OOPSE-3.0: An Object-Oriented Parallel Simulation Engine for Molecular Dynamics

Teng Lin, Christopher J. Fennell, Charles F. Vardeman II,
Kyle Daily, Yang Zheng, Matthew A. Meineke, and J. Daniel Gezelter
Department of Chemistry and Biochemistry
University of Notre Dame
Notre Dame, Indiana 46556

January 16, 2006

Preface

OOPSE is a new molecular dynamics simulation program which is capable of efficiently integrating equations of motion for atom types with orientational degrees of freedom (e.g. "sticky" atoms and point dipoles). Transition metals can also be simulated using the embedded atom method (EAM) potential included in the code. Parallel simulations are carried out using the force-based decomposition method. Simulations are specified using a very simple C-based meta-data language. A number of advanced integrators are included, and the basic integrator for orientational dynamics provides substantial improvements over older quaternion-based schemes.

Contents

Chapter 1

Introduction

There are a number of excellent molecular dynamics packages available to the chemical physics community.[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] All of these packages are stable, polished programs which solve many problems of interest. Most are now capable of performing molecular dynamics simulations on parallel computers. Some have source code which is freely available to the entire scientific community. Few, however, are capable of efficiently integrating the equations of motion for atom types with orientational degrees of freedom (e.g. point dipoles, and "sticky" atoms). And only one of the programs referenced can handle transition metal force fields like the Embedded Atom Method (EAM). The direction our research program has taken us now involves the use of atoms with orientational degrees of freedom as well as transition metals. Since these simulation methods may be of some use to other researchers, we have decided to release our program (and all related source code) to the scientific community.

This document communicates the algorithmic details of our program, which we have been calling the Object-Oriented Parallel Simulation Engine (i.e. OOPSE). We have structured this document to first discuss the underlying concepts in this simulation package (Sec. 2). The empirical energy functions implemented are discussed in Sec. 3. Sec. 4 describes the various Molecular Dynamics algorithms OOPSE implements in the integration of Hamilton's equations of motion. Program design considerations for parallel computing are presented in Sec. 9. Concluding remarks are presented in Sec. 10.

Chapter 2

Concepts & Files

A simulation in OOPSE is built using a few fundamental conceptual building blocks most of which are chemically intuitive. The basic unit of a simulation is an `atom`. The parameters describing an `atom` have been generalized to make it as flexible as possible; this means that in addition to translational degrees of freedom, `Atoms` may also have *orientational* degrees of freedom.

The fundamental (static) properties of `atoms` are defined by the `forceField` chosen for the simulation. The atomic properties specified by a `forceField` might include (but are not limited to) charge, σ and ϵ values for Lennard-Jones interactions, the strength of the dipole moment (μ), the mass, and the moments of inertia. Other more complicated properties of atoms might also be specified by the `forceField`.

`Atoms` can be grouped together in many ways. A `rigidBody` contains atoms that exert no forces on one another and which move as a single rigid unit. A `cutoffGroup` may contain atoms which function together as a (rigid *or* non-rigid) unit for potential energy calculations,

$$V_{ab} = s(r_{ab}) \sum_{i \in a} \sum_{j \in b} V_{ij}(r_{ij}) \tag{2.1}$$

Here, a and b are two `cutoffGroups` containing multiple atoms ($a = \{i\}$ and $b = \{j\}$). $s(r_{ab})$ is a generalized switching function which insures that the atoms in the two `cutoffGroups` are treated identically as the two groups enter or leave an interaction region.

`Atoms` may also be grouped in more traditional ways into `bonds`, `bends`, and `torsions`. These groupings allow the correct choice of interaction parameters for short-range interactions to be chosen from the definitions in the `forceField`.

All of these groups of `atoms` are brought together in the `molecule`, which is the fundamental structure for setting up and OOPSE simulation. `Molecules` contain lists of `atoms` followed by listings of the other atomic groupings (`bonds`, `bends`, `torsions`, `rigidBodies`, and `cutoffGroups`) which relate the atoms to one another.

Simulations often involve heterogeneous collections of molecules. To specify a mixture of `molecule` types, OOPSE uses `components`. Even simulations containing only one type of molecule must specify a single `component`.

Starting a simulation requires two types of information: *meta-data*, which describes the types of objects present in the simulation, and *configuration* information, which describes the initial state of these objects. The meta-data is given to OOPSE using a C-based syntax that is parsed at the beginning of the simulation. Configuration information is specified using an extended XYZ file format. Both the meta-data and configuration file formats are described in the following sections.

```
molecule{
  name = "Ar";
  atom[0]{
    type="Ar";
    position( 0.0, 0.0, 0.0 );
  }
}

component{
  type = "Ar";
  nMol = 108;
}

initialConfig = "./argon.in";

forceField = "LJ";
ensemble = "NVE"; // specify the simulation ensemble
dt = 1.0;         // the time step for integration
runTime = 1e3;    // the total simulation run time
sampleTime = 100; // trajectory file frequency
statusTime = 50;  // statistics file frequency
```

Scheme 2.1: An example showing a complete meta-data file.

2.1 Meta-data Files

OOPSE uses a C-based script syntax to parse the meta-data files at run time. These files allow the user to completely describe the system they wish to simulate, as well as tailor OOPSE's behavior during the simulation. Meta-data files are typically denoted with the extension `.md` (which can stand for Meta-Data or Molecular Dynamics or Molecule Definition depending on the user's mood). An example meta-data file is shown in Scheme 2.1.

Within the meta-data file it is necessary to provide a complete description of the molecule before it is actually placed in the simulation. OOPSE's meta-data syntax was originally developed with this goal in mind, and allows for the use of *include files* to specify all atoms in a molecular prototype, as well as any bonds, bends, or torsions. Include files allow the user to describe a molecular prototype once, then simply include it into each simulation containing that molecule. Returning to the example in Scheme 2.1, the include file's contents would be Scheme 2.2, and the new meta-data file would become Scheme 2.3.

2.2 Atoms, Molecules, and other ways of grouping atoms

As mentioned above, the fundamental unit for an OOPSE simulation is the `atom`. Atoms can be collected into secondary structures such as `rigidBodies`, `cutoffGroups`, or `molecules`. The `molecule` is a way for OOPSE to keep track of the atoms in a simulation in logical manner. Molecular units store the identities of all the atoms and rigid bodies associated with themselves, and they are responsible for the evaluation of their own internal interactions (*i.e.* bonds, bends, and torsions). Scheme 2.2 shows how one creates a molecule in an included meta-data file. The positions of the atoms given in the declaration are relative to the origin of the molecule, and the origin is used when creating a system containing the molecule.

```
molecule{
  name = "Ar";
  atom[0]{
    type="Ar";
    position( 0.0, 0.0, 0.0 );
  }
}
```

Scheme 2.2: An example molecule definition in an include file.

```
#include "argon.md"

component{
  type = "Ar";
  nMol = 108;
}

initialConfig = "./argon.in";

forceField = "LJ";
ensemble = "NVE";
dt = 1.0;
runTime = 1e3;
sampleTime = 100;
statusTime = 50;
```

Scheme 2.3: Revised meta-data example.

One of the features that sets OOPSE apart from most of the current molecular simulation packages is the ability to handle rigid body dynamics. Rigid bodies are non-spherical particles or collections of particles (e.g. C_{60}) that have a constant internal potential and move collectively.[11] They are not included in most simulation packages because of the algorithmic complexity involved in propagating orientational degrees of freedom. Integrators which propagate orientational motion with an acceptable level of energy conservation for molecular dynamics are relatively new inventions.

Moving a rigid body involves determination of both the force and torque applied by the surroundings, which directly affect the translational and rotational motion in turn. In order to accumulate the total force on a rigid body, the external forces and torques must first be calculated for all the internal particles. The total force on the rigid body is simply the sum of these external forces. Accumulation of the total torque on the rigid body is more complex than the force because the torque is applied to the center of mass of the rigid body. The space-fixed torque on rigid body i is

$$\boldsymbol{\tau}_i = \sum_a \Big[(\mathbf{r}_{ia} - \mathbf{r}_i) \times \mathbf{f}_{ia} + \boldsymbol{\tau}_{ia} \Big], \tag{2.2}$$

where $\boldsymbol{\tau}_i$ and $\mathbf{r}_i$ are the torque on and position of the center of mass respectively, while $\mathbf{f}_{ia}$, $\mathbf{r}_{ia}$, and $\boldsymbol{\tau}_{ia}$ are the force on, position of, and torque on the component particles of the rigid body.

The summation of the total torque is done in the body fixed axis of each rigid body. In order to move between the space fixed and body fixed coordinate axes, parameters describing the orientation must be maintained for each rigid body. At a minimum, the rotation matrix (A) can be described by the three Euler angles (ϕ, θ, and ψ), where the elements of A are composed of trigonometric operations involving ϕ, θ, and ψ.[11] In order to avoid numerical instabilities inherent in using the Euler angles, the four parameter "quaternion" scheme is often used. The elements of A can be expressed as arithmetic operations involving the four quaternions (q_w, q_x, q_y, and q_z).[12] Use of quaternions also leads to performance enhancements, particularly for very small systems.[13]

Rather than use one of the previously stated methods, OOPSE utilizes a relatively new scheme that propagates the entire nine parameter rotation matrix. Further discussion on this choice can be found in Sec. 4.1. An example definition of a rigid body can be seen in Scheme 2.4.

2.3 Creating a Metadata File

The actual creation of a metadata file requires several key components. The first part of the file needs to be the declaration of all of the molecule prototypes used in the simulation. This is typically done through included metadata files. Only the molecules actually present in the simulation need to be declared; however, OOPSE allows for the declaration of more molecules than are needed. This gives the user the ability to build up a library of commonly used molecules into a single include file.

Once all prototypes are declared, the ordering of the rest of the script is less stringent. The molecular composition of the simulation is specified with `component` statements. Each different type of molecule present in the simulation is considered a separate component (an example is shown in Sch. 2.3). The component blocks tell OOPSE the number of molecules that will be in the simulation, and the order in which the components blocks are declared sets the ordering of the real atoms in the configuration file as well as in the output files. The remainder of the script then sets the various simulation parameters for the system of interest.

The required set of parameters that must be present in all simulations is given in Table 2.1. Since the user can use OOPSE to perform energy minimizations as well as molecular dynamics simulations, one of the `minimizer` or `ensemble` keywords must be present. The `ensemble` keyword is responsible for selecting the integration method used for the calculation of the equations of motion. An in depth discussion of the various methods available in OOPSE

```
molecule{
  name = "TIP3P";
  atom[0]{
    type = "O_TIP3P";
    position( 0.0, 0.0, -0.06556 );
  }
  atom[1]{
    type = "H_TIP3P";
    position( 0.0, 0.75695, 0.52032 );
  }
  atom[2]{
    type = "H_TIP3P";
    position( 0.0, -0.75695, 0.52032 );
  }

  rigidBody[0]{
    members(0, 1, 2);
  }

  cutoffGroup[0]{
    members(0, 1, 2);
  }
}
```

Scheme 2.4: A sample definition of a molecule containing a rigid body and a cutoff group

can be found in Sec. 4. The `minimizer` keyword selects which minimization method to use, and more details on the choices of minimizer parameters can be found in Sec. 6. The `forceField` statement is important for the selection of which forces will be used in the course of the simulation. OOPSE supports several force fields, as outlined in Sec. 3. The force fields are interchangeable between simulations, with the only requirement being that all atoms needed by the simulation are defined within the selected force field.

For molecular dynamics simulations, the time step between force evaluations is set with the `dt` parameter, and `runTime` will set the time length of the simulation. Note, that `runTime` is an absolute time, meaning if the simulation is started at t = 10.0 ns with a `runTime` of 25.0 ns, the simulation will only run for an additional 15.0 ns.

For energy minimizations, it is not necessary to specify `dt` or `runTime`.

The final required parameter is the `initialConfig` statement. This will set the initial coordinates for the system, as well as the initial time if the `useInitalTime` flag is set to `true`. The format of the file specified in `initialConfig`, is given in Sec. 2.4. Additional parameters are summarized in Table 2.2.

It is important to note the fundamental units in all files which are read and written by OOPSE. Energies are in kcal mol^{-1}, distances are in Å, times are in fs, translational velocities are in Å fs^{-1}, and masses are in amu. Orientational degrees of freedom are described using quaternions (unitless, but $q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1$), body-fixed angular momenta (amu Å^{2}radians fs^{-1}), and body-fixed moments of inertia (amu Å^2).

Table 2.1: Meta-data Keywords: Required Parameters

keyword	**units**	**use**	**remarks**
`forceField`	string	Sets the force field.	Possible force fields are DUFF, WATER, LJ, EAM, SC, and CLAY.
`component`		Defines the molecular components of the system	Every `.md` file must have a component block.
`initialConfig`	string	Sets the file containing the initial configuration.	Can point to any file containing the configuration in the correct order.
`minimizer`	string	Chooses a minimizer	Possible minimizers are SD and CG. Either `ensemble` or `minimizer` must be specified.
`ensemble`	string	Sets the ensemble.	Possible ensembles are NVE, NVT, NPTi, NPAT, NPTf, and NPTxyz. Either `ensemble` or `minimizer` must be specified.
`dt`	fs	Sets the time step.	Selection of `dt` should be small enough to sample the fastest motion of the simulation. (`dt` is required for molecular dynamics simulations)
`runTime`	fs	Sets the time at which the simulation should end.	This is an absolute time, and will end the simulation when the current time meets or exceeds the `runTime`. (`runTime` is required for molecular dynamics simulations)

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	units	use	remarks
forceFieldVariant	string	Sets the name of the variant of the force field.	EAM has three variants: u3, u6, and VC.
forceFieldFileName	string	Overrides the default force field file name	Each force field has a default file name, and this parameter can override the default file name for the chosen force field.
usePeriodicBoundaryConditions	logical	Turns periodic boundary conditions on/off.	Default is true.
orthoBoxTolerance	double		decides how orthogonal the periodic box must be before we can use cheaper box calculations
cutoffRadius	Å	Manually sets the cutoff radius	the default value is set by the cutoffPolicy
cutoffPolicy	string	one of mix, max, or traditional	the traditional cutoff policy is to set the cutoff radius for all atoms in the system to the same value (governed by the largest atom). mix and max are pair-dependent cutoff methods.
skinThickness	Å	thickness of the skin for the Verlet neighbor lists	defaults to 1 Å
switchingRadius	Å	Manually sets the inner radius for the switching function.	Defaults to 85 % of the cutoffRadius.
switchingFunctionType	string	cubic or fifth_order_polynomial	Default is cubic.
useInitialTime	logical	Sets whether the initial time is taken from the .in file.	Useful when recovering a simulation from a crashed processor. Default is false.
useInitialExtendedSystemState	logical	keep the extended system variables?	Should the extended variables (the thermostat and barostat) be kept from the input file?
sampleTime	fs	Sets the frequency at which the .dump file is written.	The default is equal to the runTime.
resetTime	fs	Sets the frequency at which the extended system variables are reset to zero	The default is to never reset these variables.
statusTime	fs	Sets the frequency at which the .stat file is written.	The default is equal to the sampleTime.

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	units	use	remarks
finalConfig	string	Sets the name of the final output file.	Useful when stringing simulations together. Defaults to the root name of the initial meta-data file but with an .eor extension.
compressDumpFile	logical		should the .dump file be compressed on the fly?
statFileFormat	string	columns to print in the .stat file where each column is separated by a pipe (|) symbol. Allowed column names are: TIME, TOTAL_ENERGY, POTENTIAL_ENERGY, KINETIC_ENERGY, TEMPERATURE, PRESSURE, VOLUME, CONSERVED_QUANTITY, TRANSLATIONAL_KINETIC, ROTATIONAL_KINETIC, LONG_RANGE_POTENTIAL, SHORT_RANGE_POTENTIAL, VANDERWAALS_POTENTIAL, ELECTROSTATIC_POTENTIAL, BOND_POTENTIAL, BEND_POTENTIAL, DIHEDRAL_POTENTIAL, IMPROPER_POTENTIAL, VRAW, VHARM, PRESSURE_TENSOR_X, PRESSURE_TENSOR_Y, PRESSURE_TENSOR_Z	(The default is the first eight of these columns in order.)
printPressureTensor	logical	sets whether OOPSE will print out the pressure tensor	can be useful for calculations of the bulk modulus
electrostaticSummationMethod	string	none, shifted_potential, shifted_force, or reaction_field	default is none.
electrostaticScreeningMethod	string	undamped or damped	default is undamped
dielectric	unitless	Sets the dielectric constant for reaction field.	If electrostaticSummationMethod is set to reaction_field, then dielectric must be set.
dampingAlpha	Å^{-1}	governs strength of electrostatic damping	defaults to 0.2 Å^{-1}.
tempSet	logical	resample velocities from a Maxwell-Boltzmann distribution set to targetTemp	default is false.
thermalTime	fs	how often to perform a tempSet	default is never
targetTemp	K	sets the target temperature	no default value

```
nIntegrable
time; Hxx Hyx Hzx; Hxy Hyy Hzy; Hxz Hyz Hzz;
Name1 x y z vx vy vz qw qx qy qz jx jy jz
Name2 x y z vx vy vz qw qx qy qz jx jy jz
etc...
```

Scheme 2.5: An example of the format of the coordinate files. The fist line is the number of `integrableObjects` (freely-moving atoms and rigid bodies). The second line begins with the time stamp followed by the three H column vectors. It is important to note that for extended system ensembles, additional information pertinent to the integrators may be stored on this line as well. The next lines are the coordinates for all integrable objects in the system. The name of the integrable object is followed by position, velocity, quaternions, and lastly, body fixed angular momentum.

Table 2.2: Meta-data Keywords: Optional Parameters

keyword	**units**	**use**	**remarks**
`tauThermostat`	fs	time constant for Nosé-Hoover thermostat	times from 1000-10,000 fs are reasonable
`targetPressure`	atm	sets the target pressure	no default value
`surfaceTension`		sets the target surface tension in the x-y plane	no default value
`tauBarostat`	fs	time constant for the Nosé-Hoover-Andersen barostat	times from 10,000 to 100,000 fs are reasonable
`seed`	integer	Sets the seed value for the random number generator.	The seed needs to be at least 9 digits long. The default is to take the seed from the CPU clock.

2.4 Coordinate Files

The standard format for storage of a systems coordinates is a modified xyz-file syntax, the exact details of which can be seen in Scheme 2.5. As all bonding and molecular information is stored in the meta-data files, the coordinate files contain only the coordinates of the objects which move independently during the simulation. It is important to note that *not all atoms* are capable of independent motion. Atoms which are part of rigid bodies are not "integrable objects" in the equations of motion; the rigid bodies themselves are the integrable objects. Therefore, the coordinate file contains coordinates of all the `integrableObjects` in the system. For systems without rigid bodies, this is simply the coordinates of all the atoms.

It is important to note that although the simulation propagates the complete rotation matrix, directional entities are written out using quaternions to save space in the output files. All objects (atoms, orientational atoms, and rigid bodies) are given quaternions and angular momenta in coordinate files which are output by OOPSE, but it is not necessary for the user to specify the quaternions or angular momenta for atoms without orientational degrees of freedom.

The `name` field for atoms is simply the atom type as specified in the meta-data file. The `name` field for a rigid body is specified as `MOLTYPE_RB_N`, to specify that this is `rigidBody` N in a molecule of type MOLTYPE. In simulations with rigid body models of water, a sample coordinate line might be:

```
TIP3P_RB_0 x y z vx vy vz qw qx qy qz jx jy jz
```

which tells the program that the rigid body representing a TIP3P molecule (rigid body # 0) is listed on that line.

There are three files used by OOPSE which are written in the coordinate format. They are: the initial coordinate file (`.in`), the simulation trajectory file (`.dump`), and the final coordinates or "end-of-run" for the simulation (`.eor`). The initial coordinate file is necessary for OOPSE to start the simulation with the proper coordinates, and this file must be generated by the user before the simulation run. The trajectory (or "dump") file is updated during simulation and is used to store snapshots of the coordinates at regular intervals. The first frame is a duplication of the `.in` file, and each subsequent frame is appended to the file at an interval specified in the meta-data file with the `sampleTime` flag. The final coordinate file is the "end-of-run" file. The `.eor` file stores the final configuration of the system for a given simulation. The file is updated at the same time as the `.dump` file, but it only contains the most recent frame. In this way, an `.eor` file may be used to initialize a second simulation should it be necessary to recover from a crash or power outage.

2.5 Generation of Initial Coordinates

As was stated in Sec. 2.4, an initial coordinate file is needed to provide the starting coordinates for a simulation. Since each simulation is different, system creation is left to the end user; however, we have included a few sample programs which make some specialized structures. The `.in` file must list the integrable objects in the correct order. The ordering of the integrable objects relies on the ordering of molecules within the meta-data file. OOPSE expects the order to comply with the following guidelines:

1. All of the molecules of the first declared component are given before proceeding to the molecules of the second component, and so on for all subsequently declared components.

2. The ordering of the atoms for each molecule follows the order declared in the molecule's declaration within the model file.

3. Only atoms which are not members of a `rigidBody` are included.

4. Rigid Body coordinates for a molecule are listed immediately after the the other atoms in a molecule. Some molecules may be entirely rigid, in which case, only the rigid body coordinates are given.

An example is given in the meta-data file in Scheme 2.6 which results in the `.in` file shown in Scheme 2.7.

2.6 The Statistics File

The last output file generated by OOPSE is the statistics file. This file records such statistical quantities as the instantaneous temperature (in K), volume (in Å^3), pressure (in atm), etc. It is written out with the frequency specified in the meta-data file with the `statusTime` keyword. The file allows the user to observe the system variables as a function of simulation time while the simulation is in progress. One useful function the statistics file serves is to monitor the conserved quantity of a given simulation ensemble, allowing the user to gauge the stability of the integrator. The statistics file is denoted with the `.stat` file extension.

```
molecule{
  name = "I2";
  atom[0]{
    type = "I";
  }
  atom[1]{
    type = "I";
  }
  bond{
    members( 0, 1);
  }
}

molecule{
  name = "HCl"
  atom[0]{
    type = "H";
  }
  atom[1]{
    type = "Cl";
  }
  bond{
    members( 0, 1);
  }
}

component{
  type = "HCl";
  nMol = 4;
}
component{
  type = "I2";
  nMol = 1;
}

initialConfig = "mixture.in";
```

Scheme 2.6: Example declaration of the I_2 molecule and the HCl molecule. The two molecules are then included into a simulation.

```
10
0.0;  10.0  0.0  0.0;  0.0  10.0  0.0;  0.0  0.0  10.0;
H  ...
Cl ...
H  ...
Cl ...
H  ...
Cl ...
H  ...
Cl ...
I  ...
I  ...
```

Scheme 2.7: The contents of the `mixture.in` file matching the declarations in Scheme 2.6. Note that even though I_2 is declared before HCl, the `.in` file follows the order *in which the components were included*.

Chapter 3

The Empirical Energy Functions

Like many simulation packages, OOPSE splits the potential energy into the short-ranged (bonded) portion and a long-range (non-bonded) potential,

$$V = V_{\mathrm{short-range}} + V_{\mathrm{long-range}}. \tag{3.1}$$

The short-ranged portion includes the explicit bonds, bends, and torsions which have been defined in the meta-data file for the molecules which are present in the simulation. The functional forms and parameters for these interactions are defined by the force field which is chosen.

Calculating the long-range (non-bonded) potential involves a sum over all pairs of atoms (except for those atoms which are involved in a bond, bend, or torsion with each other). If done poorly, calculating the the long-range interactions for N atoms would involve $N(N-1)/2$ evaluations of atomic distances. To reduce the number of distance evaluations between pairs of atoms, OOPSE uses a switched cutoff with Verlet neighbor lists.[12] It is well known that neutral groups which contain charges will exhibit pathological forces unless the cutoff is applied to the neutral groups evenly instead of to the individual atoms.[14] OOPSE allows users to specify cutoff groups which may contain an arbitrary number of atoms in the molecule. Atoms in a cutoff group are treated as a single unit for the evaluation of the switching function:

$$V_{\mathrm{long-range}} = \sum_{a} \sum_{b>a} s(r_{ab}) \sum_{i \in a} \sum_{j \in b} V_{ij}(r_{ij}), \tag{3.2}$$

where r_{ab} is the distance between the centers of mass of the two cutoff groups (a and b).

The sums over a and b are over the cutoff groups that are present in the simulation. Atoms which are not explicitly defined as members of a `cutoffGroup` are treated as a group consisting of only one atom. The switching function, $s(r)$ is the standard cubic switching function,

$$S(r) = \begin{cases} 1 & \text{if } r \leq r_{\mathrm{sw}}, \\ \frac{(r_{\mathrm{cut}}+2r-3r_{\mathrm{sw}})(r_{\mathrm{cut}}-r)^2}{(r_{\mathrm{cut}}-r_{\mathrm{sw}})^2} & \text{if } r_{\mathrm{sw}} < r \leq r_{\mathrm{cut}}, \\ 0 & \text{if } r > r_{\mathrm{cut}}. \end{cases} \tag{3.3}$$

Here, r_{sw} is the `switchingRadius`, or the distance beyond which interactions are reduced, and r_{cut} is the `cutoffRadius`, or the distance at which interactions are truncated.

Users of OOPSE do not need to specify the `cutoffRadius` or `switchingRadius`. In simulations containing only Lennard-Jones atoms, the cutoff radius has a default value of $2.5\sigma_{ii}$, where σ_{ii} is the largest Lennard-Jones length parameter present in the simulation. In simulations containing charged or dipolar atoms, the default cutoff radius is 15Å.

```
#include "argon.md"

component{
  type = "Ar";
  nMol = 108;
}

initialConfig = "./argon.in";

forceField = "LJ";
```

Scheme 3.1: A sample meta-data file for a small Lennard-Jones simulation.

The `switchingRadius` is set to a default value of 95% of the `cutoffRadius`. In the special case of a simulation containing *only* Lennard-Jones atoms, the default switching radius takes the same value as the cutoff radius, and OOPSE will use a shifted potential to remove discontinuities in the potential at the cutoff. Both radii may be specified in the meta-data file.

Force fields can be added to OOPSE, although it comes with a few simple examples (Lennard-Jones, DUFF, WATER, and EAM) which are explained in the following sections.

3.1 The Lennard Jones Force Field

The most basic force field implemented in OOPSE is the Lennard-Jones force field, which mimics the van der Waals interaction at long distances and uses an empirical repulsion at short distances. The Lennard-Jones potential is given by:

$$V_{\mathrm{LJ}}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right], \tag{3.4}$$

where r_{ij} is the distance between particles i and j, σ_{ij} scales the length of the interaction, and ϵ_{ij} scales the well depth of the potential. Scheme 3.1 gives an example meta-data file that sets up a system of 108 Ar particles to be simulated using the Lennard-Jones force field.

Interactions between dissimilar particles requires the generation of cross term parameters for σ and ϵ. These parameters are determined using the Lorentz-Berthelot mixing rules:[12]

$$\sigma_{ij} = \frac{1}{2}[\sigma_{ii} + \sigma_{jj}], \tag{3.5}$$

and

$$\epsilon_{ij} = \sqrt{\epsilon_{ii}\epsilon_{jj}}. \tag{3.6}$$

3.2 Dipolar Unified-Atom Force Field

The dipolar unified-atom force field (DUFF) was developed to simulate lipid bilayers. These types of simulations require a model capable of forming bilayers, while still being sufficiently computationally efficient to allow large systems ($\sim$100's of phospholipids, $\sim$1000's of waters) to be simulated for long times ($\sim$10's of nanoseconds). With this goal in mind, DUFF has no point charges. Charge-neutral distributions are replaced with dipoles, while most atoms

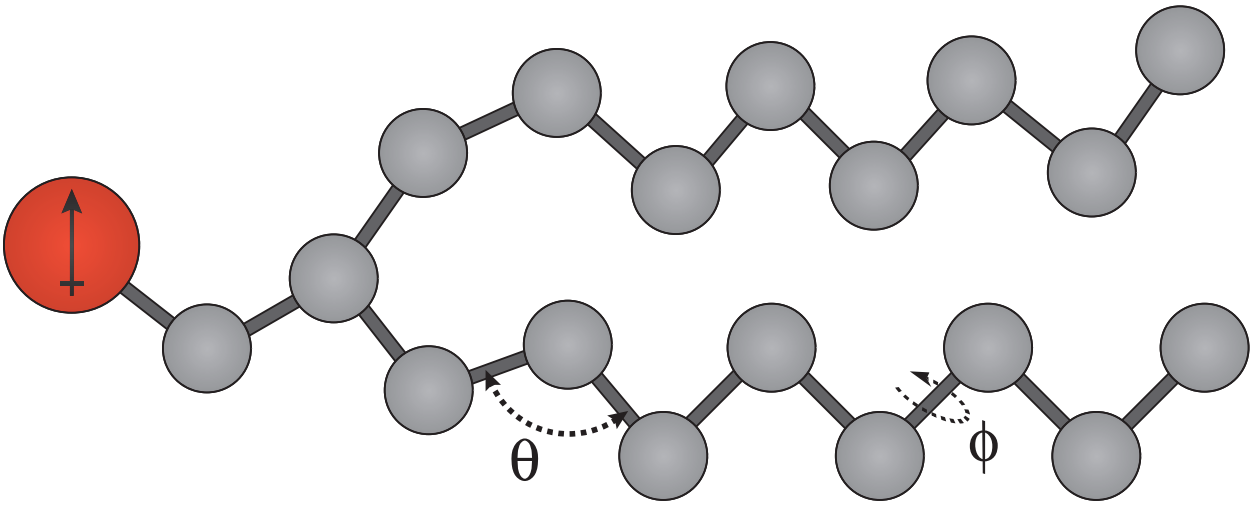

Figure 3.1: A representation of the lipid model. ϕ is the torsion angle, θ is the bend angle, and μ is the dipole moment of the head group.

and groups of atoms are reduced to Lennard-Jones interaction sites. This simplification reduces the length scale of long range interactions from $\frac{1}{r}$ to $\frac{1}{r^3}$, removing the need for the computationally expensive Ewald sum. Instead, Verlet neighbor-lists and cutoff radii are used for the dipolar interactions, and, if desired, a reaction field may be added to mimic longer range interactions.

As an example, lipid head-groups in DUFF are represented as point dipole interaction sites. Placing a dipole at the head group's center of mass mimics the charge separation found in common phospholipid head groups such as phosphatidylcholine.[15] Additionally, a large Lennard-Jones site is located at the pseudoatom's center of mass. The model is illustrated by the red atom in Fig. 3.1. The water model we use to complement the dipoles of the lipids is a reparameterization[16] of the soft sticky dipole (SSD) model of Ichiye *et al.*[17]

A set of scalable parameters has been used to model the alkyl groups with Lennard-Jones sites. For this, parameters from the TraPPE force field of Siepmann *et al.*[18] have been utilized. TraPPE is a unified-atom representation of n-alkanes which is parametrized against phase equilibria using Gibbs ensemble Monte Carlo simulation techniques.[18] One of the advantages of TraPPE is that it generalizes the types of atoms in an alkyl chain to keep the number of pseudoatoms to a minimum; thus, the parameters for a unified atom such as CH_2 do not change depending on what species are bonded to it.

As is required by TraPPE, DUFF also constrains all bonds to be of fixed length. Typically, bond vibrations are the fastest motions in a molecular dynamic simulation. With these vibrations present, small time steps between force evaluations must be used to ensure adequate energy conservation in the bond degrees of freedom. By constraining the bond lengths, larger time steps may be used when integrating the equations of motion. A simulation using DUFF is illustrated in Scheme 3.2.

3.2.1 DUFF Energy Functions

The total potential energy function in DUFF is

$$V = \sum_{I=1}^{N} V^{I}_{\text{Internal}} + \sum_{I=1}^{N-1} \sum_{J>I} V^{IJ}_{\text{Cross}}, \tag{3.7}$$

```
#include "water.md"
#include "lipid.md"

component{
  type = "simpleLipid_16";
  nMol = 60;
}

component{
  type = "SSD_water";
  nMol = 1936;
}

initialConfig = "bilayer.in";

forceField = "DUFF";
```

Scheme 3.2: A portion of a meta-data file showing a simulation utilizing DUFF

where V^I_{Internal} is the internal potential of molecule I:

$$V^I_{\text{Internal}} = \sum_{\theta_{ijk}\in I} V_{\text{bend}}(\theta_{ijk}) + \sum_{\phi_{ijkl}\in I} V_{\text{torsion}}(\phi_{ijkl}) + \sum_{i\in I}\sum_{(j>i+4)\in I}\Big[V_{\text{LJ}}(r_{ij}) + V_{\text{dipole}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j)\Big]. \tag{3.8}$$

Here V_{bend} is the bend potential for all 1, 3 bonded pairs within the molecule I, and V_{torsion} is the torsion potential for all 1, 4 bonded pairs. The pairwise portions of the non-bonded interactions are excluded for atom pairs that are involved in the smae bond, bend, or torsion. All other atom pairs within a molecule are subject to the LJ pair potential.

The bend potential of a molecule is represented by the following function:

$$V_{\text{bend}}(\theta_{ijk}) = k_\theta(\theta_{ijk} - \theta_0)^2, \tag{3.9}$$

where θ_{ijk} is the angle defined by atoms i, j, and k (see Fig. 3.1), θ_0 is the equilibrium bond angle, and k_θ is the force constant which determines the strength of the harmonic bend. The parameters for k_θ and θ_0 are borrowed from those in TraPPE.[18]

The torsion potential and parameters are also borrowed from TraPPE. It is of the form:

$$V_{\text{torsion}}(\phi) = c_1[1 + \cos\phi] + c_2[1 + \cos(2\phi)] + c_3[1 + \cos(3\phi)], \tag{3.10}$$

where:

$$\cos\phi = (\hat{\mathbf{r}}_{ij} \times \hat{\mathbf{r}}_{jk}) \cdot (\hat{\mathbf{r}}_{jk} \times \hat{\mathbf{r}}_{kl}). \tag{3.11}$$

Here, $\hat{\mathbf{r}}_{\alpha\beta}$ are the set of unit bond vectors between atoms i, j, k, and l. For computational efficiency, the torsion potential has been recast after the method of CHARMM,[1] in which the angle series is converted to a power series of the form:

$$V_{\text{torsion}}(\phi) = k_3\cos^3\phi + k_2\cos^2\phi + k_1\cos\phi + k_0, \tag{3.12}$$

where:

$$
\begin{aligned}
k_0 &= c_1 + c_3, \\
k_1 &= c_1 - 3c_3, \\
k_2 &= 2c_2, \\
k_3 &= 4c_3.
\end{aligned}
$$

By recasting the potential as a power series, repeated trigonometric evaluations are avoided during the calculation of the potential energy.

The cross potential between molecules I and J, V_{Cross}^{IJ}, is as follows:

$$
V_{\text{Cross}}^{IJ} = \sum_{i \in I} \sum_{j \in J} \Bigg[V_{\text{LJ}}(r_{ij}) + V_{\text{dipole}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) + V_{\text{sticky}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) \Bigg], \tag{3.13}
$$

where V_{LJ} is the Lennard Jones potential, V_{dipole} is the dipole dipole potential, and V_{sticky} is the sticky potential defined by the SSD model (Sec. 3.2.2). Note that not all atom types include all interactions.

The dipole-dipole potential has the following form:

$$
V_{\text{dipole}}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) = \frac{|\mu_i||\mu_j|}{4\pi\epsilon_0 r_{ij}^3} \Bigg[\hat{\boldsymbol{u}}_i \cdot \hat{\boldsymbol{u}}_j - 3(\hat{\boldsymbol{u}}_i \cdot \hat{\mathbf{r}}_{ij})(\hat{\boldsymbol{u}}_j \cdot \hat{\mathbf{r}}_{ij}) \Bigg]. \tag{3.14}
$$

Here $\mathbf{r}_{ij}$ is the vector starting at atom i pointing towards j, and $\mathbf{\Omega}_i$ and $\mathbf{\Omega}_j$ are the orientational degrees of freedom for atoms i and j respectively. The magnitude of the dipole moment of atom i is $|\mu_i|$, $\hat{\boldsymbol{u}}_i$ is the standard unit orientation vector of $\mathbf{\Omega}_i$, and $\hat{\boldsymbol{r}}_{ij}$ is the unit vector pointing along $\mathbf{r}_{ij}$ ($\hat{\boldsymbol{r}}_{ij} = \mathbf{r}_{ij}/|\mathbf{r}_{ij}|$).

3.2.2 The DUFF Water Models: SSD/E and SSD/RF

In the interest of computational efficiency, the default solvent used by OOPSE is the extended Soft Sticky Dipole (SSD/E) water model.[16] The original SSD was developed by Ichiye *et al.*[17] as a modified form of the hard-sphere water model proposed by Bratko, Blum, and Luzar.[19, 20] It consists of a single point dipole with a Lennard-Jones core and a sticky potential that directs the particles to assume the proper hydrogen bond orientation in the first solvation shell. Thus, the interaction between two SSD water molecules *i* and *j* is given by the potential

$$
V_{ij} = V_{ij}^{LJ}(r_{ij}) \; + V_{ij}^{dp}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) \; + V_{ij}^{sp}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j), \tag{3.15}
$$

where the $\mathbf{r}_{ij}$ is the position vector between molecules *i* and *j* with magnitude equal to the distance r_{ij}, and $\mathbf{\Omega}_i$ and $\mathbf{\Omega}_j$ represent the orientations of the respective molecules. The Lennard-Jones and dipole parts of the potential are given by equations 3.4 and 3.14 respectively. The sticky part is described by the following,

$$
u_{ij}^{sp}(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) = \frac{\nu_0}{2} [s(r_{ij}) w(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) + s'(r_{ij}) w'(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j)] \, , \tag{3.16}
$$

where ν_0 is a strength parameter for the sticky potential, and s and s' are cubic switching functions which turn off the sticky interaction beyond the first solvation shell. The w function can be thought of as an attractive potential with tetrahedral geometry:

$$
w(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) = \sin\theta_{ij} \sin 2\theta_{ij} \cos 2\phi_{ij}, \tag{3.17}
$$

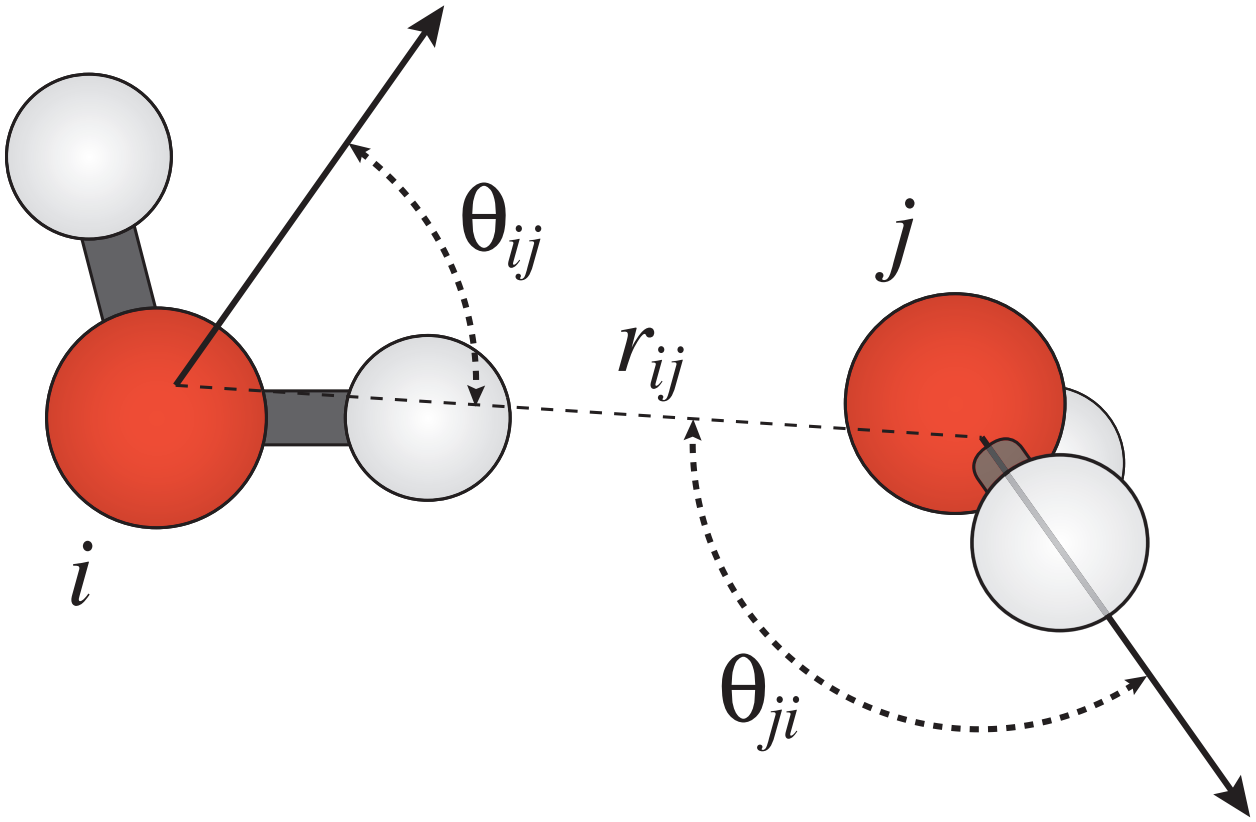

Figure 3.2: Coordinates for the interaction between two SSD/E water molecules. θ_{ij} is the angle that r_{ij} makes with the $\hat{z}$ vector in the body-fixed frame for molecule i. The $\hat{z}$ vector bisects the HOH angle in each water molecule.

while the w' function counters the normal aligned and anti-aligned structures favored by point dipoles:

$$w'(\mathbf{r}_{ij}, \mathbf{\Omega}_i, \mathbf{\Omega}_j) = (\cos\theta_{ij} - 0.6)^2(\cos\theta_{ij} + 0.8)^2 - w^0, \tag{3.18}$$

It should be noted that w is proportional to the sum of the Y_3^2 and Y_3^{-2} spherical harmonics (a linear combination which enhances the tetrahedral geometry for hydrogen bonded structures), while w' is a purely empirical function. A more detailed description of the functional parts and variables in this potential can be found in the original SSD articles.[17, 21, 22, 23]

Since SSD/E is a single-point *dipolar* model, the force calculations are simplified significantly relative to the standard *charged* multi-point models. In the original Monte Carlo simulations using this model, Ichiye *et al.* reported that using SSD decreased computer time by a factor of 6-7 compared to other models.[17] What is most impressive is that these savings did not come at the expense of accurate depiction of the liquid state properties. Indeed, SSD/E maintains reasonable agreement with the Head-Gordon diffraction data for the structural features of liquid water.[24, 17] Additionally, the dynamical properties exhibited by SSD/E agree with experiment better than those of more computationally expensive models (like TIP3P and SPC/E).[22] The combination of speed and accurate depiction of solvent properties makes SSD/E a very attractive model for the simulation of large scale biochemical simulations.

Recent constant pressure simulations revealed issues in the original SSD model that led to lower than expected densities at all target pressures.[23, 16] The default model in OOPSE is therefore SSD/E, a density corrected derivative of SSD that exhibits improved liquid structure and transport behavior. If the use of a reaction field long-range interaction correction is desired, it is recommended that the parameters be modified to those of the SSD/RF model (an SSD

variant parameterized for reaction field). These solvent parameters are listed and can be easily modified in the DUFF force field file (`DUFF.frc`). A table of the parameter values and the drawbacks and benefits of the different density corrected SSD models can be found in reference [16].

3.3 The WATER Force Field

In addition to the DUFF force field's solvent description, a separate WATER force field has been included for simulating most of the common rigid-body water models. This force field includes the simple and point-dipolar models (SSD, SSD1, SSD/E, SSD/RF, and DPD water), as well as the common charge-based models (SPC, SPC/E, TIP3P, TIP4P, and TIP5P).[17, 23, 16, 25, 26, 27, 28, 29] In order to handle these models, charge-charge interactions were included in the force-loop:

$$V_{\text{charge}}(r_{ij}) = \sum_{ij} \frac{q_i q_j e^2}{r_{ij}}, \tag{3.19}$$

where q represents the charge on particle i or j, and e is the charge of an electron in Coulombs. The charge-charge interaction support is rudimentary in the current version of OOPSE. As with the other pair interactions, charges can be simulated with a pure cutoff or a reaction field. The various methods for performing the Ewald summation have not yet been included. Also, the charge-dipole and charge-quadrupole (for interactions between SSD type water and charges) are not yet available, so it is currently inadvisable to mix dipolar and charge based molecules in the same system. The WATER force field can be easily expanded through modification of the WATER force field file (`WATER.frc`). By adding atom types and inserting the appropriate parameters, it is possible to extend the force field to handle rigid molecules other than water.

3.4 Embedded Atom Method

OOPSE implements a potential that describes bonding in transition metal systems. [30, 31, 32, 33, 34] This potential has an attractive interaction which models "Embedding" a positively charged pseudo-atom core in the electron density due to the free valance "sea" of electrons created by the surrounding atoms in the system. A pairwise part of the potential (which is primarily repulsive) describes the interaction of the positively charged metal core ions with one another. The Embedded Atom Method (EAM) [35, 36, 37, 38] has been widely adopted in the materials science community and has been included in OOPSE. A good review of EAM and other formulations of metallic potentials was given by Voter.[39]

The EAM potential has the form:

$$V = \sum_i F_i\left[\rho_i\right] + \sum_i \sum_{j \neq i} \phi_{ij}(\mathbf{r}_{ij}) \tag{3.20}$$

where F_i is an embedding functional that approximates the energy required to embed a positively-charged core ion i into a linear superposition of spherically averaged atomic electron densities given by ρ_i,

$$\rho_i = \sum_{j \neq i} f_j(\mathbf{r}_{ij}), \tag{3.21}$$

Since the density at site i (ρ_i) must be computed before the embedding functional can be evaluated, EAM and the related transition metal potentials require two loops through the atom pairs to compute the inter-atomic forces.

The pairwise portion of the potential, ϕ_{ij}, is a primarily repulsive interaction between atoms i and j. In the original formulation of EAM[35], ϕ_{ij} was an entirely repulsive term; however later refinements to EAM allowed for more general forms for ϕ.[40] The effective cutoff distance, r_{cut} is the distance at which the values of $f(r)$ and $\phi(r)$

drop to zero for all atoms present in the simulation. In practice, this distance is fairly small, limiting the summations in the EAM equation to the few dozen atoms surrounding atom i for both the density ρ and pairwise ϕ interactions.

In computing forces for alloys, mixing rules as outlined by Johnson [37] are used to compute the heterogenous pair potential,

$$\phi_{ab}(r) = \frac{1}{2}\left(\frac{f_b(r)}{f_a(r)}\phi_{aa}(r) + \frac{f_a(r)}{f_b(r)}\phi_{bb}(r)\right). \tag{3.22}$$

No mixing rule is needed for the densities, since the density at site i is simply the linear sum of density contributions of all the other atoms.

The EAM force field illustrates an additional feature of OOPSE. Foiles, Baskes and Daw fit EAM potentials for Cu, Ag, Au, Ni, Pd, Pt and alloys of these metals.[36] These fits are included in OOPSE as the `u3` variant of the EAM force field. Voter and Chen reparamaterized a set of EAM functions which do a better job of predicting melting points.[41] These functions are included in OOPSE as the `VC` variant of the EAM force field. An additional set of functions (the "Universal 6" functions) are included in OOPSE as the `u6` variant of EAM. For example, to specify the Voter-Chen variant of the EAM force field, the user would add the `forceFieldVariant = "VC";` line to the meta-data file.

The potential files used by the EAM force field are in the standard `funcfl` format, which is the format utilized by a number of other codes (e.g. ParaDyn [8], DYNAMO 86). It should be noted that the energy units in these files are in eV, not kcal mol^{-1} as in the rest of the OOPSE force field files.

3.5 The Sutton-Chen Force Field

The Sutton-Chen (SC) [32] potential has been used to study a wide range of phenomena in metals. Although it is similar in form to the EAM potential, the Sutton-Chen model takes on a simpler form,

$$U_{tot} = \sum_i \left[\frac{1}{2}\sum_{j\neq i} D_{ij} V_{ij}^{pair}(r_{ij}) - c_i D_{ii}\sqrt{\rho_i}\right], \tag{3.23}$$

where V_{ij}^{pair} and ρ_i are given by

$$V_{ij}^{pair}(r) = \left(\frac{\alpha_{ij}}{r_{ij}}\right)^{n_{ij}}, \rho_i = \sum_{j\neq i}\left(\frac{\alpha_{ij}}{r_{ij}}\right)^{m_{ij}} \tag{3.24}$$

V_{ij}^{pair} is a repulsive pairwise potential that accounts for interactions of the pseudo-atom cores. The $\sqrt{\rho_i}$ term in Eq. (3.23) is an attractive many-body potential that models the interactions between the valence electrons and the cores of the pseudo-atoms. D_{ij}, D_{ii}, c_i and α_{ij} are parameters used to tune the potential for different transition metals.

The SC potential form has also been parameterized by Qi *et al.*[33] These parameters were obtained via empirical and *ab initio* calculations to match structural features of the FCC crystal. To specify the original Sutton-Chen variant of the SC force field, the user would add the `forceFieldVariant = "SC";` line to the meta-data file, while specification of the Qi *et al.* quantum-adapted variant of the SC potential, the user would add the `forceFieldVariant = "QSC";` line to the meta-data file.

3.6 The CLAY force field

The CLAY force field is based on an ionic (nonbonded) description of the metal-oxygen interactions associated with hydrated phases. All atoms are represented as point charges and are allowed complete translational freedom. Metal-oxygen interactions are based on a simple Lennard-Jones potential combined with electrostatics. The empirical parameters were optimized by Cygan *et al.*[42] on the basis of known mineral structures, and partial atomic charges were derived from periodic DFT quantum chemical calculations of simple oxide, hydroxide, and oxyhydroxide model compounds with well-defined structures.

3.7 Periodic Boundary Conditions

Periodic boundary conditions are widely used to simulate bulk properties with a relatively small number of particles. In this method the simulation box is replicated throughout space to form an infinite lattice. During the simulation, when a particle moves in the primary cell, its image in other cells move in exactly the same direction with exactly the same orientation. Thus, as a particle leaves the primary cell, one of its images will enter through the opposite face. If the simulation box is large enough to avoid "feeling" the symmetries of the periodic lattice, surface effects can be ignored. The available periodic cells in OOPSE are cubic, orthorhombic and parallelepiped. OOPSE use a 3×3 matrix, H, to describe the shape and size of the simulation box. H is defined:

$$\mathsf{H} = (\mathbf{h}_x, \mathbf{h}_y, \mathbf{h}_z), \tag{3.25}$$

where $\mathbf{h}_\alpha$ is the column vector of the α axis of the box. During the course of the simulation both the size and shape of the box can be changed to allow volume fluctuations when constraining the pressure.

A real space vector, $\mathbf{r}$ can be transformed in to a box space vector, $\mathbf{s}$, and back through the following transformations:

$$\mathbf{s} = \mathsf{H}^{-1}\mathbf{r}, \tag{3.26}$$

$$\mathbf{r} = \mathsf{H}\mathbf{s}. \tag{3.27}$$

The vector $\mathbf{s}$ is now a vector expressed as the number of box lengths in the $\mathbf{h}_x$, $\mathbf{h}_y$, and $\mathbf{h}_z$ directions. To find the minimum image of a vector $\mathbf{r}$, OOPSE first converts it to its corresponding vector in box space, and then casts each element to lie in the range $[-0.5, 0.5]$:

$$s'_i = s_i - \mathrm{round}(s_i), \tag{3.28}$$

where s_i is the ith element of $\mathbf{s}$, and $\mathrm{round}(s_i)$ is given by

$$\mathrm{round}(x) = \begin{cases} \lfloor x + 0.5 \rfloor & \text{if } x \geq 0, \\ \lceil x - 0.5 \rceil & \text{if } x < 0. \end{cases} \tag{3.29}$$

Here $\lfloor x \rfloor$ is the floor operator, and gives the largest integer value that is not greater than x, and $\lceil x \rceil$ is the ceiling operator, and gives the smallest integer that is not less than x.

Finally, the minimum image coordinates $\mathbf{r}'$ are obtained by transforming back to real space,

$$\mathbf{r}' = \mathsf{H}^{-1}\mathbf{s}'. \tag{3.30}$$

In this way, particles are allowed to diffuse freely in $\mathbf{r}$, but their minimum images, or $\mathbf{r}'$, are used to compute the

inter-atomic forces.

Chapter 4

Mechanics

4.1 Integrating the Equations of Motion: the DLM method

The default method for integrating the equations of motion in OOPSE is a velocity-Verlet version of the symplectic splitting method proposed by Dullweber, Leimkuhler and McLachlan (DLM).[43] When there are no directional atoms or rigid bodies present in the simulation, this integrator becomes the standard velocity-Verlet integrator which is known to sample the microcanonical (NVE) ensemble.[44]

Previous integration methods for orientational motion have problems that are avoided in the DLM method. Direct propagation of the Euler angles has a known $1/\sin\theta$ divergence in the equations of motion for ϕ and ψ,[12] leading to numerical instabilities any time one of the directional atoms or rigid bodies has an orientation near $\theta = 0$ or $\theta = \pi$. Quaternion-based integration methods work well for propagating orientational motion; however, energy conservation concerns arise when using the microcanonical (NVE) ensemble. An earlier implementation of OOPSE utilized quaternions for propagation of rotational motion; however, a detailed investigation showed that they resulted in a steady drift in the total energy, something that has been observed by Laird *et al.*[45]

The key difference in the integration method proposed by Dullweber *et al.* is that the entire 3×3 rotation matrix is propagated from one time step to the next. In the past, this would not have been feasible, since the rotation matrix for a single body has nine elements compared with the more memory-efficient methods (using three Euler angles or 4 quaternions). Computer memory has become much less costly in recent years, and this can be translated into substantial benefits in energy conservation.

The basic equations of motion being integrated are derived from the Hamiltonian for conservative systems containing rigid bodies,

$$H = \sum_i \left(\frac{1}{2} m_i \mathbf{v}_i^T \cdot \mathbf{v}_i + \frac{1}{2} \mathbf{j}_i^T \cdot \overleftrightarrow{\mathsf{I}}_i^{-1} \cdot \mathbf{j}_i \right) + V\left(\{\mathbf{r}\}, \{\mathsf{A}\}\right), \tag{4.1}$$

where $\mathbf{r}_i$ and $\mathbf{v}_i$ are the cartesian position vector and velocity of the center of mass of particle i, and $\mathbf{j}_i$, $\overleftrightarrow{\mathsf{I}}_i$ are the body-fixed angular momentum and moment of inertia tensor respectively, and the superscript T denotes the transpose of the vector. A_i is the 3×3 rotation matrix describing the instantaneous orientation of the particle. V is the potential energy function which may depend on both the positions $\{\mathbf{r}\}$ and orientations $\{\mathsf{A}\}$ of all particles. The equations of motion for the particle centers of mass are derived from Hamilton's equations and are quite simple,

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{4.2}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m}, \tag{4.3}$$

where $\mathbf{f}$ is the instantaneous force on the center of mass of the particle,

$$\mathbf{f} = -\frac{\partial}{\partial \mathbf{r}} V(\{\mathbf{r}(t)\}, \{\mathsf{A}(t)\}). \tag{4.4}$$

The equations of motion for the orientational degrees of freedom are

$$\dot{\mathsf{A}} = \mathsf{A} \cdot \text{ skew}\left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right), \tag{4.5}$$

$$\dot{\mathbf{j}} = \mathbf{j} \times \left(\overleftrightarrow{\mathsf{I}}^{-1} \cdot \mathbf{j}\right) - \text{ rot}\left(\mathsf{A}^T \cdot \frac{\partial V}{\partial \mathsf{A}}\right). \tag{4.6}$$

In these equations of motion, the skew matrix of a vector $\mathbf{v} = (v_1, v_2, v_3)$ is defined:

$$\text{skew}\,(\mathbf{v}) := \begin{pmatrix} 0 & v_3 & -v_2 \\ -v_3 & 0 & v_1 \\ v_2 & -v_1 & 0 \end{pmatrix}. \tag{4.7}$$

The rot notation refers to the mapping of the 3×3 rotation matrix to a vector of orientations by first computing the skew-symmetric part $\left(\mathsf{A} - \mathsf{A}^T\right)$ and then associating this with a length 3 vector by inverting the skew function above:

$$\text{rot}\,(\mathsf{A}) := \text{ skew}^{-1}\left(\mathsf{A} - \mathsf{A}^T\right). \tag{4.8}$$

Written this way, the rot operation creates a set of conjugate angle coordinates to the body-fixed angular momenta represented by $\mathbf{j}$. This equation of motion for angular momenta is equivalent to the more familiar body-fixed forms,

$$\dot{j_x} = \tau_x^b(t) - \left(\overleftrightarrow{\mathsf{I}}_{yy}^{-1} - \overleftrightarrow{\mathsf{I}}_{zz}^{-1}\right) j_y j_z, \tag{4.9}$$

$$\dot{j_y} = \tau_y^b(t) - \left(\overleftrightarrow{\mathsf{I}}_{zz}^{-1} - \overleftrightarrow{\mathsf{I}}_{xx}^{-1}\right) j_z j_x, \tag{4.10}$$

$$\dot{j_z} = \tau_z^b(t) - \left(\overleftrightarrow{\mathsf{I}}_{xx}^{-1} - \overleftrightarrow{\mathsf{I}}_{yy}^{-1}\right) j_x j_y, \tag{4.11}$$

which utilize the body-fixed torques, τ^b. Torques are most easily derived in the space-fixed frame,

$$\tau^b(t) = \mathsf{A}(t) \cdot \tau^s(t), \tag{4.12}$$

where the torques are either derived from the forces on the constituent atoms of the rigid body, or for directional atoms, directly from derivatives of the potential energy,

$$\tau^s(t) = -\hat{\mathbf{u}}(t) \times \left(\frac{\partial}{\partial \hat{\mathbf{u}}} V\left(\{\mathbf{r}(t)\}, \{\mathsf{A}(t)\}\right)\right). \tag{4.13}$$

Here $\hat{\mathbf{u}}$ is a unit vector pointing along the principal axis of the particle in the space-fixed frame.

The DLM method uses a Trotter factorization of the orientational propagator. This has three effects:

1. the integrator is area-preserving in phase space (i.e. it is *symplectic*),

2. the integrator is time-*reversible*, making it suitable for Hybrid Monte Carlo applications, and

3. the error for a single time step is of order $\mathcal{O}\left(h^4\right)$ for timesteps of length h.

The integration of the equations of motion is carried out in a velocity-Verlet style 2-part algorithm, where $h = \delta t$:

`moveA:`

$$\begin{aligned}
\mathbf{v}\left(t+h/2\right) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\mathbf{f}(t)/m\right), \\
\mathbf{r}(t+h) &\leftarrow \mathbf{r}(t) + h\mathbf{v}\left(t+h/2\right), \\
\mathbf{j}\left(t+h/2\right) &\leftarrow \mathbf{j}(t) + \frac{h}{2}\tau^b(t), \\
\mathsf{A}(t+h) &\leftarrow \mathrm{rotate}\left(h\mathbf{j}(t+h/2)\cdot \overleftrightarrow{\mathsf{I}}^{-1}\right).
\end{aligned}$$

In this context, the rotate function is the reversible product of the three body-fixed rotations,

$$\mathrm{rotate}(\mathbf{a}) = \mathsf{G}_x(a_x/2)\cdot\mathsf{G}_y(a_y/2)\cdot\mathsf{G}_z(a_z)\cdot\mathsf{G}_y(a_y/2)\cdot\mathsf{G}_x(a_x/2), \tag{4.14}$$

where each rotational propagator, $\mathsf{G}_\alpha(\theta)$, rotates both the rotation matrix (A) and the body-fixed angular momentum ($\mathbf{j}$) by an angle θ around body-fixed axis α,

$$\mathsf{G}_\alpha(\theta) = \begin{cases} \mathsf{A}(t) & \leftarrow & \mathsf{A}(0)\cdot\mathsf{R}_\alpha(\theta)^T, \\ \mathbf{j}(t) & \leftarrow & \mathsf{R}_\alpha(\theta)\cdot\mathbf{j}(0). \end{cases} \tag{4.15}$$

R_α is a quadratic approximation to the single-axis rotation matrix. For example, in the small-angle limit, the rotation matrix around the body-fixed x-axis can be approximated as

$$\mathsf{R}_x(\theta) \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & \frac{1-\theta^2/4}{1+\theta^2/4} & -\frac{\theta}{1+\theta^2/4} \\ 0 & \frac{\theta}{1+\theta^2/4} & \frac{1-\theta^2/4}{1+\theta^2/4} \end{pmatrix}. \tag{4.16}$$

All other rotations follow in a straightforward manner.

After the first part of the propagation, the forces and body-fixed torques are calculated at the new positions and orientations

`doForces:`

$$\begin{aligned}
\mathbf{f}(t+h) &\leftarrow -\left(\frac{\partial V}{\partial \mathbf{r}}\right)_{\mathbf{r}(t+h)}, \\
\tau^s(t+h) &\leftarrow \mathbf{u}(t+h)\times\frac{\partial V}{\partial \mathbf{u}}, \\
\tau^b(t+h) &\leftarrow \mathsf{A}(t+h)\cdot\tau^s(t+h).
\end{aligned}$$

OOPSE automatically updates $\mathbf{u}$ when the rotation matrix A is calculated in `moveA`. Once the forces and torques have been obtained at the new time step, the velocities can be advanced to the same time value.

`moveB:`

$$\begin{aligned}
\mathbf{v}\left(t+h\right) &\leftarrow \mathbf{v}\left(t+h/2\right) + \frac{h}{2}\left(\mathbf{f}(t+h)/m\right), \\
\mathbf{j}\left(t+h\right) &\leftarrow \mathbf{j}\left(t+h/2\right) + \frac{h}{2}\tau^b(t+h).
\end{aligned}$$

The matrix rotations used in the DLM method end up being more costly computationally than the simpler arithmetic quaternion propagation. With the same time step, a 1024-molecule water simulation incurs an average 12% increase in computation time using the DLM method in place of quaternions. This cost is more than justified when comparing

the energy conservation achieved by the two methods. Figure 4.1 provides a comparative analysis of the DLM method versus the traditional quaternion scheme.

In Fig. 4.1, δE_1 is a measure of the linear energy drift in units of kcal mol^{-1} per particle over a nanosecond of simulation time, and δE_0 is the standard deviation of the energy fluctuations in units of kcal mol^{-1} per particle. In the top plot, it is apparent that the energy drift is reduced by a significant amount (2 to 3 orders of magnitude improvement at all tested time steps) by chosing the DLM method over the simple non-symplectic quaternion integration method. In addition to this improvement in energy drift, the fluctuations in the total energy are also dampened by 1 to 2 orders of magnitude by utilizing the DLM method.

Although the DLM method is more computationally expensive than the traditional quaternion scheme for propagating a single time step, consideration of the computational cost for a long simulation with a particular level of energy conservation is in order. A plot of energy drift versus computational cost was generated (Fig. 4.2). This figure provides an estimate of the CPU time required under the two integration schemes for 1 nanosecond of simulation time for the model 1024-molecule system. By chosing a desired energy drift value it is possible to determine the CPU time required for both methods. If a δE_1 of 1×10^{-3}kcal mol^{-1} per particle is desired, a nanosecond of simulation time will require 19 hours of CPU time with the DLM integrator, while the quaternion scheme will require 154 hours of CPU time. This demonstrates the computational advantage of the integration scheme utilized in OOPSE.

There is only one specific keyword relevant to the default integrator, and that is the time step for integrating the equations of motion.

variable	**Meta-data keyword**	**units**	**default value**
h	`dt = 2.0;`	fs	none

4.2 Extended Systems for other Ensembles

OOPSE implements a number of extended system integrators for sampling from other ensembles relevant to chemical physics. The integrator can be selected with the `ensemble` keyword in the meta-data file:

Integrator	**Ensemble**	**Meta-data instruction**
NVE	microcanonical	`ensemble = NVE;`
NVT	canonical	`ensemble = NVT;`
NPTi	isobaric-isothermal (with isotropic volume changes)	`ensemble = NPTi;`
NPTf	isobaric-isothermal (with changes to box shape)	`ensemble = NPTf;`
NPTxyz	approximate isobaric-isothermal (with separate barostats on each box dimension)	`ensemble = NPTxyz;`

The relatively well-known Nosé-Hoover thermostat[46] is implemented in OOPSE's NVT integrator. This method couples an extra degree of freedom (the thermostat) to the kinetic energy of the system and it has been shown to sample the canonical distribution in the system degrees of freedom while conserving a quantity that is, to within a constant, the Helmholtz free energy.[47]

NPT algorithms attempt to maintain constant pressure in the system by coupling the volume of the system to a barostat. OOPSE contains three different constant pressure algorithms. The first two, NPTi and NPTf have been shown to conserve a quantity that is, to within a constant, the Gibbs free energy.[47] The Melchionna modification to the Hoover barostat is implemented in both NPTi and NPTf. NPTi allows only isotropic changes in the simulation box, while box *shape* variations are allowed in NPTf. The NPTxyz integrator has *not* been shown to sample from the

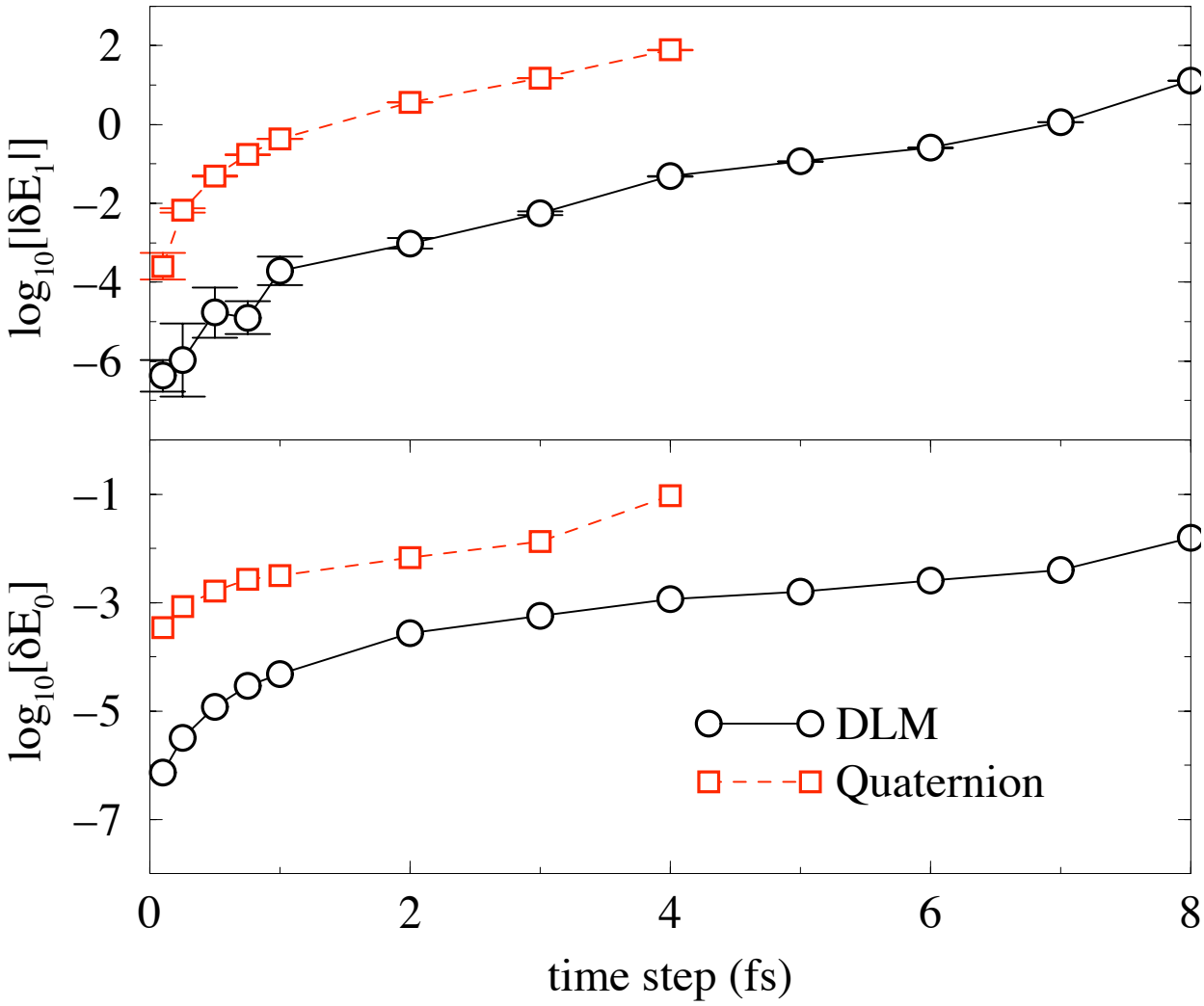

Figure 4.1: Analysis of the energy conservation of the DLM and quaternion integration methods. δE_1 is the linear drift in energy over time and δE_0 is the standard deviation of energy fluctuations around this drift. All simulations were of a 1024-molecule simulation of SSD water at 298 K starting from the same initial configuration. Note that the DLM method provides more than an order of magnitude improvement in both the energy drift and the size of the energy fluctuations when compared with the quaternion method at any given time step. At time steps larger than 4 fs, the quaternion scheme resulted in rapidly rising energies which eventually lead to simulation failure. Using the DLM method, time steps up to 8 fs can be taken before this behavior is evident.

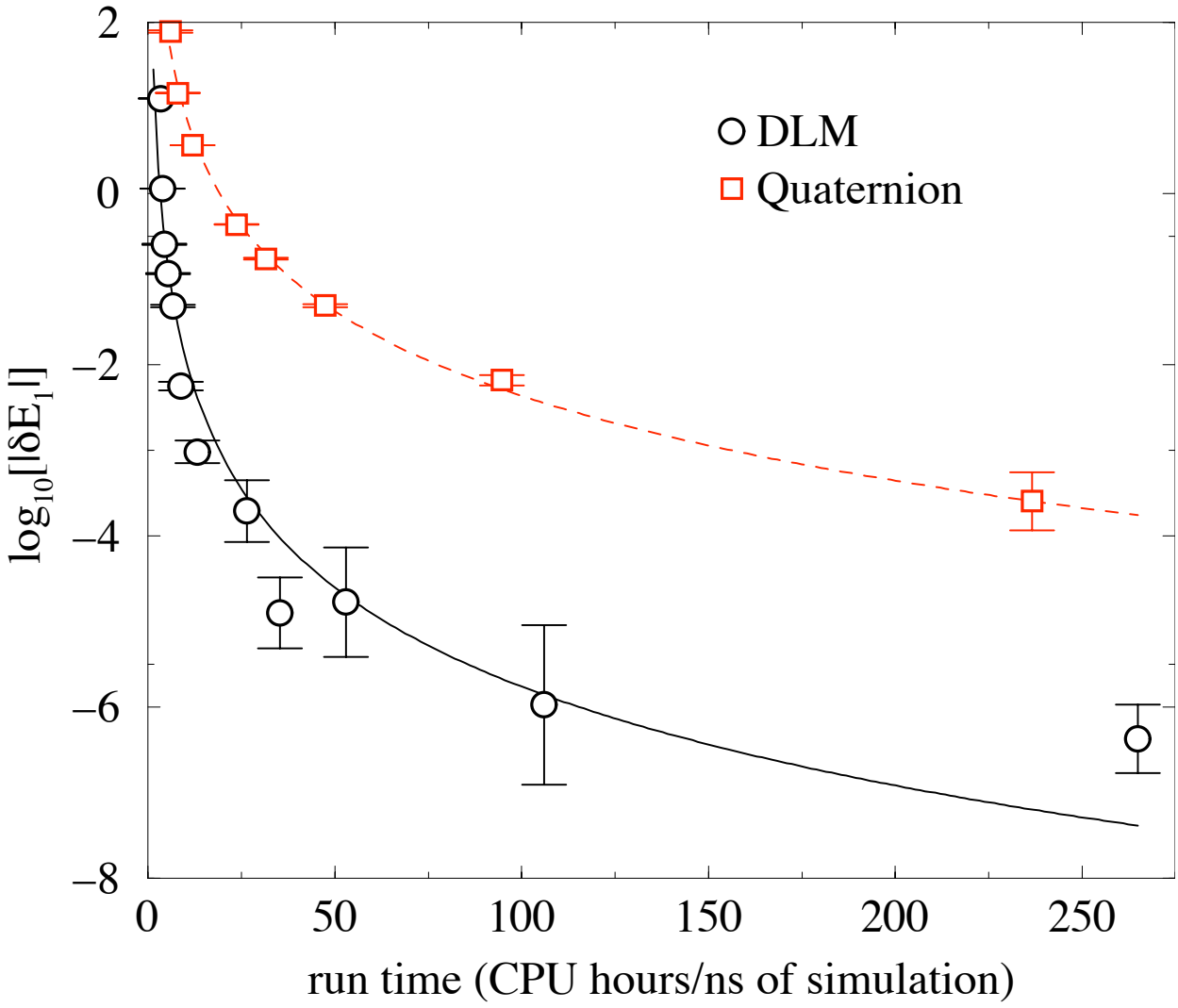

Figure 4.2: Energy drift as a function of required simulation run time. δE_1 is the linear drift in energy over time. Simulations were performed on a single 2.5 GHz Pentium 4 processor. Simulation time comparisons can be made by tracing horizontally from one curve to the other. For example, a simulation that takes 24 hours using the DLM method will take roughly 210 hours using the simple quaternion method if the same degree of energy conservation is desired.

isobaric-isothermal ensemble. It is useful, however, in that it maintains orthogonality for the axes of the simulation box while attempting to equalize pressure along the three perpendicular directions in the box.

Each of the extended system integrators requires additional keywords to set target values for the thermodynamic state variables that are being held constant. Keywords are also required to set the characteristic decay times for the dynamics of the extended variables.

variable	**Meta-data instruction**	**units**	**default value**
$T_{\rm target}$	`targetTemperature = 300;`	K	none
$P_{\rm target}$	`targetPressure = 1;`	atm	none
τ_T	`tauThermostat = 1e3;`	fs	none
τ_B	`tauBarostat = 5e3;`	fs	none
	`resetTime = 200;`	fs	none
	`useInitialExtendedSystemState = true;`	logical	true

Two additional keywords can be used to either clear the extended system variables periodically (`resetTime`), or to maintain the state of the extended system variables between simulations (`useInitialExtendedSystemState`). More details on these variables and their use in the integrators follows below.

4.3 Nosé-Hoover Thermostatting

The Nosé-Hoover equations of motion are given by[46]

$$\dot{\mathbf{r}} = \mathbf{v}, \tag{4.17}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} - \chi\mathbf{v}, \tag{4.18}$$

$$\dot{\mathsf{A}} = \mathsf{A}\cdot \text{ skew}\left(\overleftrightarrow{\mathsf{I}}^{-1}\cdot\mathbf{j}\right), \tag{4.19}$$

$$\dot{\mathbf{j}} = \mathbf{j}\times\left(\overleftrightarrow{\mathsf{I}}^{-1}\cdot\mathbf{j}\right) - \text{ rot}\left(\mathsf{A}^T\cdot\frac{\partial V}{\partial \mathsf{A}}\right) - \chi\mathbf{j}. \tag{4.20}$$

χ is an "extra" variable included in the extended system, and it is propagated using the first order equation of motion

$$\dot{\chi} = \frac{1}{\tau_T^2}\left(\frac{T}{T_{\rm target}} - 1\right). \tag{4.21}$$

The instantaneous temperature T is proportional to the total kinetic energy (both translational and orientational) and is given by

$$T = \frac{2K}{fk_B} \tag{4.22}$$

Here, f is the total number of degrees of freedom in the system,

$$f = 3N + 2N_{\rm linear} + 3N_{\rm non-linear} - N_{\rm constraints}, \tag{4.23}$$

and K is the total kinetic energy,

$$K = \sum_{i=1}^{N}\frac{1}{2}m_i\mathbf{v}_i^T\cdot\mathbf{v}_i + \sum_{i=1}^{N_{\rm linear}+N_{\rm non-linear}}\frac{1}{2}\mathbf{j}_i^T\cdot\overleftrightarrow{\mathsf{I}}_i^{-1}\cdot\mathbf{j}_i. \tag{4.24}$$

$N_{\rm linear}$ is the number of linear rotors (i.e. with two non-zero moments of inertia), and $N_{\rm non-linear}$ is the number of non-linear rotors (i.e. with three non-zero moments of inertia).

In eq.(4.21), τ_T is the time constant for relaxation of the temperature to the target value. To set values for τ_T or T_{target} in a simulation, one would use the `tauThermostat` and `targetTemperature` keywords in the meta-data file. The units for `tauThermostat` are fs, and the units for the `targetTemperature` are degrees K. The integration of the equations of motion is carried out in a velocity-Verlet style 2 part algorithm:

`moveA:`

$$\begin{aligned}
T(t) &\leftarrow \{\mathbf{v}(t)\}, \{\mathbf{j}(t)\}, \\
\mathbf{v}(t+h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\frac{\mathbf{f}(t)}{m} - \mathbf{v}(t)\chi(t)\right), \\
\mathbf{r}(t+h) &\leftarrow \mathbf{r}(t) + h\mathbf{v}(t+h/2), \\
\mathbf{j}(t+h/2) &\leftarrow \mathbf{j}(t) + \frac{h}{2}\left(\tau^b(t) - \mathbf{j}(t)\chi(t)\right), \\
\mathsf{A}(t+h) &\leftarrow \text{rotate}\left(h * \mathbf{j}(t+h/2)\overleftrightarrow{\mathsf{I}}^{-1}\right), \\
\chi(t+h/2) &\leftarrow \chi(t) + \frac{h}{2\tau_T^2}\left(\frac{T(t)}{T_{\text{target}}} - 1\right).
\end{aligned}$$

Here $\text{rotate}(h * \mathbf{j}\overleftrightarrow{\mathsf{I}}^{-1})$ is the same symplectic Trotter factorization of the three rotation operations that was discussed in the section on the DLM integrator. Note that this operation modifies both the rotation matrix A and the angular momentum $\mathbf{j}$. `moveA` propagates velocities by a half time step, and positional degrees of freedom by a full time step. The new positions (and orientations) are then used to calculate a new set of forces and torques in exactly the same way they are calculated in the `doForces` portion of the DLM integrator.

Once the forces and torques have been obtained at the new time step, the temperature, velocities, and the extended system variable can be advanced to the same time value.

`moveB:`

$$\begin{aligned}
T(t+h) &\leftarrow \{\mathbf{v}(t+h)\}, \{\mathbf{j}(t+h)\}, \\
\chi(t+h) &\leftarrow \chi(t+h/2) + \frac{h}{2\tau_T^2}\left(\frac{T(t+h)}{T_{\text{target}}} - 1\right), \\
\mathbf{v}(t+h) &\leftarrow \mathbf{v}(t+h/2) + \frac{h}{2}\left(\frac{\mathbf{f}(t+h)}{m} - \mathbf{v}(t+h)\chi(th)\right), \\
\mathbf{j}(t+h) &\leftarrow \mathbf{j}(t+h/2) + \frac{h}{2}\left(\tau^b(t+h) - \mathbf{j}(t+h)\chi(t+h)\right).
\end{aligned}$$

Since $\mathbf{v}(t+h)$ and $\mathbf{j}(t+h)$ are required to calculate $T(t+h)$ as well as $\chi(t+h)$, they indirectly depend on their own values at time $t+h$. `moveB` is therefore done in an iterative fashion until $\chi(t+h)$ becomes self-consistent. The relative tolerance for the self-consistency check defaults to a value of 10^{-6}, but OOPSE will terminate the iteration after 4 loops even if the consistency check has not been satisfied.

The Nosé-Hoover algorithm is known to conserve a Hamiltonian for the extended system that is, to within a constant, identical to the Helmholtz free energy,[47]

$$H_{\text{NVT}} = V + K + fk_BT_{\text{target}}\left(\frac{\tau_T^2\chi^2(t)}{2} + \int_0^t \chi(t')dt'\right). \tag{4.25}$$

Poor choices of h or τ_T can result in non-conservation of H_{NVT}, so the conserved quantity is maintained in the last column of the `.stat` file to allow checks on the quality of the integration.

Bond constraints are applied at the end of both the `moveA` and `moveB` portions of the algorithm. Details on the constraint algorithms are given in section 4.7.1.

4.4 Constant-pressure integration with isotropic box deformations (NPTi)

To carry out isobaric-isothermal ensemble calculations, OOPSE implements the Melchionna modifications to the Nosé-Hoover-Andersen equations of motion.[47] The equations of motion are the same as NVT with the following exceptions:

$$\dot{\mathbf{r}} = \mathbf{v} + \eta\left(\mathbf{r} - \mathbf{R}_0\right), \tag{4.26}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} - (\eta + \chi)\mathbf{v}, \tag{4.27}$$

$$\dot{\eta} = \frac{1}{\tau_B^2 f k_B T_{\text{target}}} V\left(P - P_{\text{target}}\right), \tag{4.28}$$

$$\dot{\mathcal{V}} = 3\mathcal{V}\eta. \tag{4.29}$$

χ and η are the "extra" degrees of freedom in the extended system. χ is a thermostat, and it has the same function as it does in the Nosé-Hoover NVT integrator. η is a barostat which controls changes to the volume of the simulation box. $\mathbf{R}_0$ is the location of the center of mass for the entire system, and $\mathcal{V}$ is the volume of the simulation box. At any time, the volume can be calculated from the determinant of the matrix which describes the box shape:

$$\mathcal{V} = \det(\mathsf{H}). \tag{4.30}$$

The NPTi integrator requires an instantaneous pressure. This quantity is calculated via the pressure tensor,

$$\overleftrightarrow{\mathsf{P}}(t) = \frac{1}{\mathcal{V}(t)}\left(\sum_{i=1}^{N} m_i \mathbf{v}_i(t) \otimes \mathbf{v}_i(t)\right) + \overleftrightarrow{\mathsf{W}}(t). \tag{4.31}$$

The kinetic contribution to the pressure tensor utilizes the *outer* product of the velocities, denoted by the $\otimes$ symbol. The stress tensor is calculated from another outer product of the inter-atomic separation vectors ($\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i$) with the forces between the same two atoms,

$$\overleftrightarrow{\mathsf{W}}(t) = \sum_i \sum_{j>i} \mathbf{r}_{ij}(t) \otimes \mathbf{f}_{ij}(t). \tag{4.32}$$

In systems containing cutoff groups, the stress tensor is computed between the centers-of-mass of the cutoff groups:

$$\overleftrightarrow{\mathsf{W}}(t) = \sum_a \sum_b \mathbf{r}_{ab}(t) \otimes \mathbf{f}_{ab}(t). \tag{4.33}$$

where $\mathbf{r}_{ab}$ is the distance between the centers of mass, and

$$\mathbf{f}_{ab} = s(r_{ab}) \sum_{i \in a} \sum_{j \in b} \mathbf{f}_{ij} + s'(r_{ab}) \frac{\mathbf{r}_{ab}}{|r_{ab}|} \sum_{i \in a} \sum_{j \in b} V_{ij}(\mathbf{r}_{ij}). \tag{4.34}$$

The instantaneous pressure is then simply obtained from the trace of the pressure tensor,

$$P(t) = \frac{1}{3}\text{Tr}\left(\overleftrightarrow{\mathsf{P}}(t)\right). \tag{4.35}$$

In eq.(4.29), τ_B is the time constant for relaxation of the pressure to the target value. To set values for τ_B or P_{target} in a simulation, one would use the `tauBarostat` and `targetPressure` keywords in the meta-data file.

The units for `tauBarostat` are fs, and the units for the `targetPressure` are atmospheres. Like in the NVT integrator, the integration of the equations of motion is carried out in a velocity-Verlet style two part algorithm with only the following differences:

`moveA:`

$$
\begin{aligned}
P(t) &\leftarrow \{\mathbf{r}(t)\}, \{\mathbf{v}(t)\}, \\
\mathbf{v}(t + h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2}\left(\frac{\mathbf{f}(t)}{m} - \mathbf{v}(t)\left(\chi(t) + \eta(t)\right)\right), \\
\eta(t + h/2) &\leftarrow \eta(t) + \frac{h\mathcal{V}(t)}{2Nk_BT(t)\tau_B^2}\left(P(t) - P_{\text{target}}\right), \\
\mathbf{r}(t + h) &\leftarrow \mathbf{r}(t) + h\left\{\mathbf{v}(t + h/2) + \eta(t + h/2)\left[\mathbf{r}(t + h) - \mathbf{R}_0\right]\right\}, \\
\mathsf{H}(t + h) &\leftarrow e^{-h\eta(t+h/2)}\mathsf{H}(t).
\end{aligned}
$$

The propagation of positions to time $t + h$ depends on the positions at the same time. OOPSE carries out this step iteratively (with a limit of 5 passes through the iterative loop). Also, the simulation box H is scaled uniformly for one full time step by an exponential factor that depends on the value of η at time $t + h/2$. Reshaping the box uniformly also scales the volume of the box by

$$\mathcal{V}(t + h) \leftarrow e^{-3h\eta(t+h/2)} \times \mathcal{V}(t). \tag{4.36}$$

The `doForces` step for the NPTi integrator is exactly the same as in both the DLM and NVT integrators. Once the forces and torques have been obtained at the new time step, the velocities can be advanced to the same time value.

`moveB:`

$$
\begin{aligned}
P(t + h) &\leftarrow \{\mathbf{r}(t + h)\}, \{\mathbf{v}(t + h)\}, \\
\eta(t + h) &\leftarrow \eta(t + h/2) + \frac{h\mathcal{V}(t + h)}{2Nk_BT(t + h)\tau_B^2}\left(P(t + h) - P_{\text{target}}\right), \\
\mathbf{v}(t + h) &\leftarrow \mathbf{v}(t + h/2) + \frac{h}{2}\left(\frac{\mathbf{f}(t + h)}{m} - \mathbf{v}(t + h)(\chi(t + h) + \eta(t + h))\right), \\
\mathbf{j}(t + h) &\leftarrow \mathbf{j}(t + h/2) + \frac{h}{2}\left(\tau^b(t + h) - \mathbf{j}(t + h)\chi(t + h)\right).
\end{aligned}
$$

Once again, since $\mathbf{v}(t + h)$ and $\mathbf{j}(t + h)$ are required to calculate $T(t + h)$, $P(t + h)$, $\chi(t + h)$, and $\eta(t + h)$, they indirectly depend on their own values at time $t + h$. `moveB` is therefore done in an iterative fashion until $\chi(t + h)$ and $\eta(t + h)$ become self-consistent. The relative tolerance for the self-consistency check defaults to a value of 10^{-6}, but OOPSE will terminate the iteration after 4 loops even if the consistency check has not been satisfied.

The Melchionna modification of the Nosé-Hoover-Andersen algorithm is known to conserve a Hamiltonian for the extended system that is, to within a constant, identical to the Gibbs free energy,

$$H_{\text{NPTi}} = V + K + fk_BT_{\text{target}}\left(\frac{\tau_T^2\chi^2(t)}{2} + \int_0^t \chi(t')dt'\right) + P_{\text{target}}\mathcal{V}(t). \tag{4.37}$$

Poor choices of δt, τ_T, or τ_B can result in non-conservation of H_{NPTi}, so the conserved quantity is maintained in the last column of the `.stat` file to allow checks on the quality of the integration. It is also known that this algorithm samples the equilibrium distribution for the enthalpy (including contributions for the thermostat and barostat),

$$H_{\text{NPTi}} = V + K + \frac{fk_BT_{\text{target}}}{2}\left(\chi^2\tau_T^2 + \eta^2\tau_B^2\right) + P_{\text{target}}\mathcal{V}(t). \tag{4.38}$$

Bond constraints are applied at the end of both the `moveA` and `moveB` portions of the algorithm. Details on the constraint algorithms are given in section 4.7.1.

4.5 Constant-pressure integration with a flexible box (NPTf)

There is a relatively simple generalization of the Nosé-Hoover-Andersen method to include changes in the simulation box *shape* as well as in the volume of the box. This method utilizes the full 3×3 pressure tensor and introduces a tensor of extended variables ($\overleftrightarrow{\eta}$) to control changes to the box shape. The equations of motion for this method differ from those of NPTi as follows:

$$\dot{\mathbf{r}} = \mathbf{v} + \overleftrightarrow{\eta} \cdot (\mathbf{r} - \mathbf{R}_0), \tag{4.39}$$

$$\dot{\mathbf{v}} = \frac{\mathbf{f}}{m} - (\overleftrightarrow{\eta} + \chi \cdot 1)\mathbf{v}, \tag{4.40}$$

$$\dot{\overleftrightarrow{\eta}} = \frac{1}{\tau_B^2 f k_B T_{\text{target}}} V \left(\overleftrightarrow{\mathsf{P}} - P_{\text{target}} 1 \right), \tag{4.41}$$

$$\dot{\mathsf{H}} = \overleftrightarrow{\eta} \cdot \mathsf{H}. \tag{4.42}$$

Here, 1 is the unit matrix and $\overleftrightarrow{\mathsf{P}}$ is the pressure tensor. Again, the volume, $\mathcal{V} = \det \mathsf{H}$.

The propagation of the equations of motion is nearly identical to the NPTi integration:

`moveA:`

$$\begin{aligned}
\overleftrightarrow{\mathsf{P}}(t) &\leftarrow \{\mathbf{r}(t)\}, \{\mathbf{v}(t)\}, \\
\mathbf{v}(t + h/2) &\leftarrow \mathbf{v}(t) + \frac{h}{2} \left(\frac{\mathbf{f}(t)}{m} - (\chi(t)1 + \overleftrightarrow{\eta}(t)) \cdot \mathbf{v}(t) \right), \\
\overleftrightarrow{\eta}(t + h/2) &\leftarrow \overleftrightarrow{\eta}(t) + \frac{h\mathcal{V}(t)}{2Nk_B T(t)\tau_B^2} \left(\overleftrightarrow{\mathsf{P}}(t) - P_{\text{target}} 1 \right), \\
\mathbf{r}(t + h) &\leftarrow \mathbf{r}(t) + h \left\{ \mathbf{v}(t + h/2) + \overleftrightarrow{\eta}(t + h/2) \cdot [\mathbf{r}(t + h) - \mathbf{R}_0] \right\}, \\
\mathsf{H}(t + h) &\leftarrow \mathsf{H}(t) \cdot e^{-h \overleftrightarrow{\eta}(t + h/2)}.
\end{aligned}$$

OOPSE uses a power series expansion truncated at second order for the exponential operation which scales the simulation box.

The `moveB` portion of the algorithm is largely unchanged from the NPTi integrator:

`moveB:`

$$\begin{aligned}
\overleftrightarrow{\mathsf{P}}(t + h) &\leftarrow \{\mathbf{r}(t + h)\}, \{\mathbf{v}(t + h)\}, \{\mathbf{f}(t + h)\}, \\
\overleftrightarrow{\eta}(t + h) &\leftarrow \overleftrightarrow{\eta}(t + h/2) + \frac{h\mathcal{V}(t + h)}{2Nk_B T(t + h)\tau_B^2} \left(\overleftrightarrow{P}(t + h) - P_{\text{target}} 1 \right), \\
\mathbf{v}(t + h) &\leftarrow \mathbf{v}(t + h/2) + \frac{h}{2} \left(\frac{\mathbf{f}(t + h)}{m} - (\chi(t + h)1 + \overleftrightarrow{\eta}(t + h)) \right) \cdot \mathbf{v}(t + h),
\end{aligned}$$

The iterative schemes for both `moveA` and `moveB` are identical to those described for the NPTi integrator.

The NPTf integrator is known to conserve the following Hamiltonian:

$$H_{\text{NPTf}} = V + K + f k_B T_{\text{target}} \left(\frac{\tau_T^2 \chi^2(t)}{2} + \int_0^t \chi(t') dt' \right) + P_{\text{target}} \mathcal{V}(t) + \frac{f k_B T_{\text{target}}}{2} \text{Tr} \left[\overleftrightarrow{\eta}(t) \right]^2 \tau_B^2. \tag{4.43}$$

This integrator must be used with care, particularly in liquid simulations. Liquids have very small restoring forces in the off-diagonal directions, and the simulation box can very quickly form elongated and sheared geometries which become smaller than the cutoff radius. The NPTf integrator finds most use in simulating crystals or liquid crystals which assume non-orthorhombic geometries.

4.6 Constant pressure in 3 axes (NPTxyz)

There is one additional extended system integrator which is somewhat simpler than the NPTf method described above. In this case, the three axes have independent barostats which each attempt to preserve the target pressure along the box walls perpendicular to that particular axis. The lengths of the box axes are allowed to fluctuate independently, but the angle between the box axes does not change. The equations of motion are identical to those described above, but only the *diagonal* elements of $\overleftrightarrow{\eta}$ are computed. The off-diagonal elements are set to zero (even when the pressure tensor has non-zero off-diagonal elements).

It should be noted that the NPTxyz integrator is *not* known to preserve any Hamiltonian of interest to the chemical physics community. The integrator is extremely useful, however, in generating initial conditions for other integration methods. It *is* suitable for use with liquid simulations, or in cases where there is orientational anisotropy in the system (i.e. in lipid bilayer simulations).

4.7 Constraint Methods

4.7.1 The RATTLE Method for Bond Constraints

In order to satisfy the constraints of fixed bond lengths within OOPSE, we have implemented the RATTLE algorithm of Andersen.[48] RATTLE is a velocity-Verlet formulation of the SHAKE method[49] for iteratively solving the Lagrange multipliers which maintain the holonomic constraints. Both methods are covered in depth in the literature,[14, 12] and a detailed description of this method would be redundant.

4.7.2 The Z-Constraint Method

A force auto-correlation method based on the fluctuation-dissipation theorem was developed by Roux and Karplus to investigate the dynamics of ions inside ion channels.[50] The time-dependent friction coefficient can be calculated from the deviation of the instantaneous force from its mean value:

$$\xi(z,t) = \langle \delta F(z,t) \delta F(z,0) \rangle / k_B T, \tag{4.44}$$

where

$$\delta F(z,t) = F(z,t) - \langle F(z,t) \rangle. \tag{4.45}$$

If the time-dependent friction decays rapidly, the static friction coefficient can be approximated by

$$\xi_{\text{static}}(z) = \int_0^\infty \langle \delta F(z,t) \delta F(z,0) \rangle dt. \tag{4.46}$$

This allows the diffusion constant to then be calculated through the Einstein relation:[51]

$$D(z) = \frac{k_B T}{\xi_{\text{static}}(z)} = \frac{(k_B T)^2}{\int_0^\infty \langle \delta F(z,t) \delta F(z,0) \rangle dt}. \tag{4.47}$$

The Z-Constraint method, which fixes the z coordinates of a few "tagged" molecules with respect to the center of the mass of the system is a technique that was proposed to obtain the forces required for the force auto-correlation calculation.[51] However, simply resetting the coordinate will move the center of the mass of the whole system. To avoid this problem, we have developed a new method that is utilized in OOPSE. Instead of resetting the coordinates, we reset the forces of z-constrained molecules and subtract the total constraint forces from the rest of the system after the force calculation at each time step.

After the force calculation, the total force on molecule α is:

$$G_\alpha = \sum_i F_{\alpha i}, \tag{4.48}$$

where $F_{\alpha i}$ is the force in the z direction on atom i in z-constrained molecule α. The forces on the atoms in the z-constrained molecule are then adjusted to remove the total force on molecule α:

$$F_{\alpha i} = F_{\alpha i} - \frac{m_{\alpha i} G_\alpha}{\sum_i m_{\alpha i}}. \tag{4.49}$$

Here, $m_{\alpha i}$ is the mass of atom i in the z-constrained molecule. After the forces have been adjusted, the velocities must also be modified to subtract out molecule α's center-of-mass velocity in the z direction.

$$v_{\alpha i} = v_{\alpha i} - \frac{\sum_i m_{\alpha i} v_{\alpha i}}{\sum_i m_{\alpha i}}, \tag{4.50}$$

where $v_{\alpha i}$ is the velocity of atom i in the z direction. Lastly, all of the accumulated constraint forces must be subtracted from the rest of the unconstrained system to keep the system center of mass of the entire system from drifting.

$$F_{\beta i} = F_{\beta i} - \frac{m_{\beta i} \sum_\alpha G_\alpha}{\sum_\beta \sum_i m_{\beta i}}, \tag{4.51}$$

where β denotes all *unconstrained* molecules in the system. Similarly, the velocities of the unconstrained molecules must also be scaled:

$$v_{\beta i} = v_{\beta i} + \sum_\alpha \frac{\sum_i m_{\alpha i} v_{\alpha i}}{\sum_i m_{\alpha i}}. \tag{4.52}$$

This method will pin down the centers-of-mass of all of the z-constrained molecules, and will also keep the entire system fixed at the original system center-of-mass location.

At the very beginning of the simulation, the molecules may not be at their desired positions. To steer a z-constrained molecule to its specified position, a simple harmonic potential is used:

$$U(t) = \frac{1}{2} k_{\text{Harmonic}} (z(t) - z_{\text{cons}})^2, \tag{4.53}$$

where k_{Harmonic} is an harmonic force constant, $z(t)$ is the current z coordinate of the center of mass of the constrained molecule, and z_{cons} is the desired constraint position. The harmonic force operating on the z-constrained molecule at time t can be calculated by

$$F_{z_{\text{Harmonic}}}(t) = -\frac{\partial U(t)}{\partial z(t)} = -k_{\text{Harmonic}} (z(t) - z_{\text{cons}}). \tag{4.54}$$

The user may also specify the use of a constant velocity method (steered molecular dynamics) to move the molecules to their desired initial positions. Based on concepts from atomic force microscopy, SMD has been used to study many processes which occur via rare events on the time scale of a few hundreds of picoseconds. For example,SMD has been used to observe the dissociation of Streptavidin-biotin Complex.[52]

To use of the z-constraint method in an OOPSE simulation, the molecules must be specified using the `nZconstraints` keyword in the meta-data file. The other parameters for modifying the behavior of the z-constraint method are listed in table 4.1.

Table 4.1: Meta-data Keywords: Z-Constraint Parameters

keyword	**units**	**use**	**remarks**
`zconsTime`	fs	Sets the frequency at which the `.fz` file is written	
`zconsForcePolicy`	string	The strategy for subtracting the z-constraint force from the *unconstrained* molecules	Possible strategies are `BYMASS` and `BYNUMBER`. The default strategy is `BYMASS`
`zconsGap`	Å	Sets the distance between two adjacent constraint positions	Used mainly to move molecules through a simulation to estimate potentials of mean force.
`zconsFixtime`	fs	Sets the length of time the z-constraint molecule is held fixed	`zconsFixtime` must be set if `zconsGap` is set
`zconsUsingSMD`	logical	Flag for using Steered Molecular Dynamics to move the molecules to the correct constrained positions	Harmonic Forces are used by default

Chapter 5

Thermodynamic Integration

Thermodynamic integration is an established technique that has been used extensively in the calculation of free energies for condensed phases of materials.[53, 54, 55, 56, 57]. This method uses a sequence of simulations during which the system of interest is converted into a reference system for which the free energy is known analytically (A_0). The difference in potential energy between the reference system and the system of interest (ΔV) is then integrated in order to determine the free energy difference between the two states:

$$A = A_0 + \int_0^1 \langle \Delta V \rangle_\lambda \, d\lambda. \tag{5.1}$$

Here, λ is the parameter that governs the transformation between the reference system and the system of interest. For crystalline phases, an harmonically-restrained (Einstein) crystal is chosen as the reference state, while for liquid phases, the ideal gas is taken as the reference state.

In an Einstein crystal, the molecules are restrained at their ideal lattice locations and orientations. Using harmonic restraints, as applied by Bàez and Clancy, the total potential for this reference crystal (V_{EC}) is the sum of all the harmonic restraints,

$$V_{\mathrm{EC}} = \sum_i \left[\frac{K_{\mathrm{v}}}{2}(r_i - r_i^\circ)^2 + \frac{K_\theta}{2}(\theta_i - \theta_i^\circ)^2 + \frac{K_\omega}{2}(\omega_i - \omega_i^\circ)^2 \right], \tag{5.2}$$

where K_{v}, K_θ, and K_ω are the spring constants restraining translational motion and deflection of and rotation around the principle axis of the molecule respectively. The values of θ range from 0 to π, while ω ranges from $-\pi$ to π.

The partition function for a molecular crystal restrained in this fashion can be evaluated analytically, and the Helmholtz Free Energy (*A*) is given by

$$\begin{aligned} \frac{A}{N} = \frac{E_m}{N} \; - \; kT \ln \left(\frac{kT}{h\nu} \right)^3 \quad &- \quad kT \ln \left[\pi^{\frac{1}{2}} \left(\frac{8\pi^2 I_{\mathrm{A}} kT}{h^2} \right)^{\frac{1}{2}} \left(\frac{8\pi^2 I_{\mathrm{B}} kT}{h^2} \right)^{\frac{1}{2}} \left(\frac{8\pi^2 I_{\mathrm{C}} kT}{h^2} \right)^{\frac{1}{2}} \right] \\ &- \quad kT \ln \left[\frac{kT}{2(\pi K_\omega K_\theta)^{\frac{1}{2}}} \exp\left(-\frac{kT}{2K_\theta} \right) \int_0^{\left(\frac{kT}{2K_\theta}\right)^{\frac{1}{2}}} \exp(t^2) \mathrm{d}t \right], \end{aligned} \tag{5.3}$$

where $2\pi\nu = (K_{\mathrm{v}}/m)^{1/2}$, and E_m is the minimum potential energy of the ideal crystal.[56]

OOPSE can perform the simulations that aid the user in constructing the thermodynamic path from the molecular system to one of the reference systems. To do this, the user sets the value of λ (between 0 & 1) in the meta-data file. If the system of interest is crystalline, OOPSE must be able to find the *reference* configuration of the system in a file called `idealCrystal.in` in the directory from which the simulation was run. This file is a standard `.dump` file,

but all information about velocities and angular momenta are discarded when the file is read.

The configuration found in the `idealCrystal.in` file is used for the reference positions and molecular orientations of the Einstein crystal. To complete the specification of the Einstein crystal, a set of force constants must also be specified; one for displacments of the molecular centers of mass, and two for displacements from the ideal orientations of the molecules.

To construct a thermodynamic integration path, the user would run a sequence of N simulations, each with a different value of lambda between 0 and 1. When `useSolidThermInt` is set to `true` in the meta-data file, two additional energy columns are reported in the `.stat` file for the simulation. The first, `vRaw`, is the unperturbed energy for the configuration, and the second, `vHarm`, is the energy of the harmonic (Einstein) system in an identical configuration. The total potential energy of the configuration is a linear combination of `vRaw` and `vHarm` weighted by the value of λ.

From a running average of the difference between `vRaw` and `vHarm`, the user can obtain the integrand in Eq. (5.1) for fixed value of λ.

There are two additional files with the suffixes `.zang0` and `.zang` generated by OOPSE during the first run of a solid thermodynamic integration. These files contain the initial (`.zang0`) and final (`.zang`) values of the angular displacement coordinates for each of the molecules. These are particularly useful when chaining a number of simulations (with successive values of λ) together.

For *liquid* thermodynamic integrations, the reference system is the ideal gas (with a potential exactly equal to 0), so the `.stat` file contains only the standard columns. The potential energy column contains the potential of the *unperturbed* system (and not the λ-weighted potential. This allows the user to use the potential energy directly as the ΔV in the integrand of Eq. (5.1).

Meta-data parameters concerning thermodynamic integrations are given in Table 5.1

Table 5.1: Meta-data Keywords: Thermodynamic Integration Parameters

keyword	**units**	**use**	**remarks**
`useSolidThermInt`	logical	perform thermodynamic integration to an Einstein crystal?	default is "false"
`useLiquidThermInt`	logical	perform thermodynamic integration to an ideal gas?	default is "false"
`thermodynamicIntegrationLambda`	double	transformation parameter	Sets how far along the thermodynamic integration path the simulation will be.
`thermodynamicIntegrationK`	double		power of λ governing shape of integration pathway
`thermIntDistSpringConst`	kcal mol^{-1}Å^{-2}		spring constant for translations in Einstein crystal
`thermIntThetaSpringConst`	kcal mol^{-1}rad^{-2}		spring constant for deflection away from z-axis in Einstein crystal
`thermIntOmegaSpringConst`			

Table 5.1: Meta-data Keywords: Thermodynamic Integration Parameters

keyword	**units**	**use**	**remarks**
	kcal $\mathrm{mol}^{-1}\mathrm{rad}^{-2}$		spring constant for rotation around z-axis in Einstein crystal

Chapter 6

Energy Minimization

As one of the basic procedures of molecular modeling, energy minimization is used to identify local configurations that are stable points on the potential energy surface. There is a vast literature on energy minimization algorithms have been developed to search for the global energy minimum as well as to find local structures which are stable fixed points on the surface. We have included two simple minimization algorithms: steepest descent, (SD) and conjugate gradient (CG) to help users find reasonable local minima from their initial configurations. Since OOPSE handles atoms and rigid bodies which have orientational coordinates as well as translational coordinates, there is some subtlety to the choice of parameters for minimization algorithms.

Given a coordinate set x_k and a search direction d_k, a line search algorithm is performed along d_k to produce $x_{k+1} = x_k + \lambda_k d_k$. In the steepest descent (SD) algorithm,

$$d_k = -\nabla V(x_k). \tag{6.1}$$

The gradient and the direction of next step are always orthogonal. This may cause oscillatory behavior in narrow valleys. To overcome this problem, the Fletcher-Reeves variant [58] of the conjugate gradient (CG) algorithm is used to generate d_{k+1} via simple recursion:

$$d_{k+1} = -\nabla V(x_{k+1}) + \gamma_k d_k \tag{6.2}$$

where

$$\gamma_k = \frac{\nabla V(x_{k+1})^T \nabla V(x_{k+1})}{\nabla V(x_k)^T \nabla V(x_k)}. \tag{6.3}$$

The Polak-Ribiere variant [59] of the conjugate gradient (γ_k) is defined as

$$\gamma_k = \frac{[\nabla V(x_{k+1}) - \nabla V(x)]^T \nabla V(x_{k+1})}{\nabla V(x_k)^T \nabla V(x_k)} \tag{6.4}$$

It is widely agreed that the Polak-Ribiere variant gives better convergence than the Fletcher-Reeves variant, so the conjugate gradient approach implemented in OOPSE is the Polak-Ribiere variant.

The conjugate gradient method assumes that the conformation is close enough to a local minimum that the potential energy surface is very nearly quadratic. When the initial structure is far from the minimum, the steepest descent method can be superior to the conjugate gradient method. Hence, the steepest descent method is often used for the first 10-100 steps of minimization. Another useful feature of minimization methods in OOPSE is that a modified SHAKE algorithm can be applied during the minimization to constraint the bond lengths if this is required by the force field. Meta-data parameters concerning the minimizer are given in Table 6.1

Table 6.1: Meta-data Keywords: Energy Minimizer Parameters

keyword	**units**	**use**	**remarks**
`minimizer`	string	selects the minimization method to be used	either `CG` (conjugate gradient) or `SD` (steepest descent)
`minimizerMaxIter`	steps	Sets the maximum number of iterations for the energy minimization	The default value is 200
`minimizerWriteFreq`	steps	Sets the frequency with which the `.dump` and `.stat` files are writtern during energy minimization	
`minimizerStepSize`	Å	Sets the step size for the line search	The default value is 0.01
`minimizerFTol`	kcal mol^{-1}	Sets the energy tolerance for stopping the minimzia-tion.	The default value is 10^{-8}
`minimizerGTol`	kcal mol^{-1}Å^{-1}	Sets the gradient tolerance for stopping the minimiza-tion.	The default value is 10^{-8}
`minimizerLSTol`	kcal mol^{-1}	Sets line search tolerance for terminating each step of the minimization.	The default value is 10^{-8}
`minimizerLSMaxIter`	steps	Sets the maximum number of iterations for each line search	The default value is 50

Chapter 7

Analysis of Physical Properties

OOPSE includes a few utility programs which compute properties from the dump files that are generated during a molecular dynamics simulation. These programs are:

Dump2XYZ Converts an OOPSE dump file into a file suitable for viewing in a molecular dynamics viewer like Jmol

StaticProps Computes static properties like the pair distribution function, $g(r)$.

DynamicProps Computes time correlation functions like the velocity autocorrelation function, $\langle v(t) \cdot v(0) \rangle$, or the mean square displacement $\langle |r(t) - r(0)|^2 \rangle$.

These programs often need to operate on a subset of the data contained within a dump file. For example, if you want only the *oxygen-oxygen* pair distribution from a water simulation, or if you want to make a movie including only the water molecules within a 6 angstrom radius of lipid head groups, you need a way to specify your selection to these utility programs. OOPSE has a selection syntax which allows you to specify the selection in a compact form in order to generate only the data you want. For example a common use of the StaticProps command would be:

```
StaticProps -i tp4.dump --gofr --sele1="select O*" --sele2="select O*"
```

This command computes the oxygen-oxygen pair distribution function, $g_{OO}(r)$, from a file named `tp4.dump`. In order to understand this selection syntax and to make full use of the selection capabilities of the analysis programs, it is necessary to understand a few of the core concepts that are used to perform simulations.

7.1 Concepts

OOPSE manipulates both traditional atoms as well as some objects that *behave like atoms*. These objects can be rigid collections of atoms or atoms which have orientational degrees of freedom. Here is a diagram of the class heirarchy:

- A **StuntDouble** is *any* object that can be manipulated by the integrators and minimizers.
- An **Atom** is a fundamental point-particle that can be moved around during a simulation.
- A **DirectionalAtom** is an atom which has *orientational* as well as translational degrees of freedom.
- A **RigidBody** is a collection of **Atom**s or **DirectionalAtom**s which behaves as a single unit.

Every Molecule, Atom and DirectionalAtom in OOPSE have their own names which are specified in the `.md` file. In contrast, RigidBodies are denoted by their membership and index inside a particular molecule: [Molecule-Name]_RB_[index] (the contents inside the brackets depend on the specifics of the simulation). The names of rigid bodies are generated automatically. For example, the name of the first rigid body in a DMPC molecule is DMPC_RB_0.

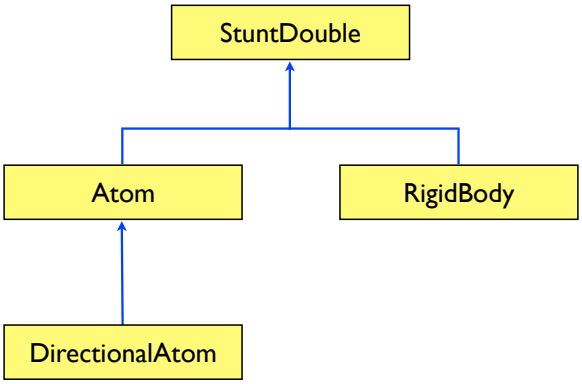

Figure 7.1:
The class heirarchy of StuntDoubles in OOPSE-3.0. The selection syntax allows the user to select any of the objects that are descended from a StuntDouble.

7.2 Syntax of the Select Command

The most general form of the select command is: `select` *expression*

This expression represents an arbitrary set of StuntDoubles (Atoms or RigidBodies) in OOPSE. Expressions are composed of either name expressions, index expressions, predefined sets, user-defined expressions, comparison operators, within expressions, or logical combinations of the above expression types. Expressions can be combined using parentheses and the Boolean operators.

7.2.1 Logical expressions

The logical operators allow complex queries to be constructed out of simpler ones using the standard boolean connectives **and**, **or**, **not**. Parentheses can be used to alter the precedence of the operators.

logical operator	**equivalent operator**
and	“&”, “&&”
or	“|”, “||”, “,”
not	“!”

7.2.2 Name expressions

type of expression	examples	translation of examples
expression without "."	select DMPC	select all StuntDoubles belonging to all DMPC molecules
	select C*	select all atoms which have atom types beginning with C
	select DMPC_RB_*	select all RigidBodies in DMPC molecules (but only select the rigid bodies, and not the atoms belonging to them).
expression has one "."	select TIP3P.O_TIP3P	select the O_TIP3P atoms belonging to TIP3P molecules
	select DMPC_RB_O.PO4	select the PO4 atoms belonging to the first RigidBody in each DMPC molecule
	select DMPC.20	select the twentieth StuntDouble in each DMPC molecule
expression has two "."s	select DMPC.DMPC_RB_?.*	select all atoms belonging to all rigid bodies within all DMPC molecules

7.2.3 Index expressions

examples	translation of examples
select 20	select all of the StuntDoubles belonging to Molecule 20
select 20 to 30	select all of the StuntDoubles belonging to molecules which have global indices between 20 (inclusive) and 30 (exclusive)

7.2.4 Predefined sets

keyword	description
all	select all StuntDoubles
none	select none of the StuntDoubles

7.2.5 User-defined expressions

Users can define arbitrary terms to represent groups of StuntDoubles, and then use the define terms in select commands. The general form for the define command is: **define** *term expression*

Once defined, the user can specify such terms in boolean expressions

```
define SSDWATER SSD or SSD1 or SSDRF
select SSDWATER
```

7.2.6 Comparison expressions

StuntDoubles can be selected by using comparision operators on their properties. The general form for the comparison command is: a property name, followed by a comparision operator and then a number.

property	mass, charge
comparison operator	">", "<", "=", ">=", "<=", "! ="

For example, the phrase `select mass > 16.0 and charge < -2` wouldselect StuntDoubles which have mass greater than 16.0 and charges less than -2.

7.2.7 Within expressions

The "within" keyword allows the user to select all StuntDoubles within the specified distance (in Angstroms) from a selection, including the selected atom itself. The general form for within selection is: `select within(distance, expression)`

For example, the phrase `select within(2.5, PO4 or NC4)` would select all StuntDoubles which are within 2.5 angstroms of PO4 or NC4 atoms.

7.3 Tools which use the selection command

7.3.1 Dump2XYZ

Dump2XYZ can transform an OOPSE dump file into a xyz file which can be opened by other molecular dynamics viewers such as Jmol and VMD. The options available for Dump2XYZ are as follows:

Table 7.1: Dump2XYZ Command-line Options

option	verbose option	behavior
-h	`--help`	Print help and exit
-V	`--version`	Print version and exit
-i	`--input=filename`	input dump file
-o	`--output=filename`	output file name
-n	`--frame=INT`	print every n frame (default='1')
-w	`--water`	skip the the waters (default=off)
-m	`--periodicBox`	map to the periodic box (default=off)
-z	`--zconstraint`	replace the atom types of zconstraint molecules (default=off)
-r	`--rigidbody`	add a pseudo COM atom to rigidbody (default=off)
-t	`--watertype`	replace the atom type of water model (default=on)
-b	`--basetype`	using base atom type (default=off)
	`--repeatX=INT`	The number of images to repeat in the x direction (default='0')
	`--repeatY=INT`	The number of images to repeat in the y direction (default='0')
	`--repeatZ=INT`	The number of images to repeat in the z direction (default='0')
-s	`--selection=selection script`	By specifying `--selection`="selection command" with Dump2XYZ, the user can select an arbitrary set of StuntDoubles to be converted.
	`--originsele`	By specifying `--originsele`="selection command" with Dump2XYZ, the user can re-center the origin of the system around a specific StuntDouble

Table 7.1: Dump2XYZ Command-line Options

option	verbose option	behavior
	`--refsele`	In order to rotate the system, `--originsele` and `--refsele` must be given to define the new coordinate set. A StuntDouble which contains a dipole (the direction of the dipole is always (0, 0, 1) in body frame) is specified by `--originsele`. The new x-z plane is defined by the direction of the dipole and the StuntDouble is specified by `--refsele`.

7.3.2 StaticProps

`StaticProps` can compute properties which are averaged over some or all of the configurations that are contained within a dump file. The most common example of a static property that can be computed is the pair distribution function between atoms of type A and other atoms of type B, $g_{AB}(r)$. StaticProps can also be used to compute the density distributions of other molecules in a reference frame *fixed to the body-fixed reference frame* of a selected atom or rigid body.

There are five seperate radial distribution functions availiable in OOPSE. Since every radial distrbution function invlove the calculation between pairs of bodies, `--sele1` and `--sele2` must be specified to tell StaticProps which bodies to include in the calculation.

`--gofr` Computes the pair distribution function,

$$g_{AB}(r) = \frac{1}{\rho_B}\frac{1}{N_A}\langle\sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\rangle$$

`--r_theta` Computes the angle-dependent pair distribution function. The angle is defined by the intermolecular vector $\vec{r}$ and z-axis of DirectionalAtom A,

$$g_{AB}(r, \cos\theta) = \frac{1}{\rho_B}\frac{1}{N_A}\langle\sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\delta(\cos\theta_{ij} - \cos\theta)\rangle$$

`--r_omega` Computes the angle-dependent pair distribution function. The angle is defined by the z-axes of the two DirectionalAtoms A and B.

$$g_{AB}(r, \cos\omega) = \frac{1}{\rho_B}\frac{1}{N_A}\langle\sum_{i\in A}\sum_{j\in B}\delta(r - r_{ij})\delta(\cos\omega_{ij} - \cos\omega)\rangle$$

`--theta_omega` Computes the pair distribution in the angular space θ, ω defined by the two angles mentioned above.

$$g_{AB}(\cos\theta, \cos\omega) = \frac{1}{\rho_B}\frac{1}{N_A}\langle\sum_{i\in A}\sum_{j\in B}\langle\delta(\cos\theta_{ij} - \cos\theta)\delta(\cos\omega_{ij} - \cos\omega)\rangle$$

`--gxyz` Calculates the density distribution of particles of type B in the body frame of particle A. Therefore, `--originsele` and `--refsele` must be given to define A's internal coordinate set as the reference frame for the calculation.

The vectors (and angles) associated with these angular pair distribution functions are most easily seen in the figure below:

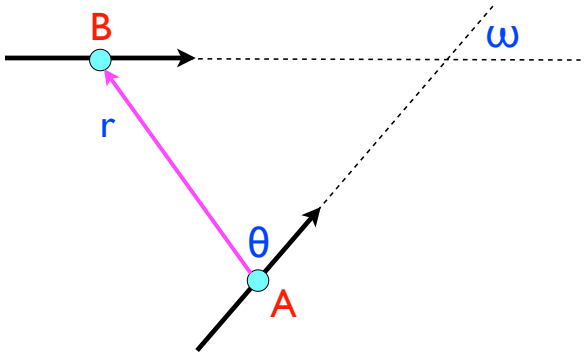

Figure 7.2:
Any two directional objects (DirectionalAtoms and RigidBodies) have a set of two angles (θ, and ω) between the z-axes of their body-fixed frames.

The options available for `StaticProps` are as follows:

Table 7.2: StaticProps Command-line Options

option	**verbose option**	**behavior**
-h	`--help`	Print help and exit
-V	`--version`	Print version and exit
-i	`--input=filename`	input dump file
-o	`--output=filename`	output file name
-n	`--step=INT`	process every n frame (default='1')
-r	`--nrbins=INT`	number of bins for distance (default='100')
-a	`--nanglebins=INT`	number of bins for cos(angle) (default= '50')
-l	`--length=DOUBLE`	maximum length (Defaults to 1/2 smallest length of first frame)
	`--sele1=selection script`	select the first StuntDouble set
	`--sele2=selection script`	select the second StuntDouble set
	`--sele3=selection script`	select the third StuntDouble set
	`--refsele=selection script`	select reference (can only be used with `--gxyz`)
	`--molname=STRING`	molecule name
	`--begin=INT`	begin internal index
	`--end=INT`	end internal index
One option from the following group of options is required:		
	`--gofr`	$g(r)$
	`--r_theta`	$g(r, \cos(\theta))$
	`--r_omega`	$g(r, \cos(\omega))$
	`--theta_omega`	$g(\cos(\theta), \cos(\omega))$
	`--gxyz`	$g(x, y, z)$
	`--p2`	P_2 order parameter (`--sele1` and `--sele2` must be specified)
	`--scd`	S_{CD} order parameter(either `--sele1`, `--sele2`, `--sele3` are specified or `--molname`, `--begin`, `--end` are specified)

Table 7.2: StaticProps Command-line Options

option	verbose option	behavior
	`--density`	density plot (`--sele1` must be specified)
	`--slab_density`	slab density (`--sele1` must be specified)

7.3.3 DynamicProps

`DynamicProps` computes time correlation functions from the configurations stored in a dump file. Typical examples of time correlation functions are the mean square displacement and the velocity autocorrelation functions. Once again, the selection syntax can be used to specify the StuntDoubles that will be used for the calculation. A general time correlation function can be thought of as:

$$C_{AB}(t) = \langle \vec{u}_A(t) \cdot \vec{v}_B(0) \rangle \tag{7.1}$$

where $\vec{u}_A(t)$ is a vector property associated with an atom of type A at time t, and $\vec{v}_B(t')$ is a different vector property associated with an atom of type B at a different time t'. In most autocorrelation functions, the vector properties ($\vec{v}$ and $\vec{u}$) and the types of atoms (A and B) are identical, and the three calculations built in to `DynamicProps` make these assumptions. It is possible, however, to make simple modifications to the `DynamicProps` code to allow the use of *cross* time correlation functions (i.e. with different vectors). The ability to use two selection scripts to select different types of atoms is already present in the code.

The options available for DynamicProps are as follows:

Table 7.3: DynamicProps Command-line Options

option	verbose option	behavior
-h	`--help`	Print help and exit
-V	`--version`	Print version and exit
-i	`--input=filename`	input dump file
-o	`--output=filename`	output file name
	`--sele1=selection script`	select first StuntDouble set
	`--sele2=selection script`	select second StuntDouble set (if sele2 is not set, use script from sele1)
One option from the following group of options is required:		
-r	`--rcorr`	compute mean square displacement
-v	`--vcorr`	compute velocity correlation function
-d	`--dcorr`	compute dipole correlation function

Chapter 8

Preparing Input Configurations

OOPSE version 3 and higher now come with a few utility programs to aid in setting up initial configuration and meta-data files. Usually, a user is interested in either importing a structure from some other format (usually XYZ or PDB), or in building an initial configuration in some perfect crystalline lattice. The programs bundled with OOPSE which import coordinate files are `atom2mdin`, `xyz2mdin`, and `pdb2mdin`.The program which generates perfect crystals is called `SimpleBuilder`.

8.1 atom2mdin, xyz2mdin, and pdb2mdin

`atom2mdin`, `xyz2mdin`, and `pdb2mdin`attempt to construct `.md` and `.in` files from a single file containing only atomic coordinate information. To do this task, they make reasonable guesses about bonding from the distance between atoms in the coordinate, and attempt to identify other terms in the potential energy from the topology of the graph of discovered bonds. This procedure is not perfect, and the user should check the discovered bonding topology that is contained in the meta-data file that is generated.

Typically, the user would run:

`atom2mdin <input spec> [Options]`

Here `<input spec>` can be used to specify the type of file being used for configuration input. I.e. using `-ipdb` specifies that the input file contains coordinate information in the PDB format.

The options available for atom2mdin are as follows:

Table 8.1: atom2mdin Command-line Options

option	**behavior**
-f #	Start import at molecule # specified
-l #	End import at molecule # specified
-t	All input files describe a single molecule
-e	Continue with next object after error, if possible
-z	Compress the output with gzip
-H	Outputs this help text
-Hxxx	(xxx is file format ID e.g. -Hpdb) gives format info
-Hall	Outputs details of all formats
-V	Outputs version number
The following file formats are recognized:	

Table 8.1: atom2mdin Command-line Options

option	behavior
ent	Protein Data Bank format
in	OOPSE cartesian coordinates format
pdb	Protein Data Bank format
prep	Amber Prep format
xyz	XYZ cartesian coordinates format
More specific info and options are available using -H<format-type>, e.g. -Hpdb	

The specific programs `xyz2mdin` and `pdb2mdin` are identical to `atom2mdin`, but they use a specific input format and do not expect the the input specifier on the command line.

8.2 SimpleBuilder

`SimpleBuilder` creates simple lattice structures. It requires an initial meta-data file to specify the components that are to be placed on the lattice. The total number of placed molecules will be shown at the top of the configuration file that is generated, and that number may not match the original meta-data file, so a new meta-data file is also generated which matches the lattice structure.

The options available for SimpleBuilder are as follows:

Table 8.2: SimpleBuilder Command-line Options

option	verbose option	behavior
-h	`--help`	Print help and exit
-V	`--version`	Print version and exit
-o	`--output=STRING`	Output file name
	`--latticetype=STRING`	Lattice type. Valid types are fcc, hcp, bcc, and hcp-water. (default='fcc')
	`--density=DOUBLE`	density ($\mathrm{g\ cm^{-3}}$)
	`--nx=INT`	number of unit cells in x
	`--ny=INT`	number of unit cells in y
	`--nz=INT`	number of unit cells in z

Chapter 9

Parallel Simulation Implementation

Although processor power is continually improving, it is still unreasonable to simulate systems of more than 10,000 atoms on a single processor. To facilitate study of larger system sizes or smaller systems for longer time scales, parallel methods were developed to allow multiple CPU's to share the simulation workload. Three general categories of parallel decomposition methods have been developed: these are the atomic,[60] spatial [61] and force [8] decomposition methods.

Algorithmically simplest of the three methods is atomic decomposition, where N particles in a simulation are split among P processors for the duration of the simulation. Computational cost scales as an optimal $\mathcal{O}(N/P)$ for atomic decomposition. Unfortunately, all processors must communicate positions and forces with all other processors at every force evaluation, leading the communication costs to scale as an unfavorable $\mathcal{O}(N)$, *independent of the number of processors*. This communication bottleneck led to the development of spatial and force decomposition methods, in which communication among processors scales much more favorably. Spatial or domain decomposition divides the physical spatial domain into 3D boxes in which each processor is responsible for calculation of forces and positions of particles located in its box. Particles are reassigned to different processors as they move through simulation space. To calculate forces on a given particle, a processor must simply know the positions of particles within some cutoff radius located on nearby processors rather than the positions of particles on all processors. Both communication between processors and computation scale as $\mathcal{O}(N/P)$ in the spatial method. However, spatial decomposition adds algorithmic complexity to the simulation code and is not very efficient for small N, since the overall communication scales as the surface to volume ratio $\mathcal{O}(N/P)^{2/3}$ in three dimensions.

The parallelization method used in OOPSE is the force decomposition method.[62] Force decomposition assigns particles to processors based on a block decomposition of the force matrix. Processors are split into an optimally square grid forming row and column processor groups. Forces are calculated on particles in a given row by particles located in that processor's column assignment. One deviation from the algorithm described by Hendrickson *et al.* is the use of column ordering based on the row indexes preventing the need for a transpose operation necessitating a second communication step when gathering the final force components. Force decomposition is less complex to implement than the spatial method but still scales computationally as $\mathcal{O}(N/P)$ and scales as $\mathcal{O}(N/\sqrt{P})$ in communication cost. Plimpton has also found that force decompositions scale more favorably than spatial decompositions for systems up to 10,000 atoms and favorably compete with spatial methods up to 100,000 atoms.[61]

Chapter 10

Conclusion

We have presented a new parallel simulation program called OOPSE. This program offers some novel capabilities, but mostly makes available a library of modern object-oriented code for the scientific community to use freely. Notably, OOPSE can handle symplectic integration of objects (atoms and rigid bodies) which have orientational degrees of freedom. It can also work with transition metal force fields and point-dipoles. It is capable of scaling across multiple processors through the use of force based decomposition. It also implements several advanced integrators allowing the end user control over temperature and pressure. In addition, it is capable of integrating constrained dynamics through both the RATTLE algorithm and the z-constraint method.

We encourage other researchers to download and apply this program to their own research problems. By making the code available, we hope to encourage other researchers to contribute their own code and make it a more powerful package for everyone in the molecular dynamics community to use. All source code for OOPSE is available for download at `http://oopse.org`.

Chapter 11

Acknowledgments

Development of OOPSE was funded by a New Faculty Award from the Camille and Henry Dreyfus Foundation and by the National Science Foundation under grant CHE-0134881. Computation time was provided by the Notre Dame Bunch-of-Boxes (B.o.B) computer cluster under NSF grant DMR-0079647.

Bibliography

[1] B. R. Brooks, R. E. Bruccoleri, B. D. Olafson, D. J. States, S. Swaminathan, and M. Karplus, *J. Comp. Chem.*, **4**, 187–217 (1983).

[2] A. D. MacKerell, Jr., B. Brooks, C. L. Brooks III, L. Nilsson, B. Roux, Y. Won, and M. Karplus In *The Encyclopedia of Computational Chemistry*, P. v. R. Schleyer, *et al.*, Ed., Vol. 1; John Wiley & Sons, New York, 1998; pages 271–277.

[3] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, and P. Kollman, *Comp. Phys. Comm.*, **91**, 1–41 (1995).

[4] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen, *Comp. Phys. Comm.*, **91**, 43–56 (1995).

[5] E. Lindahl, B. Hess, and D. van der Spoel, *J. Mol. Modelling*, **7**, 306–317 (2001).

[6] W. Smith and T. Forester, *J. Mol. Graphics*, **14**(3), 136–141 (1996).

[7] J. W. Ponder and F. M. Richards, *J. Comp. Chem.*, **8**(7), 1016–1024 (1987).

[8] S. J. Plimpton and B. A. Hendrickson In J. Broughton, P. Bristowe, and J. Newsam, Eds., *Materials Theory and Modelling*, Vol. 291 of *MRS Proceedings*, page 37, Pittsburgh, PA, 1993. Materials Research Society.

[9] L. Kalé, R. Skeel, M. Bhandarkar, R. Brunner, A. Gursoy, N. Krawetz, J. Phillips, A. Shinozaki, K. Varadarajan, and K. Schulten, *J. Comp. Phys.*, **151**, 283–312 (1999).

[10] F. Mohamadi, N. G. J. Richards, W. C. Guida, R. Liskamp, M. Lipton, C. Caufield, G. Chang, T. Hendrickson, and W. C. Still, *J. Comp. Chem.*, **11**, 440 (1990).

[11] H. Goldstein, C. Poole, and J. Safko, *Classical Mechanics*, Addison Wesley, San Francisco, 3rd ed., 2001.

[12] M. P. Allen and D. J. Tildesley, *Computer Simulations of Liquids*, Oxford University Press, New York, 1987.

[13] D. J. Evans, *Mol. Phys.*, **34**, 317–325 (1977).

[14] A. Leach, *Molecular Modeling: Principles and Applications*, Pearson Educated Limited, Harlow, England, 2nd ed., 2001.

[15] G. Cevc and D. Marsh, *Phospholipid Bilayers*, John Wiley & Sons, New York, 1987.

[16] C. J. Fennell and J. D. Gezelter, *J. Chem. Phys.*, **120**(19), 9175–9184 (2004).

[17] Y. Liu and T. Ichiye, *J. Phys. Chem.*, **100**, 2723–2730 (1996).

[18] M. Martin and J. I. Siepmann, *J. Phys. Chem. B*, **102**, 2569–2577 (1998).

[19] D. Bratko, L. Blum, and A. Luzar, *J. Chem. Phys.*, **83**(12), 6367–6370 (1985).

[20] L. Blum, F. Vericat, and D. Bratko, *J. Chem. Phys.*, **102**(3), 1461–1462 (1995).

[21] Y. Liu and T. Ichiye, *Chem. Phys. Lett.*, **256**, 334–340 (1996).

[22] A. Chandra and T. Ichiye, *J. Chem. Phys.*, **111**(6), 2701–2709 (1999).

[23] M.-L. Tan, J. T. Fischer, A. Chandra, B. R. Brooks, and T. Ichiye, *Chem. Phys. Lett.*, **376**, 646–652 (2003).

[24] G. Hura, J. M. Sorenson, R. M. Glaeser, and T. Head-Gordon, *J. Chem. Phys.*, **113**, 9140–9148 (2000).

[25] S. J. Marrink, E. Lindahl, O. Edholm, and A. E. Mark, *J. Am. Chem. Soc.*, **123**, 8638–8639 (2001).

[26] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, and J. Hermans In *Intermolecular Forces*, B. Pullman, Ed.; Reidel, Dordrecht, 1981; page 331.

[27] H. J. C. Berendsen, J. R. Grigera, and T. P. Straatsma, *J. Phys. Chem.*, **91**, 6269–6271 (1987).

[28] W. L. Jorgensen, J. Chandrasekhar, J. D. Madura, R. W. Impey, and M. L. Klein, *J. Chem. Phys.*, **79**, 926–935 (1983).

[29] M. W. Mahoney and W. L. Jorgensen, *J. Chem. Phys.*, **112**(20), 8910–8922 (2000).

[30] M. W. Finnis and J. E. Sinclair, *Phil. Mag. A*, **50**, 45–55 (1984).

[31] F. Ercolessi, M. Parrinello, and E. Tosatti, *Phil. Mag. A*, **58**, 213–226 (1988).

[32] A. P. Sutton and J. Chen, *Phil. Mag. Lett.*, **61**, 139–146 (1990).

[33] Y. Qi, T. Çağin, Y. Kimura, and W. A. Goddard III, *Phys. Rev. B*, **59**(5), 3527–3533 (1999).

[34] U. Tartaglino, E. Tosatti, D. Passerone, and F. Ercolessi, *Phys. Rev. B*, **65**, 241406 (2002).

[35] M. S. Daw and M. I. Baskes, *Phys. Rev. B*, **29**(12), 6443–6453 (1984).

[36] S. M. Foiles, M. I. Baskes, and M. S. Daw, *Phys. Rev. B*, **33**(12), 7983 (1986).

[37] R. A. Johnson, *Phys. Rev. B*, **39**(17), 12554 (1989).

[38] J. Lu and J. A. Szpunar, *Phil. Mag. A*, **75**, 1057–1066 (1997).

[39] A. Voter, *Intermetallic Compounds: Principles and Practice*, **1**, 77 (1995).

[40] M. S. Daw, *Phys. Rev. B*, **39**, 7441–7452 (1989).

[41] A. Voter and S. Chen, *Mat. Res. Soc. Symp. Proc.*, **82**, 175 (1987).

[42] R. T. Cygan, J.-J. Liang, and A. G. Kalinichev, *J. Phys. Chem. B*, **108**, 1255–1266 (2004).

[43] A. Dullweber, B. Leimkuhler, and R. McLachlan, *J. Chem. Phys.*, **107**(15), 5840–5851 (1997).

[44] D. Frenkel and B. Smit, *Understanding Molecular Simulation : From Algorithms to Applications*, Academic Press, New York, 1996.

[45] A. Kol, B. B. Laird, and B. J. Leimkuhler, *J. Chem. Phys.*, **107**(7), 2580–2588 (1997).

[46] W. G. Hoover, *Phys. Rev. A*, **31**, 1695 (1985).

[47] S. Melchionna, G. Ciccotti, and B. L. Holian, *Mol. Phys.*, **78**, 533–544 (1993).

[48] H. C. Andersen, *J. Comp. Phys.*, **52**, 24–34 (1983).

[49] J. P. Ryckaert, G. Ciccotti, and H. J. C. Berendsen, *J. Comp. Phys.*, **23**, 327–341 (1977).

[50] B. Roux and M. Karplus, *J. Phys. Chem.*, **95**(15), 4856–4868 (1991).

[51] S. J. Marrink and H. J. C. Berendsen, *J. Phys. Chem.*, **98**(15), 4155–4168 (1994).

[52] H. Grubmuller, B. Heymann, and P. Tavan, *Science*, **271**, 997–999 (1996).

[53] D. Frenkel and A. J. C. Ladd, *J. Chem. Phys.*, **81**(7), 3188–3193 (1984).

[54] J. Hermans, A. Pathiaseril, and A. Anderson, *J. Am. Chem. Soc.*, **110**, 5982–5986 (1988).

[55] E. J. Meijer, D. Frenkel, R. A. LeSar, and A. J. C. Ladd, *J. Chem. Phys.*, **92**(12), 7570–7575 (1990).

[56] L. A. Bàez and P. Clancy, *Mol. Phys.*, **86**(3), 385–396 (1995).

[57] M. J. Vlot, J. Huinink, and J. P. van der Eerden, *J. Chem. Phys.*, **110**(1), 55–61 (1999).

[58] R. Fletcher and C. M. Reeves, *Comput. J.*, **7**(2), 149–154 (1964).

[59] E. Polak and G. Ribiere, *Rev. Fr. Inform. Rech. Oper.*, **16-R1**, 35–43 (1969).

[60] G. C. Fox, M. A. Johnson, G. A. Lyzenga, S. W. Otto, J. K. Salmon, and D. W. Walker, *Solving Promblems on Concurrent Processors*, Vol. I, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[61] S. Plimpton, *J. Comp. Phys.*, **117**, 1–19 (1995).

[62] B. Hendrickson and S. Plimpton, *Journal of Parallel and Distributed Computing*, **27**, 15–25 (1995).