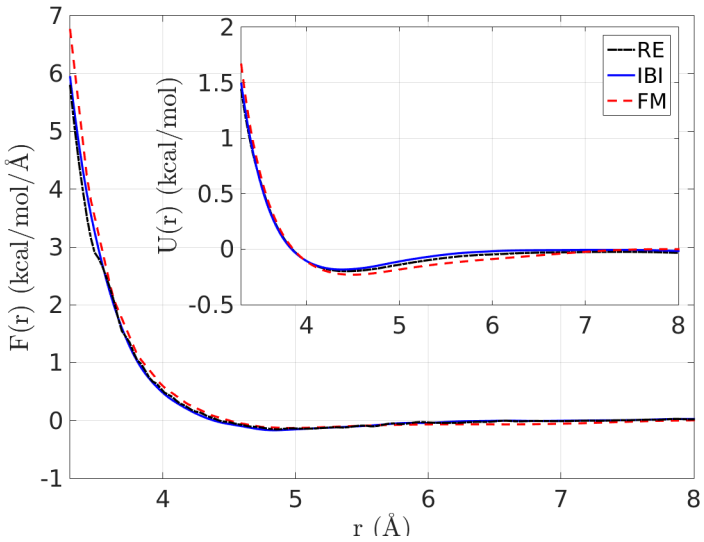


Figure 4.12: Comparison of the results of all methods for the bulk methane at 80K

4.1.3 Methane CG Model Quality

For the methane at 100K system, we used the different CG models (approximated pair CG interaction potentials) derived above, to predict the properties of the bulk CG methane fluid. In all cases we compared with the reference all-atom bulk system. First, in figure 4.13 we examine the structure of the model CG methane liquid by presenting the resulting CG $g(r)$ (RDF) from the different models and from the all-atom data of the system. As expected the CG model derived from the IBI method gives a $g(r)$ very close to the one derived from the analysis of the all-atom data. Interestingly the CG model derived from the FM model is also in good agreement to the reference one, despite the small differences in the CG interaction potential (see figure 4.9). This is not surprising if we consider that for most molecular systems small differences in the interaction potential lead to even smaller differences in the obtained pair correlation function. Overall, differences between the different sets of data of figure 4.13 are less than 5% using square differences defined at the level of $g(r)$. Similar is the case also for other temperatures (T = 80 K) studied.

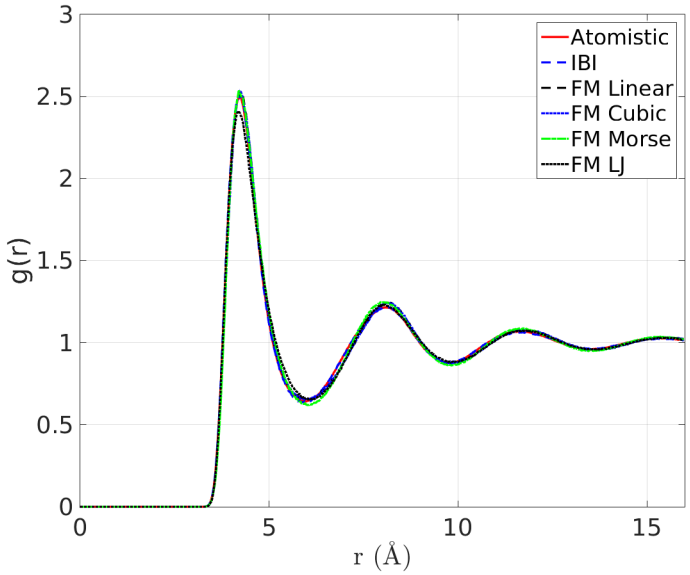


Figure 4.13: CG RDF for the different CG models and from the reference all-atom simulations, for the bulk Methane model at 100K.

Second, in figure 4.14 we shortly discuss the dynamics of the model CG methane liquid at 100K by presenting the mean square displacements (msd's) of the CG particles derived from the different models, as well as from the analysis of the all-atom methane simulations.

Diffusion is the mean by which mass transportation takes place in liquids and gasses. In an atomistic view diffusion occurs as a result of random walks of atoms and molecules. To measure how fast particles travel in a system we define the mean square displacement as,

$$\Delta R(t) = \left\langle (\boldsymbol{Q}(t) - \boldsymbol{Q}(0))^2 \right\rangle \quad ,$$

where $\boldsymbol{Q}$ is a particle's coordinate and the average is sampling every particle at time t.

Mean square displacement increases with time, irregularly for small times, but eventually linearly on a larger scale, as can be easily shown using some simple random walk model.

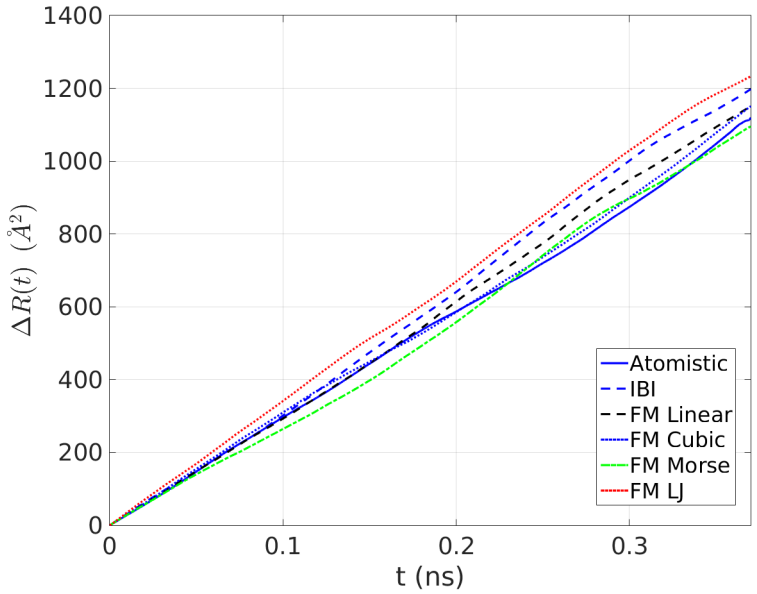


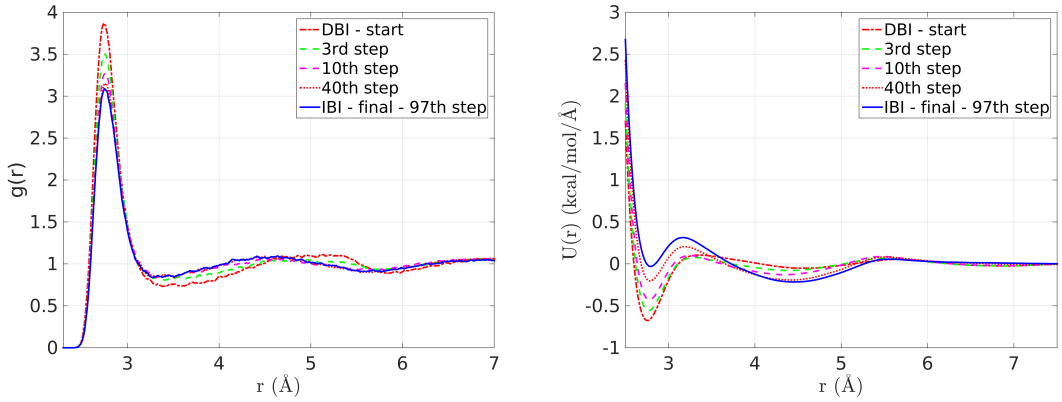
Figure 4.14: Mean square displacement of methane molecules obtained from the different CG models and from the reference all-atom simulations (T = 100 K).

Note that in all CG simulations used here dynamics is not expected to follow the all-atom one, since the intrinsic time scale of the CG model is not the same as that of the underlying chemical system. The reason is that due to the reduced degrees of freedom in the CG description, the friction between the CG beads is significantly reduced compared to what it would be if the monomers were represented in full atomistic detail. We observe in figure 4.14 that the atomistic msd is lower than the CG msds.

4.1.4 Water

The last system we worked with is bulk water at 300K. We obtained a single atomistic MD trajectory, using one of the most typical atomistic force fields, the SPC/E. The model system consists of 1192 molecules at ambient conditions (T = 300 K, P = 1 atm). The time step was 1 fs. A cut-off distance of 10 Angstroms was used, while electrostatic interactions were calculated using the Particle Mesh Ewald method (section 1.3.2). We first equilibrate the system under NPT conditions for about 50 ns. Then an NVT simulation in the average density was performed for 20 ns. All-atom configurations were recorded every 10 ps. For the coarse-grained representation of H2O, we have also used a one-site representation with a pair potential. In the CG representation of water electrostatic interactions were not required to be introduced.

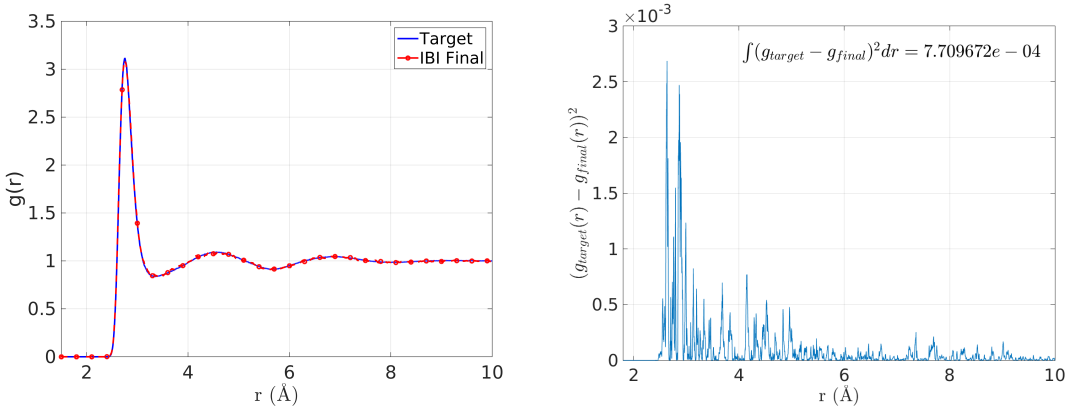
We first performed the IBI procedure. The number of iterations needed for convergence, approximately 100, is much higher than that of the methane system. This has to do both with the tightly packed structure of water and with it's asymmetrical molecule. We can observe the evolution of the obtained RDF and of the potential with respect to the iterations at figures 4.15a and 4.15b.



(a) RDF (b) Potential

Figure 4.15: Evolution of the RDF and of the Potential during the IBI of the water system

The match of the target RDF and the IBI result is displayed at figure 4.16a. The squared difference of the target RDF and the result RDF is depicted at figure 4.16b and we see that it is of the order of 10^{-4}. Again we see that the large source of errors is at the high energy region (at short distances), where we cannot sample enough and have to rely on extrapolation of the potential.



(a) Target RDF vs IBI result (b) Squared distance of Target RDF from IBI

Figure 4.16: IBI RDF quality results for the water system

We then applied the Force Matching and the Relative Entropy procedures. For the FM procedure, we only tested the cubic spline basis. It was not possible to acquire good results from the Relative Entropy method, using the modified Robins-Monro scheme. The Jacobian norm could not approach zero and there was not enough time to properly investigate the numerical issues. Instead we acquired data from [3, 17] to compare with the RE method. There the RE minimization was performed by a plain Newton-Raphson scheme.

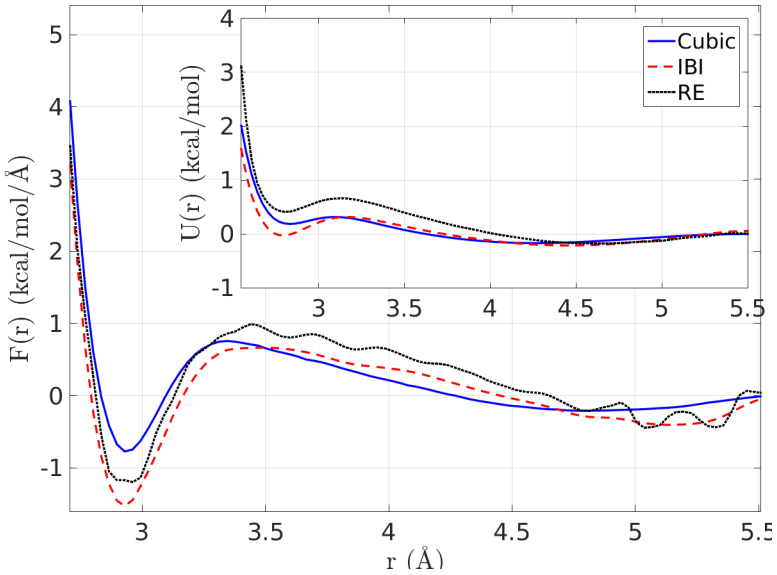


Figure 4.17: Comparison of the three methods for Water.

The comparison of the results from all three methods is shown at plot 4.17. We see that all methods produce similar, but not the same, results. Possible reasons for these discrepancies are related to the fact that FM and RE are only asymptotically equivalent, meaning that finite size basis sets effects might be important during the numerical optimization procedure. Clearly more work is required in order to clarify such differences.

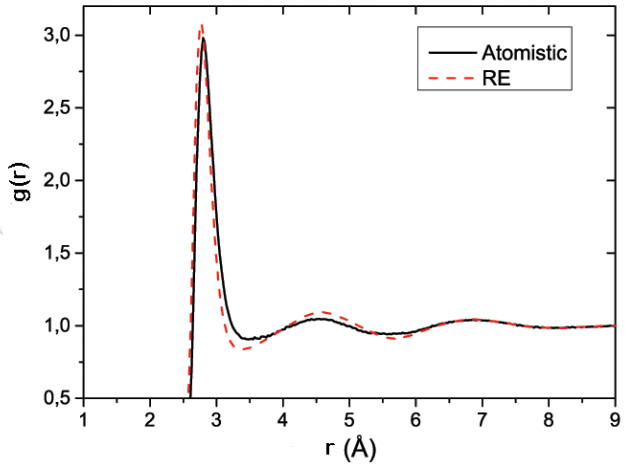


Figure 4.18: Comparison of the Atomistic and the Relative Entropy RDF for Water.

Finally in figure 4.18 we show the CG RDF obtained from the RE minimization problem, together with the reference curve obtained from the analysis of the all-atom data. The curves are very close to each other. However there are small differences, in particular in small distances close to the first maximum. Note that theoretically it is expected that the RE outcome, at the level of the RDF, should agree with the IBI one. The observed differences should be attributed to the numerical issues of the RE method.

4.2 Comments on the numerical issues of the three methods

All methods studied in this work are computationally demanding. As a result parallel calculations are mandatory, not only while performing the MD simulations, but also when calculating the force residual at the FM method or calculating the Jacobian and the Hessian at the RE method. The heavy workload makes it difficult to experiment with different parameters. A single FM experiment may take hours to a day, a single IBI experiment may take a couple of days and a single RE experiment may take a couple of days to a week.

The steadiest method of three is the Force Matching method, as it lacks problems related to the iterative schemes of the other methods. The basis spline functions must not be too narrow and they must be defined only over the range of the MD sampled distances. Information on the latter can be extracted from the RDF. The optimal width of the basis functions is related to the number of MD sampled configurations and it is usually determined empirically.

The IBI method may have convergence problems related to the extrapolation of the short distance potential. The minimal distance before which the extrapolation occurs must be carefully chosen. Such information is extracted from the RDF. Also the method of extrapolation must be robust to always produce repulsive potentials. The number of knots of the interpolating spline must not be too large so as to avoid artificial oscillations of the potential. Smoothing the observed RDF's may be necessary to avoid instabilities of the MD and of the iterative scheme.

The Relative Entropy method has the greatest numerical difficulties of all methods. It suffers from the same difficulties of the short distance potential extrapolation as the IBI method does. Choice of the number of basis functions as well as the number of CG MD simulation steps impacts the noise of the Jacobian and the Hessian and thus of the convergence of the method. Being of the safe end by choosing very high CG MD simulation steps make the iteration times much too large. Also, choice of the conjugate gradient steps and of the a_k sequence of the modified Robins-Monro method both have unexplored properties. Future work should focus on a method to dynamically select the number of CG MD simulation steps as the iterations proceed and maybe to resort to Newton-Raphson steps when the Jacobian stabilizes to a non zero norm.

4.3 Conclusions and Future Work

- Conclusions
  - Results from the three methods for the methane systems are identical. This is not the case for water. There results from the three methods are similar, but not the same. Different methods are expected to produce different results. We attribute the exact match of the methane results to the symmetry of the methane molecule.
  - CG potentials reproduce the atomistic structure. CG dynamics are close to the atomistic but faster, due to less friction in the CG system.
  - Trial potential functional forms must be flexible enough to capture the actual CG potential. The proper choice of the number of spline basis functions is important to the quality of the results. Convergence of the Relative Entropy process can be problematic for complex systems.

  - All methods are computationally demanding. IBI and RE are the most as they require MD simulations at each iterative step. Ad-hoc procedures such as smoothing and extrapolation are important for the convergence of the iterative methods.

- Future Work
  - Perform the RE modified Robins-Monro process for water with cubic spline basis functions.
  - Create adaptive scheme for the modified Robins-Monro RE process, where the sampling is increased when close to convergence, so as to achieve better results.
  - Implement variable width spline basis functions.
  - Conclude on robust extrapolation techniques for the non-bonded and bonded potentials.
  - Work with more complex systems having CG bonded interactions, like hexane.

Appendices

Appendix A

Representation of non-bonded potentials

We follow the symbolism used in [24] and expand to define the basis functions used in our work. The potentials and forces are defined as a linear combination of basis functions.

$$U_{\zeta i}(x) = \sum_d \phi_{\zeta id} u_{\zeta id}(x)$$

Where the ζ index indicates the interaction type (van der Waals, bonded, angular, torsional) and the i index the functional form of the interaction. $u_{\zeta id}$ is the dth basis function and $\phi_{\zeta id}$ is its contribution to the potential.

$$F_{\zeta i}(x) = -dU_{\zeta i}(x)/dx = \sum_d \phi_{\zeta id} f_{\zeta id}(x)$$

$$f_{\zeta id}(x) = -du_{\zeta id}(x)/dx$$

In the force matching method it is natural to explicitly define the $f_{\zeta id}$ basis functions and derive the $u_{\zeta id}$ from them.

$$\begin{aligned} \int_a^b \frac{du_{\zeta id}(x)}{dx} dx = u_{\zeta id}(b) - u_{\zeta id}(a) \Rightarrow \\ u_{\zeta id}(b) = u_{\zeta id}(a) - \int_a^b f_{\zeta id}(x) dx \Rightarrow \\ u_{\zeta id}(x) = u_{\zeta id}(-\infty) - \int_{-\infty}^x f_{\zeta id}(y) dy \end{aligned}$$

Note that the above are valid only for var der Waals potentials where $u_{\zeta id}(\pm\infty) = 0$. Extension for angular and torsional potentials is straightforward as the integrals may simply start from zero. For the bonded potentials one should not only start from zero but also branch the integral towards the two opposite sign infinities.

A mesh of $N_{\zeta i}$ equally spaced grid points is defined around which the basis functions are defined.

$$\{x_{\zeta id}\} = \{x_{\zeta i1} + (d-1)\Delta_{\zeta i}\} \quad , \quad d = 1, ..., N_{\zeta i}$$

$$\Delta_{\zeta i} = \frac{max(x_{\zeta i}) - min(x_{\zeta i})}{N_{\zeta i} - 1}$$

A.1 Linear Basis

A linear basis is defined for the forces from *(A3-A5)* at [24] as below.

$$A_d(x) = 1 - B_d(x) = \frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}}$$
$$B_d(x) = 1 - A_d(x) = \frac{x - x_{\zeta id}}{x_{\zeta i(d+1)} - x_{\zeta id}}$$

$$\eta_d(x) = \begin{cases} B_{d-1}(x) & x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ A_d(x) & x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & \text{elsewhere} \end{cases}$$

The basis functions $f_{\zeta id}(x) = \eta_d(x)$ are such that

$$F_{\zeta i}(x) = \sum_d \phi_{\zeta id} f_{\zeta id}(x)$$

Working a bit with $\eta_d(x)$ we get

$$f_{\zeta id} : [x_{\zeta i1}, x_{\zeta i N_{\zeta i}}] \mapsto [0, 1]$$

$$f_{\zeta id}(x) = \begin{cases} \frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} & x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ \frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}} & x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & \text{elsewhere} \end{cases}$$

We integrate the force to get the potential basis functions $u_{\zeta id}(x)$

$$\int_{-\infty}^{x} f_{\zeta id}(y)dy = \int_{\alpha}^{\beta} \frac{y - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} dy + \int_{\beta}^{\gamma} \frac{x_{\zeta i(d+1)} - y}{x_{\zeta i(d+1)} - x_{\zeta id}} dy \Rightarrow$$

$$u_{\zeta id}(x) = \begin{cases} -\frac{(\beta^2-\alpha^2)/2 - x_{\zeta i(d-1)}(\beta-\alpha)}{x_{\zeta id} - x_{\zeta i(d-1)}} - \frac{x_{\zeta i(d+1)}(\gamma-\beta) - (\gamma^2-\beta^2)/2}{x_{\zeta i(d+1)} - x_{\zeta id}} & 1 < d < N_{\zeta i} \\ -\frac{x_{\zeta i(d+1)}(\gamma-\beta) - (\gamma^2-\beta^2)/2}{x_{\zeta i(d+1)} - x_{\zeta id}} & d = 1 \\ -\frac{(\beta^2-\alpha^2)/2 - x_{\zeta i(d-1)}(\beta-\alpha)}{x_{\zeta id} - x_{\zeta i(d-1)}} & d = N_{\zeta i} \end{cases}$$

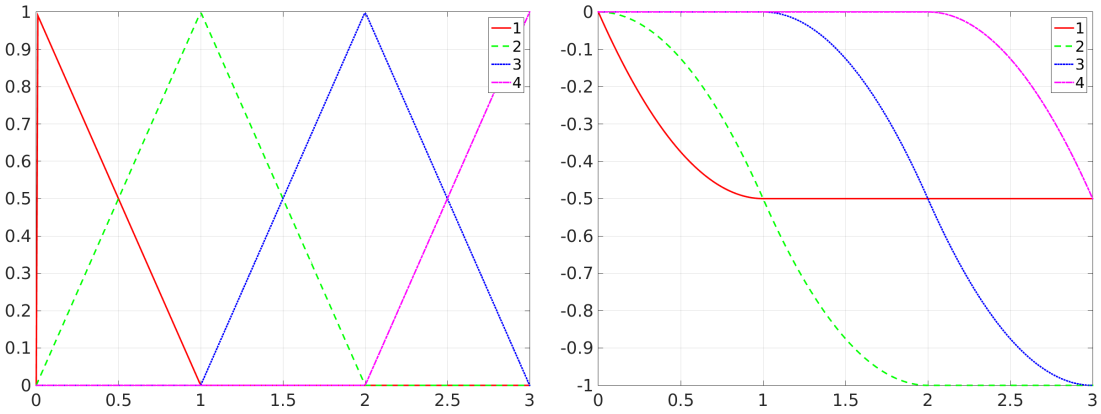
$$\alpha = min(x_{\zeta i(d-1)}, x)$$
$$\beta = min(x_{\zeta id}, x)$$
$$\gamma = min(x_{\zeta i(d+1)}, x)$$

As we work on meshes of equally spaced points we may simplify our equations by setting $h \equiv \Delta_{\zeta i} = x_{\zeta id} - x_{\zeta i(d-1)}$.

$$f_{\zeta id}(x) = \begin{cases} \frac{x - x_{\zeta id} + h}{h} & x_{\zeta id} - h < x \leq x_{\zeta id} \\ \frac{x_{\zeta id} - x + h}{h} & x_{\zeta id} < x \leq x_{\zeta id} + h \\ 0 & \text{elsewhere} \end{cases} \tag{A.1}$$

$$u_{\zeta id}(x) = \begin{cases} -\frac{(\beta^2-\alpha^2)/2 - (x_{\zeta id}-h)(\beta-\alpha)}{h} - \frac{(x_{\zeta id}+h)(\gamma-\beta) - (\gamma^2-\beta^2)/2}{h} & 1 < d < N_{\zeta i} \\ -\frac{(x_{\zeta id}+h)(\gamma-\beta) - (\gamma^2-\beta^2)/2}{h} & d = 1 \\ -\frac{(\beta^2-\alpha^2)/2 - (x_{\zeta id}-h)(\beta-\alpha)}{h} & d = N_{\zeta i} \end{cases}$$

$$\alpha = min(x_{\zeta id} - h, x)$$
$$\beta = min(x_{\zeta id}, x)$$
$$\gamma = min(x_{\zeta id} + h, x)$$



(a) Force (b) Potential

Figure A.1: Linear basis functions for the force matching scheme.

A.2 Cubic Spline Basis

Now we need $2N_{\zeta i}$ coefficients of $\phi_{\zeta i}$ to define the potential on a mesh of $N_{\zeta i}$ equally spaced points. The following are from equations (A3-A9) of [24].

$$
\begin{aligned}
A_d(x) &= \frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}} \\
B_d(x) &= \frac{x - x_{\zeta id}}{x_{\zeta i(d+1)} - x_{\zeta id}} \\
C_d(x) &= \frac{1}{6}\left(A_d^3(x) - A_d(x)\right)\left(x_{\zeta i(d+1)} - x_{\zeta id}\right)^2 \\
D_d(x) &= \frac{1}{6}\left(B_d^3(x) - B_d(x)\right)\left(x_{\zeta i(d+1)} - x_{\zeta id}\right)^2 \\
\eta_d(x) &= \begin{cases} B_{d-1}(x) & , \quad x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ A_d(x) & , \quad x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & , \quad \text{elsewhere} \end{cases} \\
\mu_d(x) &= \begin{cases} D_{d-1}(x) & , \quad x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ C_d(x) & , \quad x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & , \quad \text{elsewhere} \end{cases}
\end{aligned}
$$

The force is now expressed in terms of the basis functions as below

$$F_{\zeta i}(x) = \sum_{d} \phi_{\zeta id} f_{\zeta id}(x) = \sum_{d=1}^{N_{\zeta i}} \left[\phi_{\zeta i(2d-1)} \eta_d(x) + \phi_{\zeta i(2d)} \mu_d(x) \right]$$

It shows that

$$f_{\zeta id}(x) = \begin{cases} \eta_{\frac{d+1}{2}}(x) & , \quad d \text{ is odd} \\ \mu_{\frac{d}{2}}(x) & , \quad d \text{ is even} \end{cases} \tag{A.2}$$

We need $B_{d-1}(x)$ and $D_{d-1}(x)$.

$$B_{d-1}(x) = \frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}}$$
$$D_{d-1}(x) = \frac{1}{6} \left[\left(\frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} \right)^3 - \frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} \right] \left(x_{\zeta id} - x_{\zeta i(d-1)} \right)^2$$

We explicitly write $\eta_d(x)$ and $\mu_d(x)$.

$$\eta_d(x) = \begin{cases} \frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} & , \quad x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ \frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}} & , \quad x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

$$\mu_d(x) = \begin{cases} \frac{1}{6} \left[\left(\frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} \right)^3 - \frac{x - x_{\zeta i(d-1)}}{x_{\zeta id} - x_{\zeta i(d-1)}} \right] \left(x_{\zeta id} - x_{\zeta i(d-1)} \right)^2 & , \quad x_{\zeta i(d-1)} < x \leq x_{\zeta id} \\ \frac{1}{6} \left[\left(\frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}} \right)^3 - \frac{x_{\zeta i(d+1)} - x}{x_{\zeta i(d+1)} - x_{\zeta id}} \right] \left(x_{\zeta i(d+1)} - x_{\zeta id} \right)^2 & , \quad x_{\zeta id} < x \leq x_{\zeta i(d+1)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

When d is odd

$$\eta_{\frac{d+1}{2}}(x) = \begin{cases} \frac{x - x_{\zeta i\left(\frac{d-1}{2}\right)}}{x_{\zeta i\left(\frac{d+1}{2}\right)} - x_{\zeta i\left(\frac{d-1}{2}\right)}} & , \quad x_{\zeta i\left(\frac{d-1}{2}\right)} < x \leq x_{\zeta i\left(\frac{d+1}{2}\right)} \\ \frac{x_{\zeta i\left(\frac{d+3}{2}\right)} - x}{x_{\zeta i\left(\frac{d+3}{2}\right)} - x_{\zeta i\left(\frac{d+1}{2}\right)}} & , \quad x_{\zeta i\left(\frac{d+1}{2}\right)} < x \leq x_{\zeta i\left(\frac{d+3}{2}\right)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

The basis is of dimension $2N_{\zeta i}$ and so d runs from 1 to $2N_{\zeta i}$ with the last odd being $2N_{\zeta i}-1$. Thus for odd d we may define $\lambda = \frac{d+1}{2}, \lambda = 1, ..., N_{\zeta i}$. So now each λ correspond to one grid point.

$$\eta_\lambda(x) = \begin{cases} \frac{x - x_{\zeta i(\lambda-1)}}{x_{\zeta i\lambda} - x_{\zeta i(\lambda-1)}} & , \quad x_{\zeta i(\lambda-1)} < x \leq x_{\zeta i\lambda} \\ \frac{x_{\zeta i(\lambda+1)} - x}{x_{\zeta i(\lambda+1)} - x_{\zeta i\lambda}} & , \quad x_{\zeta i\lambda} < x \leq x_{\zeta i(\lambda+1)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

On a uniform grid with spacing h this becomes

$$\eta_\lambda(x) = \begin{cases} \frac{x - x_{\zeta i\lambda} + h}{h} & , \quad x_{\zeta i\lambda} - h < x \leq x_{\zeta i\lambda} \\ \frac{x_{\zeta i\lambda} - x + h}{h} & , \quad x_{\zeta i\lambda} < x \leq x_{\zeta i\lambda} + h \\ 0 & , \quad \text{elsewhere} \end{cases}$$

When d is even

$$\mu_{\frac{d}{2}}(x) = \begin{cases} \frac{1}{6}\left[\left(\frac{x - x_{\zeta i\left(\frac{d}{2}-1\right)}}{x_{\zeta i\left(\frac{d}{2}\right)} - x_{\zeta i\left(\frac{d}{2}-1\right)}}\right)^3 - \frac{x - x_{\zeta i\left(\frac{d}{2}-1\right)}}{x_{\zeta i\left(\frac{d}{2}\right)} - x_{\zeta i\left(\frac{d}{2}-1\right)}}\right]\left(x_{\zeta i\left(\frac{d}{2}\right)} - x_{\zeta i\left(\frac{d}{2}-1\right)}\right)^2 & , \quad x_{\zeta i\left(\frac{d}{2}-1\right)} < x \leq x_{\zeta i\left(\frac{d}{2}\right)} \\ \frac{1}{6}\left[\left(\frac{x_{\zeta i\left(\frac{d}{2}+1\right)} - x}{x_{\zeta i\left(\frac{d}{2}+1\right)} - x_{\zeta i\left(\frac{d}{2}\right)}}\right)^3 - \frac{x_{\zeta i\left(\frac{d}{2}+1\right)} - x}{x_{\zeta i\left(\frac{d}{2}+1\right)} - x_{\zeta i\left(\frac{d}{2}\right)}}\right]\left(x_{\zeta i\left(\frac{d}{2}+1\right)} - x_{\zeta i\left(\frac{d}{2}\right)}\right)^2 & , \quad x_{\zeta i\left(\frac{d}{2}\right)} < x \leq x_{\zeta i\left(\frac{d}{2}+1\right)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

The basis is of dimension $2N_{\zeta i}$ and so d runs from 1 to $2N_{\zeta i}$. Thus for even d we may define $k = \frac{d}{2}, k = 1, ..., N_{\zeta i}$. So now each k correspond to one grid point.

$$\mu_k(x) = \begin{cases} \frac{1}{6}\left[\left(\frac{x - x_{\zeta i(k-1)}}{x_{\zeta ik} - x_{\zeta i(k-1)}}\right)^3 - \frac{x - x_{\zeta i(k-1)}}{x_{\zeta ik} - x_{\zeta i(k-1)}}\right]\left(x_{\zeta ik} - x_{\zeta i(k-1)}\right)^2 & , \quad x_{\zeta i(k-1)} < x \leq x_{\zeta ik} \\ \frac{1}{6}\left[\left(\frac{x_{\zeta i(k+1)} - x}{x_{\zeta i(k+1)} - x_{\zeta ik}}\right)^3 - \frac{x_{\zeta i(k+1)} - x}{x_{\zeta i(k+1)} - x_{\zeta ik}}\right]\left(x_{\zeta i(k+1)} - x_{\zeta ik}\right)^2 & , \quad x_{\zeta ik} < x \leq x_{\zeta i(k+1)} \\ 0 & , \quad \text{elsewhere} \end{cases}$$

On a uniform grid with spacing h this becomes

$$\mu_k(x) = \begin{cases} \frac{1}{6}\left[\left(\frac{x-x_{\zeta ik}+h}{h}\right)^3 - \frac{x-x_{\zeta ik}+h}{h}\right]h^2 & , \quad x_{\zeta ik} - h < x \leq x_{\zeta ik} \\ \frac{1}{6}\left[\left(\frac{x_{\zeta ik}-x+h}{h}\right)^3 - \frac{x_{\zeta ik}-x+h}{h}\right]h^2 & , \quad x_{\zeta ik} < x \leq x_{\zeta ik} + h \\ 0 & , \quad \text{elsewhere} \end{cases}$$

We substitute η and μ into A.2 to get

$$f_{\zeta id}(x) = \begin{cases} \eta_\lambda(x) & , \quad d \text{ is odd} \quad , \quad \lambda = \frac{d+1}{2} \\ \mu_k(x) & , \quad d \text{ is even} \quad , \quad k = \frac{d}{d} \end{cases}$$

Which we can further simplify by setting

$$\boxed{k_d = \begin{cases} \frac{d+1}{2} & , d \text{ is odd} \\ \frac{d}{2} & , d \text{ is even} \end{cases}}$$

So

$$f_{\zeta id}(x) = \begin{cases} \eta_{k_d}(x) & , \quad d \text{ is odd} \\ \mu_{k_d}(x) & , \quad d \text{ is even} \end{cases}$$

Finally

$$\boxed{f_{\zeta id}(x) = \begin{cases} \frac{x-x_{\zeta ik_d}+h}{h} & , \quad x_{\zeta ik_d} - h < x \leq x_{\zeta ik_d} \quad , d \text{ is odd} \\ \frac{1}{6}\left[\left(\frac{x-x_{\zeta ik_d}+h}{h}\right)^3 - \frac{x-x_{\zeta ik_d}+h}{h}\right]h^2 & , \quad x_{\zeta ik_d} - h < x \leq x_{\zeta ik_d} \quad , d \text{ is even} \\ \frac{x_{\zeta ik_d}-x+h}{h} & , \quad x_{\zeta ik_d} < x \leq x_{\zeta ik_d} + h \quad , d \text{ is odd} \\ \frac{1}{6}\left[\left(\frac{x_{\zeta ik_d}-x+h}{h}\right)^3 - \frac{x_{\zeta ik_d}-x+h}{h}\right]h^2 & , \quad x_{\zeta ik_d} < x \leq x_{\zeta ik_d} + h \quad , d \text{ is even} \\ 0 & , \quad \text{otherwise} \end{cases}}$$

In this form it is easier to see that for each interval of our mesh there are both cubic and linear contributions originating from two successive basis functions. Below we verify some statements in [24].

We observe that $\mu_{k_d}(x) \to 0$ and $\eta_{k_d}(x) \to 1$ when $x \to x_{\zeta i k_d}$. It is easy to see that half the *phi* coefficients correspond to the magnitude of the force at the grid points. Indeed $\phi_{\zeta i(2d-1)} = F_{\zeta i}(x_{\zeta i d})$.

The derivative is discontinuous at each grid point. Indeed both η_{k_d} and μ_{k_d} contributions to the derivative at the $x_{\zeta i k_d}$ grid point are of the opposite sign as $x \to x_{\zeta i k_d}$ from the left and from the right.

$$\frac{df_{\zeta i d}(x)}{dx} = \begin{cases} \frac{1}{h} & , \quad x_{\zeta i k_d} - h < x \leq x_{\zeta i k_d} \quad , d \text{ is odd} \\ \frac{\left(x - x_{\zeta i k_d} + h\right)^2}{2h} - h & , \quad x_{\zeta i k_d} - h < x \leq x_{\zeta i k_d} \quad , d \text{ is even} \\ -\frac{1}{h} & , \quad x_{\zeta i k_d} < x \leq x_{\zeta i k_d} + h \quad , d \text{ is odd} \\ -\frac{\left(x_{\zeta i k_d} - x + h\right)^2}{2h} + h & , \quad x_{\zeta i k_d} < x \leq x_{\zeta i k_d} + h \quad , d \text{ is even} \\ 0 & , \quad \text{otherwise} \end{cases}$$

We also verify the statement that the μ_{k_d} contributions correspond to the limit of the second derivative at the grid point.

$$\frac{d^2 f_{\zeta i d}(x)}{dx^2} = \begin{cases} 0 & , \quad x_{\zeta i k_d} - h < x \leq x_{\zeta i k_d} \quad , d \text{ is odd} \\ \frac{x - x_{\zeta i k_d} + h}{h} & , \quad x_{\zeta i k_d} - h < x \leq x_{\zeta i k_d} \quad , d \text{ is even} \\ 0 & , \quad x_{\zeta i k_d} < x \leq x_{\zeta i k_d} + h \quad , d \text{ is odd} \\ \frac{x_{\zeta i k_d} - x + h}{h} & , \quad x_{\zeta i k_d} < x \leq x_{\zeta i k_d} + h \quad , d \text{ is even} \\ 0 & , \quad \text{otherwise} \end{cases}$$

So as $x \to x_{\zeta i k_d}$ only the μ_{k_d} contributions to the second derivative remain and they become 1, meaning that $\phi_{\zeta i(2d)} = \lim_{x \to x_{\zeta i d}} d^2 F_{\zeta i}/dx^2$.

Now we want to integrate $f_{\zeta i d}$ to get the potential basis functions $u_{\zeta i d}$.

$$\int_{-\infty}^{x} f_{\zeta i d}(y) dy = \begin{cases} \int_{\alpha}^{\beta} \frac{y - x_{\zeta i k_d} + h}{h} dy + \int_{\beta}^{\gamma} \frac{x_{\zeta i k_d} - y + h}{h} dy & , d \text{ is odd} \\ \frac{h^2}{6} \left[\int_{\alpha}^{\beta} \left(\frac{y - x_{\zeta i k_d} + h}{h} \right)^3 dy + \int_{\beta}^{\gamma} \left(\frac{x_{\zeta i k_d} - y + h}{h} \right)^3 dy \right] & \\ -\frac{h^2}{6} \left[\int_{\alpha}^{\beta} \frac{y - x_{\zeta i k_d} + h}{h} dy + \int_{\beta}^{\gamma} \frac{x_{\zeta i k_d} - y + h}{h} dy \right] & , \quad d \text{ is even} \end{cases}$$

$$\alpha = min(x_{\zeta ik_d} - h, x)$$
$$\beta = min(x_{\zeta ik_d}, x)$$
$$\gamma = min(x_{\zeta ik_d} + h, x)$$

We make use of

$$\int_{\alpha}^{\beta} \frac{y - x_{\zeta ik_d} + h}{h} dy = -\frac{(\alpha - \beta)\left(\alpha + \beta - 2x_{\zeta \mathrm{ik}_d} + 2h\right)}{2h}$$

$$\int_{\alpha}^{\beta} \left(\frac{y - x_{\zeta ik_d} + h}{h}\right)^3 dy = \frac{\left(\beta - x_{\zeta \mathrm{ik}_d} + h\right)^4 - \left(\alpha - x_{\zeta \mathrm{ik}_d} + h\right)^4}{4h^3}$$

$$\int_{\beta}^{\gamma} \frac{x_{\zeta ik_d} - y + h}{h} dy = \frac{(\beta - \gamma)\left(\beta + \gamma - 2x_{\zeta \mathrm{ik}_d} - 2h\right)}{2h}$$

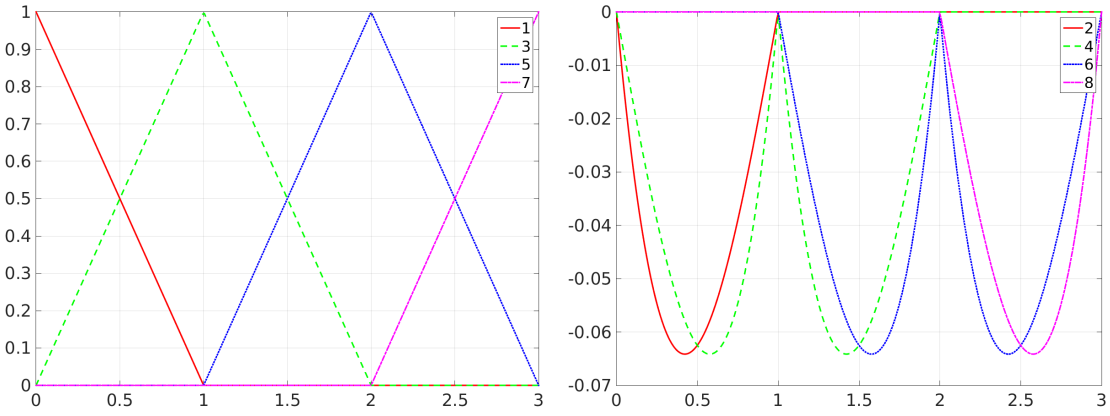
$$\int_{\beta}^{\gamma} \left(\frac{x_{\zeta ik_d} - y + h}{h}\right)^3 dy = \frac{\left(-\beta + x_{\zeta \mathrm{ik}_d} + h\right)^4 - \left(-\gamma + x_{\zeta \mathrm{ik}_d} + h\right)^4}{4h^3}$$

So

$$u_{\zeta id}(x) = -\int_{-\infty}^{x} f_{\zeta id}(y) dy$$

$$= \begin{cases} -\frac{(\beta-\gamma)\left(\beta+\gamma-2x_{\zeta \mathrm{ik}_d}-2h\right)}{2h} & , \quad d = 1 \\ \frac{(\alpha-\beta)\left(\alpha+\beta-2x_{\zeta \mathrm{ik}_d}+2h\right)}{2h} - \frac{(\beta-\gamma)\left(\beta+\gamma-2x_{\zeta \mathrm{ik}_d}-2h\right)}{2h} & , \quad d \text{ is odd} \neq 1 \\ -\frac{\left(\beta-x_{\zeta \mathrm{ik}_d}+h\right)^4-\left(\alpha-x_{\zeta \mathrm{ik}_d}+h\right)^4}{24h} - \frac{\left(-\beta+x_{\zeta \mathrm{ik}_d}+h\right)^4-\left(-\gamma+x_{\zeta \mathrm{ik}_d}+h\right)^4}{24h} & \\ -\frac{(\alpha-\beta)\left(\alpha+\beta-2x_{\zeta \mathrm{ik}_d}+2h\right)h}{12} + \frac{(\beta-\gamma)\left(\beta+\gamma-2x_{\zeta \mathrm{ik}_d}-2h\right)h}{12} & , \quad d \text{ is even} \neq N_{\zeta i} \\ -\frac{\left(\beta-x_{\zeta \mathrm{ik}_d}+h\right)^4-\left(\alpha-x_{\zeta \mathrm{ik}_d}+h\right)^4}{24h} - \frac{(\alpha-\beta)\left(\alpha+\beta-2x_{\zeta \mathrm{ik}_d}+2h\right)h}{12} & , \quad d = N_{\zeta i} \end{cases}$$

$$\alpha = min(x_{\zeta ik_d} - h, x)$$
$$\beta = min(x_{\zeta ik_d}, x)$$
$$\gamma = min(x_{\zeta ik_d} + h, x)$$



(a) Odd (b) Even

Figure A.2: Cubic basis functions (for the force) for the force matching scheme.

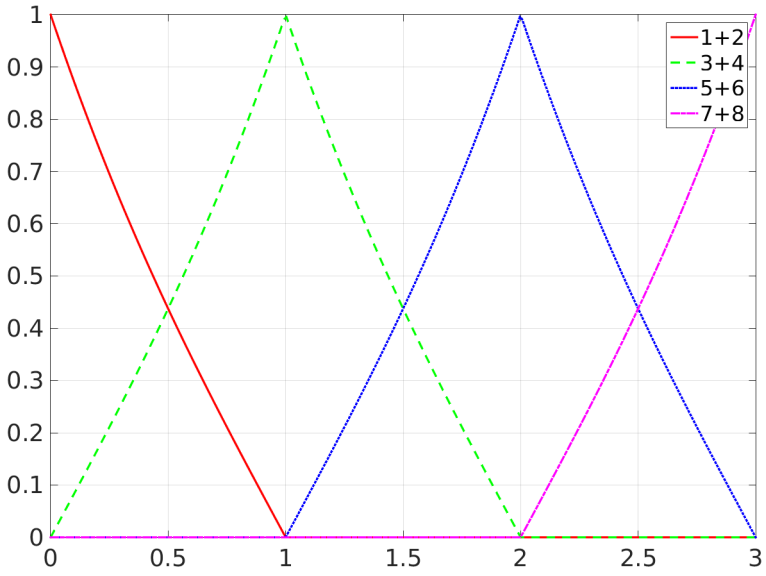
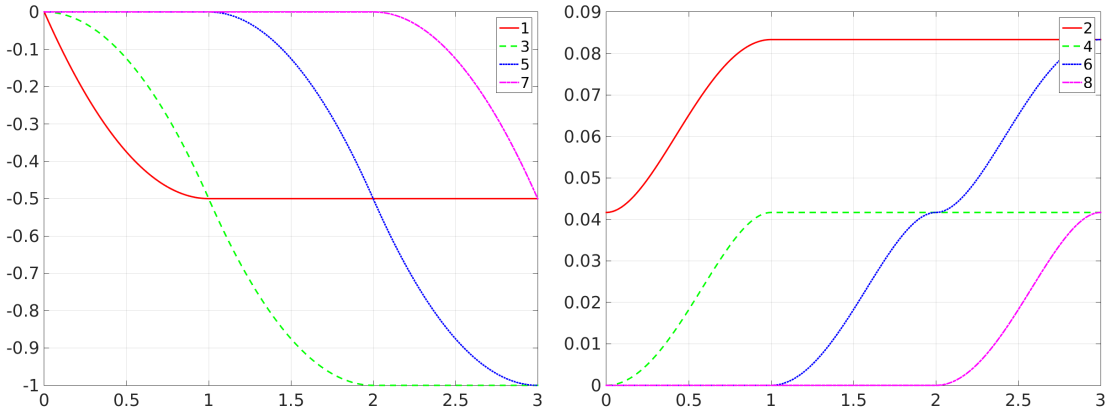


Figure A.3: Cubic basis functions (for the force) for the force matching scheme (combined odd + even).



(a) Odd (b) Even

Figure A.4: Cubic basis functions (for the potential) for the force matching scheme.

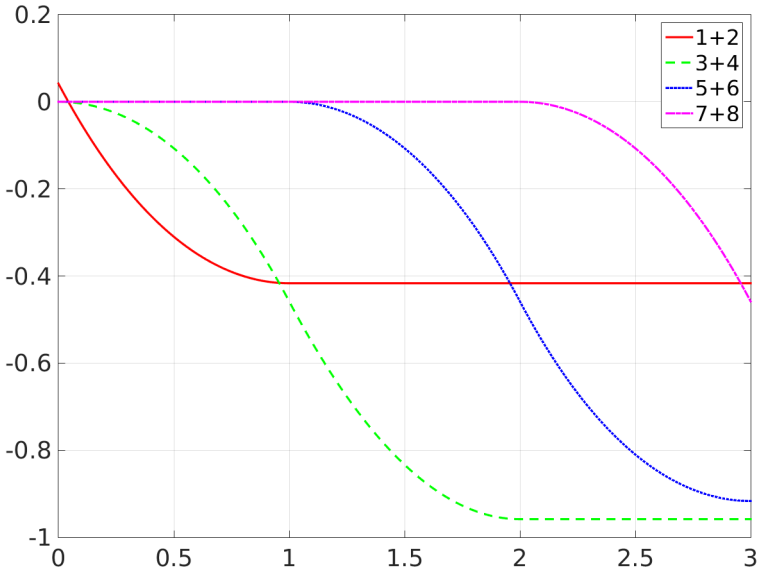


Figure A.5: Cubic basis functions (for the potential) for the force matching scheme (combined odd + even).

Appendix B

Relation of the Relative Entropy between the Atomistic and the CG probability distributions

We first express the atomistic probability in relation to the CG probability and the back-mapping probability.

$$p(\boldsymbol{q}) = P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q}) \tag{B.1}$$

We start from the atomistic expression for the relative entropy.

$$\begin{aligned}
S_{rel}(\hat{U}) &= \left\langle ln\frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))}\right\rangle_{\boldsymbol{q}} = \int d\boldsymbol{q}\ \ p(\boldsymbol{q})ln\frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} \\
&= \int\int d\boldsymbol{Q}d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})ln\frac{P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \\
&= \int\int d\boldsymbol{Q}d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})\left(ln\frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} + ln\ \ p(\boldsymbol{q}|\boldsymbol{Q})\right) \\
&= \int\int d\boldsymbol{Q}d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})ln\frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \\
&+ \int\int d\boldsymbol{Q}d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})P(\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})\ \ ln\ \ p(\boldsymbol{q}|\boldsymbol{Q}) \\
&= \int d\boldsymbol{Q}\ \ P(\boldsymbol{Q})ln\frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})}\int d\boldsymbol{q}\delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q}) \\
&+ \int d\boldsymbol{Q}P(\boldsymbol{Q})\int d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})\ \ ln\ \ p(\boldsymbol{q}|\boldsymbol{Q}) \\
&= \int d\boldsymbol{Q}\ \ P(\boldsymbol{Q})ln\frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} + \int d\boldsymbol{Q}P(\boldsymbol{Q})\int d\boldsymbol{q}\ \ \delta(\boldsymbol{M}(\boldsymbol{q})-\boldsymbol{Q})p(\boldsymbol{q}|\boldsymbol{Q})\ \ ln\ \ p(\boldsymbol{q}|\boldsymbol{Q})
\end{aligned}$$

$$= \left\langle ln \frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \right\rangle_{\boldsymbol{Q}} - S_{map}(\boldsymbol{M})$$

Where

$$S_{map}((M)) = -\int d\boldsymbol{Q} P(\boldsymbol{Q}) \int_{\{\boldsymbol{q}:\boldsymbol{M}(\boldsymbol{q})=\boldsymbol{Q}\}} d\boldsymbol{q} \;\; p(\boldsymbol{q}|\boldsymbol{Q}) \;\; ln \;\; p(\boldsymbol{q}|\boldsymbol{Q})$$

Finally

$$S_{rel} = \left\langle ln \frac{P(\boldsymbol{Q})}{P_{\hat{U}}(\boldsymbol{Q})} \right\rangle_{\boldsymbol{Q}} = \left\langle ln \frac{p(\boldsymbol{q})}{P_{\hat{U}}(\boldsymbol{M}(\boldsymbol{q}))} \right\rangle_{\boldsymbol{q}} + S_{map}(\boldsymbol{M})$$

Appendix C

Excerpts of Code

C.1 Iterative Boltzmann Inversion

Below is the listing of the Python code performing the IBI iterations, for the methane system at 100K.

Listing C.1: Python code performing the IBI iterations.

```
import shutil
import subprocess
from numpy import *
from scipy import interpolate
from scipy.optimize import curve_fit
from matplotlib.pyplot import *
from scipy.interpolate import InterpolatedUnivariateSpline


def savitzky_golay(y, window_size, order, deriv=0, rate=1):
  r"""Smooth (and optionally differentiate) data with a Savitzky-Golay filter.
  The Savitzky-Golay filter removes high frequency noise from data.
  It has the advantage of preserving the original shape and
  features of the signal better than other types of filtering
  approaches, such as moving averages techniques.
  Parameters
  ----------
  y : array_like, shape (N,)
    the values of the time history of the signal.
  window_size : int
    the length of the window. Must be an odd integer number.
  order : int
    the order of the polynomial used in the filtering.
    Must be less then `window_size` - 1.
  deriv: int
    the order of the derivative to compute (default = 0 means only smoothing)
  Returns
  -------
  ys : ndarray, shape (N)
    the smoothed signal (or it's n-th derivative).
  Notes
  -----
  The Savitzky-Golay is a type of low-pass filter, particularly
  suited for smoothing noisy data. The main idea behind this
  approach is to make for each point a least-square fit with a
  polynomial of high order over a odd-sized window centered at
  the point.
  Examples
  --------
  t = np.linspace(-4, 4, 500)
  y = np.exp( -t**2 ) + np.random.normal(0, 0.05, t.shape)
```

```
  ysg = savitzky_golay(y, window_size=31, order=4)
  import matplotlib.pyplot as plt
  plt.plot(t, y, label='Noisy signal')
  plt.plot(t, np.exp(-t**2), 'k', lw=1.5, label='Original signal')
  plt.plot(t, ysg, 'r', label='Filtered signal')
  plt.legend()
  plt.show()
  References
  ----------
  .. [1] A. Savitzky, M. J. E. Golay, Smoothing and Differentiation of
    Data by Simplified Least Squares Procedures. Analytical
    Chemistry, 1964, 36 (8), pp 1627-1639.
  .. [2] Numerical Recipes 3rd Edition: The Art of Scientific Computing
    W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery
    Cambridge University Press ISBN-13: 9780521880688
  """
  import numpy as np
  from math import factorial

  try:
    window_size = np.abs(np.int(window_size))
    order = np.abs(np.int(order))
  except ValueError, msg:
    raise ValueError("window_size and order have to be of type int")
  if window_size % 2 != 1 or window_size < 1:
    raise TypeError("window_size size must be a positive odd number")
  if window_size < order + 2:
    raise TypeError("window_size is too small for the polynomials order")
  order_range = range(order+1)
  half_window = (window_size -1) // 2
  # precompute coefficients
  b = np.mat([[k**i for i in order_range] for k in range(-half_window, half_window+1)])
  m = np.linalg.pinv(b).A[deriv] * rate**deriv * factorial(deriv)
  # pad the signal at the extremes with
  # values taken from the signal itself
  firstvals = y[0] - np.abs( y[1:half_window+1][::-1] - y[0] )
  lastvals = y[-1] + np.abs(y[-half_window-1:-1][::-1] - y[-1])
  y = np.concatenate((firstvals, y, lastvals))
  return np.convolve( m[::-1], y, mode='valid')

def smooth_g( g ):
  n = g.shape[0]
  d = zeros( n )
  for i in range(n):
    d[i] = g[i,1]
  sm = savitzky_golay(d, 21, 3)
  res = zeros((n,2))
  for i in range(n):
    res[i,0] = g[i,0]
    res[i,1] = sm[i]
  return res

def interpolation_of( tabular_data, start=0 ):
  n = tabular_data.shape[0]-start
  x = zeros(n)
        y = zeros(n)
  for i in range(n):
    x[i] = tabular_data[i+start,0]
    y[i] = tabular_data[i+start,1]
  return InterpolatedUnivariateSpline(x, y)

def exp_fit(x,a,b,c):
  return a*exp(-b*x)+c

def lin_fit(x,a,b):
  return -a*x+b


def extrapolate_u(u, firstgood):
  global extrapolation_sample_region

  uinterp = interpolation_of( u, firstgood )

  x0 = u[firstgood,0]

  nfit = 10
  xfit = zeros(nfit)
  yfit = zeros(nfit)
  for i in range(nfit):
    xfit[i] = x0 + extrapolation_sample_region*i/nfit
    yfit[i] = uinterp( xfit[i] )

  p0 = array([1,1,0])
```

```
  popt, pcov = curve_fit(exp_fit, xfit, yfit, p0, maxfev=100000)
  a = popt[0]
  b = popt[1]
  c = popt[2]

  p0 = array([1,0])
  popt, pcov = curve_fit(lin_fit, xfit, yfit, p0, maxfev=100000)
  al = popt[0]
  bl = popt[1]

  use_lin_not_exp = (b <= 0 or a <= 0 or a*b*exp(-b*x0) < al)
  use_lin_not_exp = False

  result = zeros((u.shape[0], 2))
  for i in range(u.shape[0]):
    result[i,0] = u[i,0]
    if i >= firstgood:
      result[i,1] = u[i,1]
    else:
      if use_lin_not_exp:
        result[i,1] = lin_fit(u[i,0], al,bl)
      else:
        result[i,1] = exp_fit(u[i,0], a,b,c)

  return result

def calc_u_from_gofr( g_table, T, n_points ):
  global Kb
  global g_zero_tol
  global cutoff

  # create g(r) interpolation

  g = interpolation_of( g_table )

  # create u data

  xmin = g_table[0,0]
  xmax = min(g_table[-1,0], cutoff)
  xs = linspace(xmin, xmax, n_points)
  result = zeros((n_points,2))
  firstgood = -1    # the point before which extrapolation is needed

  for i in range(n_points):
    x = xs[i]
    result[i,0] = x
    gi = g(x)
    if gi > g_zero_tol:
      result[i,1] = -Kb*T*log(gi)
      if firstgood == -1:
        firstgood = i

  # return u plus extraplation

  return (extrapolate_u( result, firstgood ), firstgood)


def create_md_potential( u, n_points, filename, sectionname ):
  uinterp = interpolation_of( u )
  duinterp = uinterp.derivative()
  x_start = u[0,0]
  x_end = u[-1,0]
  x2s = linspace(x_start**2, x_end**2, n_points)

  uu = zeros((n_points,2))
  ff = zeros((n_points,2))

  fout = open(filename, 'w');
  fout.write('\n');
  fout.write(sectionname + '\n');
  fout.write('N %d RSQ %f %f\n' % (n_points, x_start, x_end))
  fout.write('\n');

  for i in range(n_points):
    x = sqrt(x2s[i])
    ux = uinterp(x)
    fx = -duinterp(x)
    fout.write('%d %f %f %f\n' % (i+1, x, ux, fx))
    uu[i,0] = x
    uu[i,1] = ux
    ff[i,0] = x
    ff[i,1] = fx
```

```
  fout.close()

  return uu, ff

def prep_gofr( in_filename, out_filename, N_at, Lx, Ly, Lz, start_conf ):
  d = loadtxt( in_filename );
  N_confs = size(d,0)/N_at;
  fout = open( out_filename, 'w')
  for conf in range(start_conf, N_confs):
    s = "%d\n" % N_at
    fout.write(s)
    for i in range(N_at):
      j = conf*N_at+i
      fout.write("%d %d %f %f %f\n" % (i, 0, d[j,0], d[j,1], d[j,2]))
    fout.write("%f %f %f\n" % (Lx, Ly, Lz))
  fout.close()

def save_table( t, filename ):
  fout = open( filename, 'w' )
  for i in range(t.shape[0]):
    fout.write("%g %g\n" % (t[i,0], t[i,1]))
  fout.close()


dr_gofr = 0.02
n_points_md_u = 1500
n_points_u = 60
gofr_filename = 'gofr.txt'
N_at = 512
Lx = 33.403300
Ly = 33.422700
Lz = 31.935500
start_conf = 250
Kb = 0.0019858759538811696  # Kb in Kcal/mol/K
g_zero_tol = 1e-3
extrapolation_sample_region = 0.25 # Angstroms
cutoff = 12.0 # Angstroms

# clear steps folder
subprocess.call(['./clean'])

# copy initial data
shutil.copy('../FM-Tony-3rd/xxp_gofr_100.txt', './for_init_gofr_100k.txt')

# calc initial g(r)
subprocess.call(['./calc_gofr', 'for_init_gofr_100k.txt', str(dr_gofr)])

T = 100              # Kelvin

# calc target g(r)

g_target_table = loadtxt( gofr_filename );
save_table(g_target_table, 'steps/g_target.txt')

# smooth target g(r)

g_target_smooth_table = smooth_g( g_target_table )
save_table(g_target_smooth_table, 'steps/sm_g_target.txt')

# interpolate target g(r)

g_target_interp = interpolation_of( g_target_smooth_table )

# calc initial potential

u, firstgood = calc_u_from_gofr( g_target_smooth_table, T, n_points_u )

step = 0
relax_factor = 1.0

while True:
  # run md after saving tabulated potential
  uu, ff = create_md_potential( u, n_points_md_u, 'ch4/linear_100k.table', 'CH4CH4' )
  subprocess.call(['./makeruns'])

  # measure g(r)
  shutil.copy('ch4/100k.mdstats', 'steps/mdstats_' + ("%3.3d" % step) + '.txt')
  prep_gofr('ch4/100kcoords.txt', 'for_md_gofr_100k.txt', N_at, Lx, Ly, Lz, start_conf)
  subprocess.call(['./calc_gofr', 'for_md_gofr_100k.txt', str(dr_gofr)])

  # load measured g(r), smooth it and interpolate
  g_step_table = loadtxt( gofr_filename )
  g_step_smooth_table = smooth_g( g_step_table )
```

```
  g_step_interp = interpolation_of( g_step_smooth_table )

  # save u of step and resultig g(r) convergence plotting
  save_table(u, 'steps/u_' + ("%3.3d" % step) + '.txt')
  save_table(uu, 'steps/uu_' + ("%3.3d" % step) + '.txt')
  save_table(ff, 'steps/ff_' + ("%3.3d" % step) + '.txt')
  save_table(g_step_table, 'steps/g_' + ("%3.3d" % step) + '.txt')
  save_table(g_step_smooth_table, 'steps/sm_g_' + ("%3.3d" % step) + '.txt')

  # calc correction of potential
  n = u.shape[0]
  du = zeros((n,2))
  firstgood = -1
  for i in range(n):
    x = u[i,0]
    du[i,0] = x
    gti = g_target_interp(x)
    gsi = g_step_interp(x)
    if gti > g_zero_tol and gsi > g_zero_tol:
      du[i,1] = relax_factor*( Kb*T*log( gsi / gti ) )
      if firstgood == -1:
        firstgood = i
  save_table(du, 'steps/du_' + ("%3.3d" % step) + '.txt')

  # check for convergence
  max_rel_delta = 0.0
  max_delta = 0.0
  reldu = zeros((n,2))
  for i in range(n):
    delta = du[i,1]/u[i,1]
    x = u[i,0]
    reldu[i,0] = x
    reldu[i,1] = delta
    max_delta = max(abs(du[i,1]), max_delta)
    max_rel_delta = max(abs(delta), max_rel_delta)

  save_table(reldu, 'steps/reldu_' + ("%3.3d" % step) + '.txt')

  print
  print "STEP", step, "MAX DELTA", max_delta, "MAX REL DELTA", max_rel_delta
  print

  if max_delta <= 0.005:
    save_table(u, 'steps/u_final.txt')
    save_table(uu, 'steps/uu_final.txt')
    save_table(ff, 'steps/ff_final.txt')
    save_table(g_step_table, 'steps/g_final.txt')
    save_table(g_step_smooth_table, 'steps/sm_g_final.txt')
    break

  # apply correction
  for i in range(n):
    u[i,1] = u[i,1] + du[i,1]

  # extrapolate in excluded volume area

  try:
    unew = extrapolate_u( u, firstgood )
    ok = True
    for i in range(firstgood):
      if unew[i,1] < unew[i+1,1]:
        ok = False
        break
    if ok:
      u = unew
  except Exception:
    print "Fit failed, using old head"
    continue

  step = step + 1
```

C.2 Force Matching

Below is the listing of the Matlab code performing Force Matching method, for the methane system at 100K.

Listing C.2: Matlab code performing Force Matching method.

```
clear all; close all; clc;

root = '/home/thazir/Documents/ForceMatching/';
cpus = 19;

poolobj = parpool('local',cpus);

positions_path = [root 'bulk_CH4_data/T_100/out_xxp.dat'];
forces_path = [root 'bulk_CH4_data/T_100/out_ff_nb.dat'];
N_at = 2560;
L(1) = 33.4033; L(2)=  33.4227; L(3) = 31.9355;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BULK CH4 FULL WIDTH UNWEIGHTED
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

map = SimpleCH4Map();

[xxp, ff] = load_atomistic(positions_path, forces_path, N_at, 1);
xxp_CG = map.mapPositions(xxp);
ff_CG = map.mapForces(ff);
clear xxp;
clear ff;

x_start = 3.25;
x_end = 12.0;
dimension = 48;

N_confs = size(xxp_CG, 1);
N_mol = size(xxp_CG, 2);

basis = LinearBasis(x_start, x_end, dimension);
start = tic;
[G, f_tilde] = calc_G_and_f(xxp_CG, ff_CG, L, basis, poolobj);
[phi_linsim, chi_sq_linsim] = solve_normal(G, f_tilde, N_mol, N_confs);
duration_linsim = toc(start);

basis = CubicSplineBasis(x_start, x_end, 2*dimension);
start = tic;
[G, f_tilde] = calc_G_and_f(xxp_CG, ff_CG, L, basis, poolobj);
[phi_cubsim, chi_sq_cubsim] = solve_normal(G, f_tilde, N_mol, N_confs);
duration_cubsim = toc(start);

basis = LJBasis(x_start, x_end);
start = tic;
[G, f_tilde] = calc_G_and_f(xxp_CG, ff_CG, L, basis, poolobj);
[phi_ljsim, chi_sq_ljsim] = solve_normal(G, f_tilde, N_mol, N_confs);
duration_ljsim = toc(start);

f = @(theta)force_delta(xxp_CG, ff_CG, L, theta, 1, poolobj);
theta0 = [0.2716    1.3302    4.2739]; % from curve fitting
start = tic;
options = optimoptions('lsqnonlin','Display','iter','Jacobian','on');
options.MaxFunEvals = 3000;
[phi_morsim, resnorm_morsim , residual_morsim, exitflag, output] = lsqnonlin(f,theta0,[],[],
    options);
duration_morsim = toc(start);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SAVE RESULTS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fout = fopen('measure_results_bulk_unweighted_100.m', 'w');
fprintf(fout, 'x_start = %g ;\n', x_start);
fprintf(fout, 'x_end = %g ;\n', x_end);
fprintf(fout, 'dimension = %g ;\n', dimension);

fprintf(fout, 'phi_linsim = [');
for i = 1:length(phi_linsim)
    fprintf(fout, '%g ', phi_linsim(i));
end
```

```
fprintf(fout, '];\n');
fprintf(fout, 'chi_sq_linsim = %g;\n', chi_sq_linsim);
fprintf(fout, 'duration_linsim = %g;\n', duration_linsim);

fprintf(fout, 'phi_cubsim = [');
for i = 1:length(phi_cubsim)
    fprintf(fout, '%g ', phi_cubsim(i));
end
fprintf(fout, '];\n');
fprintf(fout, 'chi_sq_cubsim = %g;\n', chi_sq_cubsim);
fprintf(fout, 'duration_cubsim = %g;\n', duration_cubsim);

fprintf(fout, 'phi_ljsim = [');
for i = 1:length(phi_ljsim)
    fprintf(fout, '%g ', phi_ljsim(i));
end
fprintf(fout, '];\n');
fprintf(fout, 'chi_sq_ljsim = %g;\n', chi_sq_ljsim);
fprintf(fout, 'duration_ljsim = %g;\n', duration_ljsim);

fprintf(fout, 'phi_morsim = [');
for i = 1:length(phi_morsim)
    fprintf(fout, '%g ', phi_morsim(i));
end
fprintf(fout, '];\n');
fprintf(fout, 'resnorm_morsim = %g;\n', resnorm_morsim);
fprintf(fout, 'duration_morsim = %g;\n', duration_morsim);

fclose(fout);

delete(poolobj);
```

Below is the listing of load_atomistic.m, that loads and arranges the atomistic trajectory and forces.

Listing C.3: load_atomistic.m

```
function [ xxp, ff ] = load_atomistic( positions_path, forces_path, N_at, reduce_every_nth )

display('Loading forces');
temp = importdata(forces_path);
if reduce_every_nth ~= 1
    display('Reducing forces');
    temp = reduce_confs(temp, N_at, reduce_every_nth);
end

N_confs = size(temp,1)/N_at;

display('Reshaping forces');
ff = zeros(N_confs, N_at, 3);
for conf=1:N_confs
    at_from = (conf-1)*N_at+1;
    at_to = conf*N_at;
    ff(conf, :, :) = temp(at_from:at_to, :);
end
display('Reshaped forces');

display('Loading positions');
temp = importdata(positions_path);
if reduce_every_nth ~= 1
    display('Reducing positions');
    temp = reduce_confs(importdata(positions_path), N_at, reduce_every_nth);
end

display('Reshaping positions');
xxp = zeros(N_confs, N_at, 3);
for conf=1:N_confs
    at_from = (conf-1)*N_at+1;
    at_to = conf*N_at;
    xxp(conf, :, :) = temp(at_from:at_to, :);
end
display('Reshaped positions');

end
```

Below is the listing of SimpleCH4Map.m, that maps the atomistic to the CG coor-

dinates.

Listing C.4: SimpleCH4Map.m

```
classdef SimpleCH4Map

    properties
        m_C
        m_H
        m_Tot
    end

    methods

        function obj = SimpleCH4Map()
            obj.m_C = 12;
            obj.m_H = 1;
            obj.m_Tot = obj.m_C + 4*obj.m_H;
        end

        function xxp_CG = mapPositions(obj, xxp)
            N_confs = size(xxp,1);
            N_at = size(xxp,2);
            N_mol = N_at/5;
            xxp_CG = zeros(N_confs, N_mol, 3);
            for conf=1:N_confs
                for mol_idx=1:N_mol
                    j = (mol_idx-1)*5+1;
                    xxp_CG(conf, mol_idx, :) = (xxp(conf, j, :)*obj.m_C + xxp(conf, j+1, :)*obj
    .m_H + ...
                        xxp(conf, j+2, :)*obj.m_H + xxp(conf, j+3, :)*obj.m_H + ...
                        xxp(conf, j+4, :)*obj.m_H)/obj.m_Tot ;
                end
            end
        end

        function ff_CG = mapForces(obj, ff)
            N_confs = size(ff,1);
            N_at = size(ff,2);
            N_mol = N_at/5;
            ff_CG = zeros(N_confs, N_mol, 3);
            for conf=1:N_confs
                for mol_idx=1:N_mol
                    j = (mol_idx-1)*5+1;
                    ff_CG(conf, mol_idx, :) = ff(conf, j, :) + ff(conf, j+1, :) + ...
                        ff(conf, j+2, :) + ff(conf, j+3, :) + ff(conf, j+4, :) ;
                end
            end
        end

    end

end
```

Below is the listing of LinearBasis.m, the linear spline basis used in the FM method.

Listing C.5: LinearBasis.m

```
classdef LinearBasis < Basis

    properties
        dx
        dxi
        x_discr
    end

    methods
        function obj = LinearBasis(x_start, x_end, dimension)
            obj@Basis(x_start, x_end, dimension);
            obj.title = 'Linear';
            obj.dx = (obj.x_end-obj.x_start)/(dimension-1);
            obj.dxi = 1.0/obj.dx;
            obj.x_discr = obj.x_start: obj.dx : obj.x_end;
        end

        function val = fElementValues(obj, xs)
            val = zeros(length(xs), obj.dimension);

            for xi=1:size(xs,1)
```

```
                x = xs(xi);

                if x >= obj.x_start && x <= obj.x_end
                    for d=1:obj.dimension
                        if x <= obj.x_discr(d) && d > 1 && obj.x_discr(d-1) < x
                            val(xi,d) = (x - obj.x_discr(d-1)) * obj.dxi ;
                        elseif x > obj.x_discr(d) && d < obj.dimension && x <= obj.x_discr(d+1)
                            val(xi,d) = (obj.x_discr(d+1) - x) * obj.dxi;
                        else
                            val(xi,d) = 0;
                        end
                    end
                else
                    for d=1:obj.dimension
                        val(xi,d) = 0;
                    end
                end
            end
        end

        function val = uElementValues(obj, xs)
            val = zeros(length(xs), obj.dimension);

            for xi=1:size(xs,1)
                x = xs(xi);

                if x >= obj.x_start && x <= obj.x_end
                    for d=1:obj.dimension
                        l = 0;
                        r = 0;
                        b = min(obj.x_discr(d), x);

                        if d > 1
                            a = min(obj.x_discr(d-1), x);
                            l = - (0.5*(b*b-a*a) - obj.x_discr(d-1)*(b-a)) * obj.dxi;
                        end

                        if d < obj.dimension
                            g = min(obj.x_discr(d+1),x);
                            r = - (obj.x_discr(d+1)*(g-b) - 0.5*(g*g-b*b)) * obj.dxi;
                        end

                        val(xi,d) = l + r;
                    end
                else
                    for d=1:obj.dimension
                        val(xi,d) = 0;
                    end
                end

            end
        end

    end

end
```

Below is the listing of Basis.m, the base class of all basis functions used in the FM method.

Listing C.6: Basis.m

```
classdef Basis

    properties
        x_start
        x_end
        dimension
        title
    end

    methods

        function obj = Basis(x_start, x_end, dimension)
            if ceil(dimension) ~= floor(dimension)
                error('dimension should be an integer');
            end
            obj.dimension = dimension;
            obj.x_start = x_start;
```

```
            obj.x_end = x_end;
            obj.title = 'Basis';
        end

        function val = fValue(obj, x, phi)
            val = obj.fElementValues(x) * phi;
        end

        function val = uValue(obj, x, phi)
            val = obj.uElementValues(x) * phi;
        end

        function [a,b,c] = extrapolate_u(obj, phi, estart, eend, nfit)
            xfit = zeros(1,nfit);
            yfit = zeros(1,nfit);
            for i=1:nfit
                xfit(i) = estart + (eend-estart)*i/nfit;
                yfit(i) = obj.uValue(xfit(i),phi);
            end
            f = @(q,x)q(1)*exp(-q(2)*x)+q(3);
            options = optimset('MaxFunEvals',1e5);
            q0 = [1 1 0];
            q = lsqcurvefit(f,q0,xfit,yfit,[],[],options);
            a = q(1);
            b = q(2);
            c = q(3);
        end

        function [a,b,c] = extrapolate_f(obj, phi, estart, eend, nfit)
            xfit = zeros(1,nfit);
            yfit = zeros(1,nfit);
            for i=1:nfit
                xfit(i) = estart + (eend-estart)*i/nfit;
                yfit(i) = obj.fValue(xfit(i),phi);
            end
            f = @(q,x)q(1)*exp(-q(2)*x)+q(3);
            options = optimset('MaxFunEvals',1e5);
            q0 = [1 1 0];
            q = lsqcurvefit(f,q0,xfit,yfit,[],[],options);
            a = q(1);
            b = q(2);
            c = q(3);
        end

    end

    methods (Abstract)
        val = fElementValues(obj, xs);
        val = uElementValues(obj, xs);
    end

end
```

Below is the listing of calc_G_and_f.m, the function that calculates the $\underline{\boldsymbol{\mathcal{G}}}$ matrix (equation 2.19) and the $\underline{\tilde{\boldsymbol{f}}}$ vector (equation) of the FM method in parallel.

Listing C.7: calc_G_and_f.m

```
function [ GG, f_tilde ] = calc_G_and_f( xxp_CG, ff_CG, L, basis, poolobj)

N_confs = size(xxp_CG, 1);
N_mol = size(xxp_CG, 2);

tic;

G = zeros(N_confs, 3*N_mol, basis.dimension);

start = tic;

% at least 100K data for each worker
confs_per_batch = ceil(10^5/(3*N_mol*basis.dimension));
display(['Calculating batches of ' num2str(confs_per_batch) ' confs']);

if isequal(poolobj, false)
    % serial execution, no parallel pool available

    confs_done = 0;
    current_batch = 1;
```

```
    while confs_done < N_confs
        conf_from = (current_batch-1)*confs_per_batch+1;
        conf_to = min(current_batch*confs_per_batch, N_confs);

        xxps = xxp_CG(conf_from:conf_to, 1:N_mol, 1:3);
        G(conf_from:conf_to,:,:) = calc_G_confs(xxps, L, basis);

        current_batch = current_batch + 1;
        confs_done = confs_done + conf_to - conf_from + 1;

        duration = toc(start) / 60.0;
        speed = duration/confs_done;
        display(['Did ' num2str(confs_done) ' configurations, speed = ' ...
            num2str(speed) ' mins/conf = ' num2str(1.0/speed) ' confs/min']);
    end

else
    % parallel execution

    % send initial conf batches to workers

    current_batch = 1;

    for worker = 1:poolobj.NumWorkers
        conf_from = (current_batch-1)*confs_per_batch+1;
        conf_to = min(current_batch*confs_per_batch, N_confs);

        if conf_from <= N_confs
            xxps = xxp_CG(conf_from:conf_to, 1:N_mol, 1:3);
            %h(worker) = parfeval(poolobj, @calc_G_confs, 1, xxps, L, basis);
            h(worker) = parfeval(poolobj, @calc_G_confs, 1, xxps, L, basis);

            batch(worker) = current_batch;
            current_batch = current_batch + 1;
        end
    end

    % receive results and send more

    confs_done = 0;

    while confs_done < N_confs

        % receive results

        [worker, result] = fetchNext(h);

        worker_batch = batch(worker);
        conf_from = (worker_batch-1)*confs_per_batch+1;
        conf_to = min(worker_batch*confs_per_batch, N_confs);

        G(conf_from:conf_to,:,:) = result;
        confs_done = confs_done + conf_to - conf_from + 1;

        duration = toc(start) / 60.0;
        speed = duration/confs_done;
        display(['Did ' num2str(confs_done) ' configurations, speed = ' ...
            num2str(speed) ' mins/conf = ' num2str(1.0/speed) ' confs/min']);

        % send more

        conf_from = (current_batch-1)*confs_per_batch+1;
        conf_to = min(current_batch*confs_per_batch, N_confs);

        if conf_from <= N_confs
            xxps = xxp_CG(conf_from:conf_to, 1:N_mol, 1:3);
            h(worker) = parfeval(poolobj, @calc_G_confs, 1, xxps, L, basis);

            batch(worker) = current_batch;
            current_batch = current_batch+1;
        end

    end

end

duration = toc / 60.0;
display(['mins / conf = ' num2str(duration/4)]);


GG = zeros(3*N_mol*N_confs, basis.dimension);
for conf=1:N_confs
    rfrom = (conf-1)*3*N_mol+1;
```

```
    rto = conf*3*N_mol;
    GG(rfrom:rto, :) = G(conf,:,:);
end

% Reformulate ff_CG to f^{tilde}_underscore which is a column vector of
% dimension 3*N_mol* N_confs  (we don't have electrostatics so
%                                          f^{tilde} == f )

display('Calculating f tilde');

f_tilde = zeros(3 * N_mol * N_confs, 1);

% NAIVE. Might use reshape() for speed...
for conf = 1:N_confs

    for i = 1:N_mol
        idx = (conf-1) * (3 * N_mol) + (i-1) * 3 + 1;
        f_tilde(idx)   = ff_CG(conf, i, 1);
        f_tilde(idx+1) = ff_CG(conf, i, 2);
        f_tilde(idx+2) = ff_CG(conf, i, 3);
    end
end

end
```

Below is the listing of calc_G_confs.m, the function that calculates the part of the $\underline{\underline{\mathcal{G}}}$ matrix (equation 2.19) that corresponds to a few atomistic configurations of the FM method.

Listing C.8: calc_G_confs.m

```
function [ G_confs ] = calc_G_confs( xxps, L, basis )

N_confs = size(xxps,1);
N_mol = size(xxps,2);
G_confs = zeros(N_confs, 3*N_mol, basis.dimension);

for conf=1:size(xxps,1)
    xxp(:,:) = xxps(conf,:,:);
    G_confs(conf,:,:) = calc_G_conf_b(xxp, L, basis);
end

end
```

Below is the listing of calc_G_conf_b.m, the function that calculates the part of the $\underline{\underline{\mathcal{G}}}$ matrix (equation 2.19) that corresponds to a single atomistic configuration of the FM method.

Listing C.9: calc_G_conf_b.m

```
function [ G_conf ] = calc_G_conf_b( xxp, L, basis )

N_mol = size(xxp,1);
gradij = zeros(1, 3);
L_inv = 1.0 ./ L;
G_conf = zeros(3*N_mol, basis.dimension);

for I = 1 : N_mol - 1
    for J = I+1 : N_mol
        % vector distance
        gradij(:) = xxp(I,:) - xxp(J,:);
        % minimum image convention
        gradij(1:3) = gradij(1:3) - L(1:3) .* round( gradij(1:3) .* L_inv(1:3) ) ;
        % metric distance
        distij = sqrt( gradij(1)*gradij(1) + gradij(2)*gradij(2) + gradij(3)*gradij(3) );
        % gradient of vector distance Ri-Rj
        % (note that grad(unitary(Rj-Ri)) = - grad(unitary(Ri-Rj))
        gradij(1:3) = gradij(1:3) / distij;

        % then calculate fzid for all distances and d's

        f_zid = basis.fElementValues(distij);
```

```
        % G is 3 * N * N_confs x d
        % G_conf is 3 * N x d

        %find index and symmetric index
        idx_x = 3*(I-1)+1;       %1-st mol in rows 1-3, 2-nd in rows 4-6 etc...
        idx_sym_x = 3*(J-1)+1;  %1-st mol in rows 1-3, 2-nd in rows 4-6 etc...

        %calc G_{I;D}
        %=============

        G_conf(idx_x, :) = G_conf(idx_x, :) + f_zid(1,:) * gradij(1);
        G_conf(idx_x+1, :) = G_conf(idx_x+1, :) + f_zid(1,:) * gradij(2);
        G_conf(idx_x+2, :) = G_conf(idx_x+2, :) + f_zid(1,:) * gradij(3);

        G_conf(idx_sym_x, :) = G_conf(idx_sym_x, :) - f_zid(1,:) * gradij(1);
        G_conf(idx_sym_x+1, :) = G_conf(idx_sym_x+1, :) - f_zid(1,:) * gradij(2);
        G_conf(idx_sym_x+2, :) = G_conf(idx_sym_x+2, :) - f_zid(1,:) * gradij(3);
    end
end

end
```

C.3 Relative Entropy

Below is the listing of the main Python script that performs the RE iterations, for the methane system at 100K.

Listing C.10: Main Python script that performs the RE iterations.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from dbi import dbi_from_g_smoothed
from extrapolate import extrapolate_u_nb_plaw
from LinearBasis import LinearBasis
from phi_from_u import phi_from_u
from sgolay import savitzky_golay
from f_from_u import f_from_u
from create_md_potential_file import create_md_potential_file
from calc_jac_hess import calc_jac
from calc_jac_hess import calc_jac_hess
import subprocess
from cg_steps import cg_steps
import os
import shutil

def ak(k):
  a = 1.0
  a_cap = 0.0
  p = 1.0
  return a / (a_cap + k)**p

def flatten_pk(arr):
  result = np.zeros((arr.size))
  i = 0
  while i < arr.size:
    result[i] = arr[i,0]
    i += 1
  return result

def flatten_jac(arr):
  result = np.zeros((arr.size))
  i = 0
  while i < arr.size:
    result[i] = arr[i,0]
    i += 1
  return result

# clean output
subprocess.call(['./clean_saved_data'])

# init params
```

```
Kb = 0.00198588; # kcal/(mol K)
T = 100;         # K
beta = 1.0/(Kb * T)
g_tol = 1e-2
use_dpi = 300
md_xstart = 0.5
npoints = 1000

jac_cpus = 6
md_cpus = 6

# init basis
x_start = 3.34
x_end = 12.0
dimension = 80
lb = LinearBasis(x_start, x_end, dimension)


if not os.path.isfile('phi_start.txt'):
  if not os.path.isfile('u_start.txt'):
    # load g(r)
    g = np.loadtxt('g_of_r.txt')
    g[:,0] = g[:,0]

    # create dbi
    u0 = dbi_from_g_smoothed(g, g_tol, Kb, T);

    # extrapolate beginning
    u = extrapolate_u_nb_plaw( u0, md_xstart, npoints )
    np.savetxt('saved_data/nosteps/u_dbi.txt', u)
  else:
    u = np.loadtxt('u_start.txt')

  # sample basis region from dbi
  n_fit = 400
  u_fit = np.zeros((n_fit, 2))
  x_fit = np.linspace(x_start, x_end, n_fit)
  yy = interp1d(u[:,0], u[:,1])
  i = 0
  while i < n_fit:
    u_fit[i,0] = x_fit[i]
    u_fit[i,1] = yy( x_fit[i] )
    i += 1

  # find phi to reproduce dbi from basis
  phi = phi_from_u(u_fit, lb)
else:
  phi = np.loadtxt('phi_start.txt')
  print "STARTING FROM PHI", phi

# create basis potential and plot it vs input to verify good phis
u_basis = lb.create_u_table(phi, npoints)

u_md = lb.create_md_potential(phi, md_xstart, npoints)

smooth_npoints = 21
u_md[:,1] = savitzky_golay(u_md[:,1], smooth_npoints, 3)


# calculate force

f_md = f_from_u(u_md, 1e-4)

istart = 0
while u_md[istart,0] < 3.0:
  istart += 1

# create md potential
create_md_potential_file(u_md, f_md, 2000, 'md_run/lj.table','AA');

positions_path = 'coords_angstrom.txt';
nat = 512
L = np.zeros((3))
L[0] = 33.403
L[1] = 33.422
L[2] = 31.935
cutoff = 12.0

# calculate the fine part of the jacobian
jac_fine = calc_jac(positions_path, nat, L, cutoff, x_start, x_end, dimension, jac_cpus)
np.savetxt('saved_data/nosteps/jacobian_fine.txt', jac_fine)
```

```
step = 1

while step < 100:
  # save potential and force
    np.savetxt('saved_data/steps/u_md_%03d.txt' % step, u_md)
    np.savetxt('saved_data/steps/f_md_%03d.txt' % step, f_md)

  # run md
  cl = ['./makerun', str(md_cpus)]
  print cl
  subprocess.call(cl)

  # calc jabobian and hessian
  (jac_cg, hess) = calc_jac_hess('md_run/out_coords_plain.txt', nat, L, cutoff, x_start, x_end,
      dimension, jac_cpus)
  shutil.copyfile('md_run/out_coords_plain.txt', 'saved_data/steps/traj_%03d.txt' % step)
  shutil.copyfile('out_jac.txt', 'saved_data/steps/calc_jac_%03d.txt' % step)

    jac = beta * (jac_fine - jac_cg)
    hess = beta * beta * hess

  # calc pk
    p0 = np.asmatrix(np.zeros(jac.shape))
    pk = cg_steps(hess, jac, p0, 70, 1e-4)
  pk = flatten_pk(pk)

  # save step data
    np.savetxt('saved_data/steps/jac_%03d.txt' % step, jac)
    np.savetxt('saved_data/steps/hess_%03d.txt' % step, hess)
    np.savetxt('saved_data/steps/pk_%03d.txt' % step, pk)
    np.savetxt('saved_data/steps/phi_%03d.txt' % step, phi)

  # calc phi
    phi = phi - ak(step) * pk
    #phi = phi - 0.5 * pk
  jac = flatten_jac(jac)

  print 'phi_new', phi.shape
    np.savetxt('saved_data/steps/phi_new_%03d.txt' % step, phi)

  # increase step
  step += 1

  # create new md potential and force
  u_md = lb.create_md_potential(phi, md_xstart, npoints)
  u_md[:,1] = savitzky_golay(u_md[:,1], smooth_npoints, 3)
  f_md = f_from_u(u_md, 1e-4)
  create_md_potential_file(u_md, f_md, 2000, 'md_run/lj.table','AA')
```

Below is the listing of the header of the Jacobian calculation C++ class. This is the calculation of a single configuration contribution to the Jacobian.

Listing C.11: Header of the Jacobian calculation C++ class.

```
#ifndef SRC_JACOBIAN_H_
#define SRC_JACOBIAN_H_

#include <eigen3/Eigen/Dense>
class Basis;

namespace re
{

Eigen::VectorXd JacobianConf(const Eigen::MatrixXd &xxp, const Eigen::Vector3d &L, const double
    &cutoff, const Basis *basis);
void JacobianHessianConf(const Eigen::MatrixXd &xxp, const Eigen::Vector3d &L, const double &
    cutoff, const Basis *basis, Eigen::VectorXd &jac, Eigen::MatrixXd &hes_a, Eigen::MatrixXd
    &hes_b, Eigen::MatrixXd &hes_c);

}

#endif /* SRC_JACOBIAN_H_ */
```

Below is the listing of the code of the Jacobian calculation C++ class. This is the calculation of a single configuration contribution to the Jacobian.

Listing C.12: Code of the Jacobian calculation C++ class.

```
#include <eigen3/Eigen/Dense>
#include "Basis.h"
#include "Common.h"

using namespace Eigen;

namespace re
{

VectorXd JacobianConf(const MatrixXd &xxp, const Vector3d &L, const double &cutoff, const Basis
     *basis)
{
  unsigned nmol = xxp.rows();
  VectorXd jac( basis->dimension() );
  MatrixXd d_jac;
  jac = VectorXd::Zero( basis->dimension() );

  Vector3d x1;
  Vector3d x2;
  double distij;
  VectorXd dist_vec(1);

  for (unsigned i=0; i<nmol-1; ++i) {
    for (unsigned j=i+1; j<nmol; ++j) {
      x1 = xxp.row(i).transpose();
      x2 = xxp.row(j).transpose();
      distij = periodicDistance(x1,x2,L);

      if (distij <= cutoff) {
        dist_vec(0) = distij;
        basis->uElementValues( dist_vec, d_jac );
            jac += d_jac.transpose();
      }
    }
  }

  return jac;
}


void JacobianHessianConf(const MatrixXd &xxp, const Vector3d &L, const double &cutoff, const
    Basis *basis, VectorXd &jac, MatrixXd &hes_a, MatrixXd &hes_b, MatrixXd &hes_c)
{
  unsigned nmol = xxp.rows();
  jac = VectorXd::Zero( basis->dimension() );
  hes_a = MatrixXd::Zero( basis->dimension(), basis->dimension());
  hes_b = MatrixXd::Zero( basis->dimension(), basis->dimension());
  hes_c = MatrixXd::Zero( basis->dimension(), basis->dimension());

  MatrixXd du_dgamma;

  Vector3d x1;
  Vector3d x2;
  double distij;
  VectorXd dist_vec(1);

  for (unsigned i=0; i<nmol-1; ++i) {
    for (unsigned j=i+1; j<nmol; ++j) {
      x1 = xxp.row(i).transpose();
      x2 = xxp.row(j).transpose();
      distij = periodicDistance(x1,x2,L);

      if (distij <= cutoff) {
        dist_vec(0) = distij;
        basis->uElementValues( dist_vec, du_dgamma);
            jac += du_dgamma.transpose();
            for (unsigned hi=0; hi<basis->dimension(); ++hi) {
              for (unsigned hj=0; hj<basis->dimension(); ++hj) {
            hes_a( hi, hj ) = hes_a( hi, hj ) + du_dgamma( hi ) * du_dgamma( hj );
            hes_b( hi, hj ) = hes_b( hi, hj ) + du_dgamma( hi ) ;
            hes_c( hi, hj ) = hes_c( hi, hj ) + du_dgamma( hj );
              }
            }
      }
    }
  }
}


}
```

Below is the listing of the header of the trajectory Jacobian calculation C++ class.

Listing C.13: Header of the trajectory Jacobian calculation C++ class.

```
#ifndef SRC_JACOBIANTRAJ_H_
#define SRC_JACOBIANTRAJ_H_

#include <eigen3/Eigen/Dense>
#include <mpi.h>

namespace re
{

void JacobianTraj(const Eigen::MatrixXd &traj, const unsigned &n_ats, const Eigen::Vector3d & L
    , const double &cutoff, const re::Basis *basis, Eigen::VectorXd &result_jac, int root,
    MPI_Comm at_comm);
void JacobianHessianTraj(const Eigen::MatrixXd &traj, const unsigned &n_ats, const Eigen::
    Vector3d & L, const double &cutoff, const re::Basis *basis, Eigen::VectorXd &result_jac,
    Eigen::MatrixXd &result_hess, int root, MPI_Comm at_comm);

}

#endif /* SRC_JACOBIANTRAJ_H_ */
```

Below is the listing of the code of the trajectory Jacobian calculation C++ class.

Listing C.14: Code of the trajectory Jacobian calculation C++ class.

```
#include <eigen3/Eigen/Dense>
#include <mpi.h>
#include <vector>

#include "Basis.h"
#include "my_mpi.h"
#include "Common.h"
#include "Jacobian.h"

#include <iostream>

using namespace std;

using namespace Eigen;

namespace re
{

void JacobianTraj(const MatrixXd &traj, const unsigned &n_ats, const Vector3d & L, const double
    &cutoff, const re::Basis *basis, VectorXd &result_jac, int root, MPI_Comm at_comm) {
  MPI_Comm comm;
  int my_rank, num_ranks;

  MPI_Comm_dup(at_comm, &comm);
  MPI_Comm_rank(comm, &my_rank);
  MPI_Comm_size(comm, &num_ranks);

  if (num_ranks < 2)
    throw "Need at least 2 ranks, a dispatcher and a worker";

  unsigned use_n_ats = n_ats;
  Vector3d use_L = L;
  double use_cutoff = cutoff;
  re::Basis *use_basis = (re::Basis*)basis;

  MPI_Bcast(&use_n_ats, 1, MPI_UNSIGNED, root, comm);
  MPI_Bcast(use_L.data(), 3, MPI_DOUBLE, root, comm);
  MPI_Bcast(&use_cutoff, 1, MPI_DOUBLE, root, comm);
  bcast_basis(&use_basis, root, comm);

  result_jac = VectorXd::Zero(use_basis->dimension());

  if (my_rank == root) {
    if (traj.rows()%use_n_ats != 0)
      throw "Matrix size incompatible to number of atoms";

    vector<char> mpi_buffer;
    int mpi_buffer_size;

    mpi_buffer_size = num_ranks*use_n_ats*3*sizeof(double) + MPI_BSEND_OVERHEAD;
    mpi_buffer.resize(mpi_buffer_size);
```

```
    MPI_Buffer_attach(mpi_buffer.data(), mpi_buffer_size);

    unsigned num_confs = traj.rows() / use_n_ats;
    unsigned confs_sent = 0;
    unsigned confs_received = 0;
    bool waiting_result[num_ranks];
    int rank;

    for (int i=0; i<num_ranks; ++i)
      waiting_result[i] = false;

    while (confs_sent < num_confs || confs_received < num_confs) {
      rank = 0;
      while (confs_sent < num_confs && rank < num_ranks) {
        if (rank != root && waiting_result[rank] == false) {
          unsigned start_row = confs_sent * use_n_ats;
          MatrixXd conf = traj.block(start_row, 0, use_n_ats, 3);
          MPI_Bsend(conf.data(), conf.size(), MPI_DOUBLE, rank, CALC_CONF_JAC_TAG, comm);
          ++confs_sent;
          waiting_result[rank] = true;
        }
        ++rank;
      }

      rank = 0;
      while (confs_received < num_confs && rank < num_ranks) {
        if (rank != root && waiting_result[rank] == true) {
          int flag;
          MPI_Status status;
          MPI_Iprobe(rank, RESULT_CONF_JAC_TAG, comm, &flag, &status);
          if (flag) {
            VectorXd jac(basis->dimension());
            MPI_Recv(jac.data(), jac.size(), MPI_DOUBLE, rank, RESULT_CONF_JAC_TAG, comm, &
    status);
            result_jac += jac;
            waiting_result[rank] = false;
            ++confs_received;
          }
        }
        ++rank;
      }

      MySleep(10);
    }

    for (rank=0; rank<num_ranks; ++rank) {
      if (rank != root)
        MPI_Bsend(0, 0, MPI_CHAR, rank, ABORT_TAG, comm);
    }

    MPI_Buffer_detach(mpi_buffer.data(), &mpi_buffer_size);

    result_jac /= num_confs;
  }
  else {  // not root rank
    int flag;
    MPI_Status status;

    while (true) {
      MPI_Iprobe(root, CALC_CONF_JAC_TAG, comm, &flag, &status);
      if (flag) {
        MatrixXd conf(use_n_ats, 3);
        VectorXd jac(use_basis->dimension());
        MPI_Recv(conf.data(), conf.size(), MPI_DOUBLE, root, CALC_CONF_JAC_TAG, comm, &status);
        jac = JacobianConf(conf, use_L, use_cutoff, use_basis);
        MPI_Bsend(jac.data(), jac.size(), MPI_DOUBLE, root, RESULT_CONF_JAC_TAG, comm);
      }

      MPI_Iprobe(root, ABORT_TAG, comm, &flag, &status);
      if (flag)
        break;

      MySleep(10);
    }
  }

  MPI_Bcast(result_jac.data(), result_jac.size(), MPI_DOUBLE, root, comm);

  MPI_Comm_free(&comm);
}
```

```
void JacobianHessianTraj(const MatrixXd &traj, const unsigned &n_ats, const Vector3d & L, const
     double &cutoff, const re::Basis *basis, VectorXd &result_jac, MatrixXd &result_hess, int
    root, MPI_Comm at_comm) {
  MPI_Comm comm;
  int my_rank, num_ranks;

  MPI_Comm_dup(at_comm, &comm);
  MPI_Comm_rank(comm, &my_rank);
  MPI_Comm_size(comm, &num_ranks);

  if (num_ranks < 2)
    throw "Need at least 2 ranks, a dispatcher and a worker";

  unsigned use_n_ats = n_ats;
  Vector3d use_L = L;
  double use_cutoff = cutoff;
  re::Basis *use_basis = (re::Basis*)basis;

  MPI_Bcast(&use_n_ats, 1, MPI_UNSIGNED, root, comm);
  MPI_Bcast(use_L.data(), 3, MPI_DOUBLE, root, comm);
  MPI_Bcast(&use_cutoff, 1, MPI_DOUBLE, root, comm);
  bcast_basis(&use_basis, root, comm);

  unsigned dim = use_basis->dimension();

  result_jac = VectorXd::Zero(dim);
  result_hess = MatrixXd::Zero(dim,dim);

  if (my_rank == root) {
    if (traj.rows()%use_n_ats != 0)
      throw "Matrix size incompatible to number of atoms";

    MatrixXd result_hess_a, result_hess_b, result_hess_c;
    result_hess_a = MatrixXd::Zero(dim,dim);
    result_hess_b = MatrixXd::Zero(dim,dim);
    result_hess_c = MatrixXd::Zero(dim,dim);

    vector<char> mpi_buffer;
    int mpi_buffer_size;

    mpi_buffer_size = num_ranks*use_n_ats*3*sizeof(double) + MPI_BSEND_OVERHEAD;
    mpi_buffer.resize(mpi_buffer_size);

    MPI_Buffer_attach(mpi_buffer.data(), mpi_buffer_size);

    unsigned num_confs = traj.rows() / use_n_ats;
    unsigned confs_sent = 0;
    unsigned confs_received = 0;
    bool waiting_result[num_ranks];
    int rank;

    for (int i=0; i<num_ranks; ++i)
      waiting_result[i] = false;

    while (confs_sent < num_confs || confs_received < num_confs) {
      rank = 0;
      while (confs_sent < num_confs && rank < num_ranks) {
        if (rank != root && waiting_result[rank] == false) {
          unsigned start_row = confs_sent * use_n_ats;
          MatrixXd conf = traj.block(start_row, 0, use_n_ats, 3);
          MPI_Bsend(conf.data(), conf.size(), MPI_DOUBLE, rank, CALC_CONF_JAC_HES_TAG, comm);
          //cout << "sent" << endl;
          ++confs_sent;
          waiting_result[rank] = true;
        }
        ++rank;
      }

      rank = 0;
      while (confs_received < num_confs && rank < num_ranks) {
        if (rank != root && waiting_result[rank] == true) {
          int flag;
          MPI_Status status;
          MPI_Iprobe(rank, RESULT_CONF_JAC_HES_TAG, comm, &flag, &status);
          if (flag) {
            unsigned dim = use_basis->dimension();
            vector<char> data;
            data.resize((dim + dim*dim*3)*sizeof(double));

            MPI_Recv(data.data(), data.size(), MPI_DOUBLE, rank, RESULT_CONF_JAC_HES_TAG, comm,
    &status);

            VectorXd jac(dim);
```

```
            MatrixXd hess_a(dim,dim);
            MatrixXd hess_b(dim,dim);
            MatrixXd hess_c(dim,dim);
            int position=0;

            //cout << "root unpacking" << endl;

            MPI_Unpack(data.data(), data.size(), &position, jac.data(), jac.size(), MPI_DOUBLE,
     comm);
            MPI_Unpack(data.data(), data.size(), &position, hess_a.data(), hess_a.size(),
    MPI_DOUBLE, comm);
            MPI_Unpack(data.data(), data.size(), &position, hess_b.data(), hess_b.size(),
    MPI_DOUBLE, comm);
            MPI_Unpack(data.data(), data.size(), &position, hess_c.data(), hess_c.size(),
    MPI_DOUBLE, comm);

            //cout << "root unpacked" << endl;

            result_jac += jac;
            result_hess_a += hess_a;
            result_hess_b += hess_b;
            result_hess_c += hess_c;

            waiting_result[rank] = false;
            ++confs_received;

            //cout << "recv result" << endl;
          }
        }
        ++rank;
      }

      MySleep(10);
    }

    for (rank=0; rank<num_ranks; ++rank) {
      if (rank != root)
        MPI_Bsend(0, 0, MPI_CHAR, rank, ABORT_TAG, comm);
    }

    MPI_Buffer_detach(mpi_buffer.data(), &mpi_buffer_size);

    result_jac /= num_confs;
    result_hess_a /= num_confs;
    result_hess_b /= num_confs;
    result_hess_c /= num_confs;

    for (unsigned i=0; i<result_hess_b.rows(); ++i) {
      for (unsigned j=0; j<result_hess_b.cols(); ++j) {
        result_hess(i,j) = result_hess_a(i,j) - result_hess_b(i,j) * result_hess_c(i,j);
      }
    }
  }
  else {  // not root rank
    unsigned dim = use_basis->dimension();
    vector<char> mpi_buffer;
    int mpi_buffer_size = (dim + 3*dim*dim)*sizeof(double) + MPI_BSEND_OVERHEAD;
    mpi_buffer.resize(mpi_buffer_size);
    MPI_Buffer_attach(mpi_buffer.data(), mpi_buffer_size);

    int flag;
    MPI_Status status;

    while (true) {
      MPI_Iprobe(root, CALC_CONF_JAC_HES_TAG, comm, &flag, &status);
      if (flag) {
        MatrixXd conf(use_n_ats, 3);
        VectorXd jac;
        MatrixXd hess_a, hess_b, hess_c;

        MPI_Recv(conf.data(), conf.size(), MPI_DOUBLE, root, CALC_CONF_JAC_HES_TAG, comm, &
     status);

        //cout << "got" << endl;

        JacobianHessianConf(conf, use_L, use_cutoff, use_basis, jac, hess_a, hess_b, hess_c);

        //cout << "done" << endl;

        int sz, tot=0;

        MPI_Pack_size(jac.size(), MPI_DOUBLE, comm, &sz);
        tot += sz;
```

```
            MPI_Pack_size(hess_a.size(), MPI_DOUBLE, comm, &sz);
            tot += sz;
            MPI_Pack_size(hess_b.size(), MPI_DOUBLE, comm, &sz);
            tot += sz;
            MPI_Pack_size(hess_c.size(), MPI_DOUBLE, comm, &sz);
            tot += sz;

            vector<char> data;
            data.resize(tot);
            int position = 0;

            MPI_Pack(jac.data(), jac.size(), MPI_DOUBLE, data.data(), data.size(), &position, comm)
        ;
            MPI_Pack(hess_a.data(), hess_a.size(), MPI_DOUBLE, data.data(), data.size(), &position,
         comm);
            MPI_Pack(hess_b.data(), hess_b.size(), MPI_DOUBLE, data.data(), data.size(), &position,
         comm);
            MPI_Pack(hess_c.data(), hess_c.size(), MPI_DOUBLE, data.data(), data.size(), &position,
         comm);

            //cout << "packed" << endl;

            MPI_Bsend(data.data(), data.size(), MPI_CHAR, root, RESULT_CONF_JAC_HES_TAG, comm);

            //cout << "bsent result" << endl;
          }

          MPI_Iprobe(root, ABORT_TAG, comm, &flag, &status);
          if (flag)
            break;

          MySleep(10);
        }

        MPI_Buffer_detach(mpi_buffer.data(), &mpi_buffer_size);
      }

      MPI_Bcast(result_jac.data(), result_jac.size(), MPI_DOUBLE, root, comm);
      MPI_Bcast(result_hess.data(), result_hess.size(), MPI_DOUBLE, root, comm);

      MPI_Comm_free(&comm);
    }



    }
```

Below is the listing of the header of the linear spline basis C++ class.

Listing C.15: Header of the linear spline basis C++ class.

```
#ifndef LINEARBASIS_H_
#define LINEARBASIS_H_

#include "Basis.h"
#include <eigen3/Eigen/Dense>
#include <mpi.h>

namespace re {

class LinearBasis: public Basis {
protected:
  double m_dx, m_dxi;
  Eigen::VectorXd m_x_discr;
public:
  LinearBasis();
  LinearBasis(double x_start, double x_end, unsigned dimension);
  virtual ~LinearBasis();
  virtual void uElementValues(const Eigen::VectorXd &at_xs, Eigen::MatrixXd &ret_vals) const;
  const double & dx() const;
  const Eigen::VectorXd & x_discr() const;

  virtual int packSize(MPI_Comm comm) const;
  virtual void pack(void *outbuf, int outsize, int *position, MPI_Comm comm);
  virtual void unpack(void *inbuf, int insize, int *position, MPI_Comm comm);
};

} /* namespace re */

#endif /* LINEARBASIS_H_ */
```

Below is the listing of the code of the linear spline basis C++ class.

Listing C.16: Code of the linear spline basis C++ class.

```
#include "LinearBasis.h"

using namespace Eigen;

namespace re {

LinearBasis::LinearBasis() : Basis() {
  m_title = "LinearBasis";
}

LinearBasis::LinearBasis(double x_start, double x_end, unsigned dimension)
      : Basis(x_start, x_end, dimension) {
  m_title = "LinearBasis";
    m_dx = (m_x_end-m_x_start)/(m_dimension-1);
    m_dxi = 1.0/m_dx;
    m_x_discr.resize(m_dimension);
    unsigned i;
    for (i=0; i<m_dimension-1; ++i) {
      m_x_discr(i) = m_x_start + i*m_dx;
    }
    m_x_discr(i) = m_x_end;
}


LinearBasis::~LinearBasis() {
  // TODO Auto-generated destructor stub
}

void LinearBasis::uElementValues(const Eigen::VectorXd &at_xs, Eigen::MatrixXd &ret_vals) const
     {
    ret_vals.resize(at_xs.size(), m_dimension);
    // set to zero ?

    for (unsigned xi=0; xi < at_xs.size(); ++xi) {
        double x = at_xs(xi);

        if (x >= m_x_start && x <= m_x_end) {
            for (unsigned d=0; d<m_dimension; ++d) {
                if (x <= m_x_discr(d) && d > 0 && m_x_discr(d-1) <= x) {
                    // matlab code: val = (x - obj.x_discr(d-1)) / (obj.x_discr(d) - obj.
    x_discr(d-1)) ;
                    ret_vals(xi,d) = (x - m_x_discr(d-1)) * m_dxi ;
                }
                else if (x >= m_x_discr(d) && d < m_dimension-1 && x <= m_x_discr(d+1)) {
                    // matlab code: val = (obj.x_discr(d+1) - x) / (obj.x_discr(d+1) - obj.
    x_discr(d));
                    ret_vals(xi,d) = (m_x_discr(d+1) - x) * m_dxi;
                }
                else {
                    ret_vals(xi,d) = 0.0;
                }
            }
        }
        else {
            for (unsigned d=0; d < m_dimension; ++d) {
                ret_vals(xi,d) = 0.0;
            }
        }
    }
}

const double & LinearBasis::dx() const {
  return m_dx;
}

const Eigen::VectorXd & LinearBasis::x_discr() const {
  return m_x_discr;
}

int LinearBasis::packSize(MPI_Comm comm) const {
  int sz, tot=0;
  MPI_Pack_size(4,MPI_DOUBLE,comm,&sz);
  tot+=sz;
  MPI_Pack_size(2,MPI_UNSIGNED,comm,&sz);
  tot+=sz;
  MPI_Pack_size(m_x_discr.size(),MPI_DOUBLE,comm,&sz);
  tot+=sz;
  return tot;
}
```

```
void LinearBasis::pack(void *outbuf, int outsize, int *position, MPI_Comm comm) {
  unsigned x_discr_sz = m_x_discr.size();

  MPI_Pack(&m_x_start, 1, MPI_DOUBLE, outbuf, outsize, position, comm);
  MPI_Pack(&m_x_end, 1, MPI_DOUBLE, outbuf, outsize, position, comm);
  MPI_Pack(&m_dx, 1, MPI_DOUBLE, outbuf, outsize, position, comm);
  MPI_Pack(&m_dxi, 1, MPI_DOUBLE, outbuf, outsize, position, comm);
  MPI_Pack(&m_dimension, 1, MPI_UNSIGNED, outbuf, outsize, position, comm);
  MPI_Pack(&x_discr_sz, 1, MPI_UNSIGNED, outbuf, outsize, position, comm);
  MPI_Pack(m_x_discr.data(), m_x_discr.size(), MPI_DOUBLE, outbuf, outsize, position, comm);
}

void LinearBasis::unpack(void *inbuf, int insize, int *position, MPI_Comm comm) {
  MPI_Unpack(inbuf, insize, position, &m_x_start, 1, MPI_DOUBLE, comm);
  MPI_Unpack(inbuf, insize, position, &m_x_end, 1, MPI_DOUBLE, comm);
  MPI_Unpack(inbuf, insize, position, &m_dx, 1, MPI_DOUBLE, comm);
  MPI_Unpack(inbuf, insize, position, &m_dxi, 1, MPI_DOUBLE, comm);
  MPI_Unpack(inbuf, insize, position, &m_dimension, 1, MPI_UNSIGNED, comm);
  unsigned x_discr_sz;
  MPI_Unpack(inbuf, insize, position, &x_discr_sz, 1, MPI_UNSIGNED, comm);
  m_x_discr.resize( x_discr_sz );
  MPI_Unpack(inbuf, insize, position, m_x_discr.data(), x_discr_sz, MPI_DOUBLE, comm);
}


} /* namespace re */
```

Bibliography

[1] Mdpar, a parallel molecular dynamics simulator. `http://macomms.tem.uoc.gr/index.php/mdpar-description-menu`, 2016.

[2] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. Gromacs: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1–2:19 – 25, 2015.

[3] Tsourtis Anastasios. *Mathematical and Computational modelling of complex molecular systems at multiple scales*. PhD thesis, Department of Applied Mathematics, University of Crete, 2017.

[4] Hans C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of Chemical Physics*, 72(4):2384–2393, 1980.

[5] Charles Matthews Ben Leimkuhler. *Molecular Dynamics: With Deterministic and Stochastic Numerical Methods*. Interdisciplinary Applied Mathematics Vol. 39. Springer, 2015 edition, 2015.

[6] Ilias Bilionis and Nicholas Zabaras. A stochastic optimization approach to coarse-graining using a relative-entropy framework. *The Journal of Chemical Physics*, 138(4):–, 2013.

[7] Aviel Chaimovich and M. Scott Shell. Coarse-graining errors and numerical optimization using a relative entropy framework. *The Journal of Chemical Physics*, 134(9):094112, 2011.

[8] David Chandler. *Introduction to modern statistical mechanics*. Oxford University Press, 1987.

[9] Berend Smit Daan Frenkel. *Understanding molecular simulation: from algorithms to applications*. Computational science series 1. Academic Press, 2nd ed edition, 2002.

[10] K. E. Gubbins. *Theory of molecular fluids*. The International series of monographs on chemistry 9. Oxford University Press, 1985.

[11] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. `http://eigen.tuxfamily.org`, 2010.

[12] Vagelis Harmandaris, Evangelia Kalligiannaki, Markos A. Katsoulakis, and Petr Plecháč. Path-space variational inference for non-equilibrium coarse-grained systems. *Journal of Computational Physics*, 314:355–383, 2016.

[13] R.L. Henderson. A uniqueness theorem for fluid pair correlation functions. *Physics Letters A*, 49(3):197 – 198, 1974.

[14] William G. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Phys. Rev. A*, 31:1695–1697, Mar 1985.

[15] William G. Hoover. Constant-pressure equations of motion. *Phys. Rev. A*, 34:2499–2500, Sep 1986.

[16] Sergei Izvekov and Gregory A. Voth. Multiscale coarse graining of liquid-state systems. *The Journal of Chemical Physics*, 123(13):134105, 2005.

[17] E. Kalligiannaki, A. Chazirakis, A. Tsourtis, M.A. Katsoulakis, P. Plecháč, and V. Harmandaris. Parametrizing coarse grained models for molecular systems at equilibrium. *The European Physical Journal Special Topics*, 225(8):1347–1372, Oct 2016.

[18] Evangelia Kalligiannaki, Vagelis Harmandaris, Markos A. Katsoulakis, and Petr Plechac. The geometry of generalized force matching and related information metrics in coarse-graining of molecular systems. *The Journal of Chemical Physics*, 143(8), 2015.

[19] M. A. Katsoulakis and P. Plechac. Information-theoretic tools for parametrized coarse-graining of non-equilibrium extended systems. *J. Chem. Phys.*, 139:4852–4863, 2013.

[20] Andrew Leach. *Molecular modelling: principles and applications*. Prentice Hall, 2nd ed edition, 2001.

[21] Glenn J. Martyna, Douglas J. Tobias, and Michael L. Klein. Constant pressure molecular dynamics algorithms. *The Journal of Chemical Physics*, 101(5):4177–4189, 1994.

[22] W. G. Noid. Perspective: Coarse-grained models for biomolecular systems. *The Journal of Chemical Physics*, 139(9):090901, 2013.

[23] W. G. Noid, Jhih-Wei Chu, Gary S. Ayton, Vinod Krishna, Sergei Izvekov, Gregory A. Voth, Avisek Das, and Hans C. Andersen. The multiscale coarse-graining method. i. a rigorous bridge between atomistic and coarse-grained models. *The Journal of Chemical Physics*, 128(24):244114, 2008.

[24] W. G. Noid, Pu Liu, Yanting Wang, Jhih-Wei Chu, Gary S. Ayton, Sergei Izvekov, Hans C. Andersen, and Gregory A. Voth. The multiscale coarse-graining method. ii. numerical implementation for coarse-grained molecular models. *The Journal of Chemical Physics*, 128(24):244115, 2008.

[25] Shuichi Nosé. A unified formulation of the constant temperature molecular dynamics methods. *The Journal of Chemical Physics*, 81(1):511–519, 1984.

[26] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1):1–19, March 1995.

[27] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[28] M. Scott Shell. The relative entropy is fundamental to multiscale and inverse thermodynamic problems. *The Journal of Chemical Physics*, 129(14):–, 2008.

[29] A.K. Soper. Empirical potential monte carlo simulation of fluid structure. *Chemical Physics*, 202(2-3):295 – 306, 1996.

[30] W. Tschöp, K. Kremer, O. Hahn, J. Batoulis, and T. Bürger. Simulation of polymer melts. I. coarse-graining procedure for polycarbonates. *Acta Polym.*, 49:61, 1998.

[31] Mark E. Tuckerman. *Statistical Mechanics: Theory and Molecular Simulation*. Oxford Graduate Texts. Oxford University Press, USA, 2010.