# CASSANDRA

# Computational Atomistic Simulation Software At Notre Dame for Research Advances

User Manual 1.2.4

3/2020

Written by:

Edward J. Maginn, Jindal K. Shah, Eliseo Marin-Rimoldi,
Brian P. Keene, Sandip Khan, Ryan Gotchy Mullen, Andrew Paluch, Neeraj Rai, Lucienne Romanielo,
Thomas Rosch, Brian Yoo, Jacob Gerace

# Preface and Disclaimer



This user manual is distributed along with the Cassandra software to aid in setting up various input files required for carrying out a Cassandra Monte Carlo simulation. Every effort is made to release the most updated and complete version of the manual when a new version of the software is released. To report any inconsistencies, errors or missing information, or to suggest improvements, send email to Edward Maginn (ed@nd.edu).

# Acknowledgements

Support for this work was provided by a grant from the National Science Foundation entitled "SI2-SSE: Development of Cassandra, a General, Efficient and Parallel Monte Carlo Multiscale Modeling Software Platform for Materials Research", grant number ACI-1339785.

Ed Maginn would like to acknowledge financial support from Sandia National Laboratory's Computer Science Research Institute, which enabled him to take a research leave and lay the foundation for Cassandra in collaboration with Jindal Shah, who stayed behind at Notre Dame and helped keep the group going. The hospitality of Steve Plimpton and co-workers at Sandia is gratefully acknowledged.

Finally, we would also like to thank the Center for Research Computing at Notre Dame, which provided support, encouragement, and infrastructure to help bring Cassandra to life.

People who have contributed to Cassandra through algorithm development and / or writing code (to date) include:

- Ed Maginn
- Jindal Shah
- Eliseo Marin
- Brian Keene
- Sandip Khan
- Ryan Gotchy Mullen
- Andrew Paluch
- Neeraj Rai
- Lucienne Romanielo
- Tom Rosch
- Brian Yoo

Some legacy code was used in the creation of Cassandra, and the following former students are recognized for their work:

- David Eike
- Jim Larentzos
- Craig Powers

# Contents

# Chapter 1

# Introduction

Cassandra is an open source Monte Carlo package capable of simulating any number of molecules composed of rings, chains, or both. It can be used to simulate compounds such as small organic molecules, oligomers, aqueous solutions and ionic liquids. It handles a standard "Class I"-type force field having fixed bond lengths, harmonic bond angles and improper angles, a CHARMM or OPLS-style dihedral potential, a Lennard-Jones 12-6 potential and fixed partial charges. It does *not* treat flexible bond lengths. Fused ring systems should be simulated with caution. Cassandra uses OpenMP parallelization and comes with a number of scripts, utilities and examples to help with simulation setup.

Cassandra is capable of simulating systems in the following ensembles:

- Canonical (NVT)
- Isothermal-Isobaric (NPT)
- Grand canonical ($\mu$VT)
- Constant volume Gibbs (NVT-Gibbs)
- Constant Pressure Gibbs (NPT- Gibbs)

## 1.1 Distribution

Cassandra is distributed through `conda-forge` and a source code tarball. Installation through conda is easiest:

```
> conda create --name cassandra -c conda-forge cassandra
```

The previous command creates a new conda environment (named "cassandra"), and installs `cassandra` from the `conda-forge` channel channel. You can verify that the installation completed by activating the environment:

```
> conda activate cassandra
```

and then checking that `cassandra.exe` is on your `PATH`:

```
> which cassandra.exe
```

should find the executable. `mcfgen.py` and `library_setup.py` will also be on your `PATH` when the `cassandra` conda environment is activated. If you choose to instead install from source, you can unpack the distribution by running the command

```
> tar -xzf Cassandra_V1.2.tar.gz
```

Upon successful unpacking of the archive file, the Cassandra_V1.2 directory will have a number of subdirectories. Please refer to the README file in the main Cassandra directory for a detailed information on each of the subdirectories.

`Documentation` - contains this user guide

`Examples` - contains example input files and short simulations of various systems in the above ensembles.

`MCF` - molecular connectivity files for a number of molecules. These can be used as the basis for generating your own MCF files for molecules of interest.

`Scripts` - useful scripts to set up simulation input files.

`Src` - Cassandra source code.

# Chapter 2

# Force Field

## 2.1 Bonds

Cassandra is designed assuming all bond lengths are fixed. If you wish to utilize a force field developed with flexible bond lengths, we recommend that you either use the nominal or "equilibrium" bond lengths of the force field as the fixed bond lengths specified for a Cassandra simulation or carry out an energy minimization of the molecule with a package that treats flexible bond lengths and utilize the bond lengths obtained from the minimization.

Table 2.1: Cassandra units for bonds

| Parameter | Symbol | Units |
|---|---|---|
| Bond length | l | Å |

## 2.2 Angles

Cassandra supports two types of bond angles:

- 'fixed' : The angle declared as fixed is not perturbed during the course of the simulation.
- 'harmonic' : The bond angle energy is calculated as

$$E_\theta = K_\theta(\theta - \theta_0)^2 \tag{2.1}$$

where the user must specify $K_\theta$ and $\theta_0$. Note that a factor of 1/2 is **not used** in the energy calculation of a bond angle. Make sure you know how the force constant is defined in any force field you use.

Table 2.2: Cassandra units for angles

| Parameter | Symbol | Units |
|---|---|---|
| Nominal bond angle | $\theta_0$ | degrees |
| Bond angle force constant | $K_\theta$ | K/rad$^2$ |

## 2.3 Dihedrals

Cassandra can handle four different types of dihedral angles:

- 'OPLS': The functional form of the dihedral potential is

$$E_\phi = a_0 + a_1\ (1 + \cos(\phi)) + a_2\ (1 - \cos(2\phi)) + a_3\ (1 + \cos(3\phi)) \tag{2.2}$$

  where $a_0$, $a_1$, $a_2$ and $a_3$ are specified by the user.

- 'CHARMM': The functional form of the potential is

$$E_\phi = a_0(1 + \cos(a_1\phi - \delta)) \tag{2.3}$$

  where $a_0$, $a_1$ and $\delta$ are specified by the user.

- 'harmonic': The dihedral potential is of the form:

$$E_\phi = K_\phi(\phi - \phi_0)^2 \tag{2.4}$$

  where $K_\phi$ and $\phi_0$ are specified by the user.

- 'none' : There is no dihedral potential between the given atoms.

## 2.4 Impropers

Improper energy calculations can be carried out with the following two options:

Table 2.3: Cassandra units for dihedrals

| Functional Form | Parameter | Units |
|---|---|---|
| OPLS | $a_0$, $a_1$, $a_2$, $a_3$ | kJ/mol |
| CHARMM | $a_0$ | kJ/mol |
| | $a_1$ | dimensionless |
| | $\delta$ | degrees |
| harmonic | $K_\phi$ | K/rad$^2$ |
| | $\phi_0$ | degrees |

- 'none': The improper energy is set to zero for the improper angle.
- 'harmonic': The following functional form is used to calculate the energy due to an improper angle

$$E_\psi = K_\psi \left(\psi - \psi_0\right)^2 \tag{2.5}$$

where $K_\psi$ and $\psi_0$ are specified by the user.

Table 2.4: Cassandra units for impropers

| Parameter | Symbol | Units |
|---|---|---|
| Force constant | $K_\psi$ | K/rad$^2$ |
| Improper | $\psi_0$ | degrees |

## 2.5 Nonbonded

The nonbonded interactions between two atoms $i$ and $j$ are due to repulsion-dispersion interactions and electrostatic interactions (if any).

### 2.5.1 Repulsion-Dispersion Interactions

The repulsion-dispersion interactions can take one of the following forms:

- Lennard-Jones 12-6 potential (LJ):

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{2.6}$$

  where $\epsilon_{ij}$ and $\sigma_{ij}$ are the energy and size parameters set by the user. For unlike interactions, different combining rules can be used, as described elsewhere. Note that this option only evaluates the energy up to a specified cutoff distance. As described below, analytic tail corrections to the pressure and energy can be specified to account for the finite cutoff distance.

- Cut and shift potential:

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] - 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{cut}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{cut}}\right)^{6}\right] \tag{2.7}$$

  where $\epsilon_{ij}$ and $\sigma_{ij}$ are the energy and size parameters set by the user and $r_{cut}$ is the cutoff distance. This option forces the potential energy to be zero at the cutoff distance. For unlike interactions, different combining rules can be used, as described elsewhere.

- Cut and switch potential:

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] f \tag{2.8}$$

  The factor $f$ takes the following values:

$$f = \begin{cases} 1.0 & r_{ij} \leq r_{on} \\ \frac{(r_{off}^2 - r_{ij}^2)^2 (r_{off}^2 - 3r_{on}^2 + 2r_{ij}^2)}{\left(r_{off}^2 - r_{on}^2\right)^3} & r_{on} < r_{ij} < r_{off} \\ 0.0 & r_{ij} \geq r_{off} \end{cases} \tag{2.9}$$

  where $\epsilon_{ij}$ and $\sigma_{ij}$ are the energy and size parameters set by the user. This option smoothly forces the potential to go to zero at a distance $r_{off}$, and begins altering the potential at a distance of $r_{on}$. Both of these parameters must be specified by the user. For unlike interactions, different combining rules can be used, as described elsewhere.

- Shifted force LJ potential:

$$\mathcal{V}_{SF}(r_{ij}) = \mathcal{V}_{LJ}(r_{ij}) - \mathcal{V}_{LJ}(r_{cut}) - (r_{ij} - r_{cut}) \frac{d\mathcal{V}(r_{ij})_{LJ}}{dr_{ij}}\Big|_{r_{cut}} \tag{2.10}$$

  where $\mathcal{V}_{LJ}$ is the standard Lennard-Jones potential (Eqn. 2.6) and $r_{cut}$ is the cutoff distance. The energy and force both smoothly go to zero at $r_{cut}$.

- Mie potential (generalized form of LJ):

$$\mathcal{V}(r_{ij}) = \left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{\frac{m}{n-m}} \epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{n} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{m}\right] \tag{2.11}$$

  where $\epsilon_{ij}$ and $\sigma_{ij}$ are the energy and size parameters and $n$ and $m$ are the repulsive and attractive exponents set by the user. This option allows for the use of a generalized LJ potential (for LJ, $n = 12$ and $m = 6$). Note that this option only evaluates the energy up to a specified cutoff distance. Both n and m can take on separate integer or float values set by the user. For unlike interactions, different combining rules can be used, as described elsewhere.

- Mie cut and shift potential:

$$\mathcal{V}(r_{ij}) = \left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{\frac{m}{n-m}} \epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{n} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{m}\right] - \left(\frac{n}{n-m}\right)\left(\frac{n}{m}\right)^{\frac{m}{n-m}} \epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{cut}}\right)^{n} - \left(\frac{\sigma_{ij}}{r_{cut}}\right)^{m}\right] \tag{2.12}$$

  where $\epsilon_{ij}$ and $\sigma_{ij}$ are the energy and size parameters and $n$ and $m$ are the repulsive and attractive exponents set by the user. This option forces the potential energy to be zero at the cutoff distance (i.e. setting $n = 12$ and $m = 6$ provides the same potential as the LJ cut and shift option). For unlike interactions, different combining rules can be used, as described elsewhere.

- Tail corrections: If the Lennard-Jones potential is used, standard Lennard-Jones tail corrections are used to approximate the long range dispersion interactions

Table 2.5: Cassandra units for repulsion-dispersion interactions

| Parameter | Symbol | Units |
|---|---|---|
| Energy parameter | $\epsilon/k_B$ | K |
| Collision diameter | $\sigma$ | Å |

### 2.5.2 Electrostatics

Electrostatic interactions are given by Coulomb's law

$$\mathcal{V}_{elec}(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}. \tag{2.13}$$

where $q_i$ and $q_j$ are the partial charges set by the user, which are placed on atomic positions given by $r_i$ and $r_j$. In a simulation, the electrostatic interactions are calculated using either an Ewald summation,

the Damped Shifted Force method [1], or a direct summation using the minimum image convention. Note that the total energy that is printed out in the property file is extensive. Consequently, to obtain intensive energies, the printed energies must divided by the total number of molecules in the system.

Table 2.6: Cassandra units for coulombic interactions

| Parameter | Symbol | Units |
|---|---|---|
| Charge | $q$ | e |

Table 2.7: Summary of Cassandra units for input variables

| | | |
|---|---|---|
| Bond length | l | Å |
| **Angles** | | |
| Nominal bond angle | $\theta_0$ | degrees |
| Bond angle force constant | $K_\theta$ | K/rad$^2$ |
| **Dihedral angle** | | |
| OPLS | $a_0$, $a_1$, $a_2$, $a_3$ | kJ/mol |
| CHARMM | $a_0$ | kJ/mol |
| | $a_1$ | dimensionless |
| | $\delta$ | degrees |
| harmonic | $K_\phi$ | K/rad$^2$ |
| | $\phi_0$ | degrees |
| **Improper angle** | | |
| Force constant | $K_\psi$ | K/rad$^2$ |
| Improper angle | $\psi_0$ | degrees |
| **Nonbonded** | | |
| Energy parameter | $\epsilon/k_B$ | K |
| Collision diameter | $\sigma$ | Å |
| Charge | q | e |
| **Simulation Parameters** | | |
| Simulation box length | | Å |
| Volume | | Å$^3$ |
| Distances | | Å |
| Rotational width | | degrees |
| Temperature | | K |
| Pressure | | bar |
| Chemical potential | | kJ/mol |
| Energy | | kJ/mol |

# Chapter 3

# Cassandra Basics

## 3.1 Flow Diagram

A flow diagram that overviews the setup for a Cassandra simulation is displayed on figure 3.1. This diagram employs two automation scripts located in the `/Scripts/` directory: `mcfgen.py` and `library_setup.py`. These scripts are particularly useful when simulating large molecules. For details about how to use them, please refer to sections 5.1 and 5.2 of this user guide, and to the README files located in the subdirectories inside the directory `/Scripts/`.

## 3.2 Cassandra Simulation Setup

Once a system is identified, setting up a Cassandra simulation from scratch requires preparation of the following files.

- A molecular connectivity file (MCF) (*.mcf) containing the molecular connectivity information on bonds, angles, dihedrals, impropers and whether the molecule is composed of fragments. For information on the MCF file, please refer to section 4.2.
- An input file (*.inp) (see section 4.1)
- If the molecule is composed of fragments, then a fragment library file for each of the fragments is required. For instructions on how to generate these files, please refer to the section 5.2.

MCF files for united-atom models of methane, isobutane, dimethylhexane, cyclohexane and diethylether are

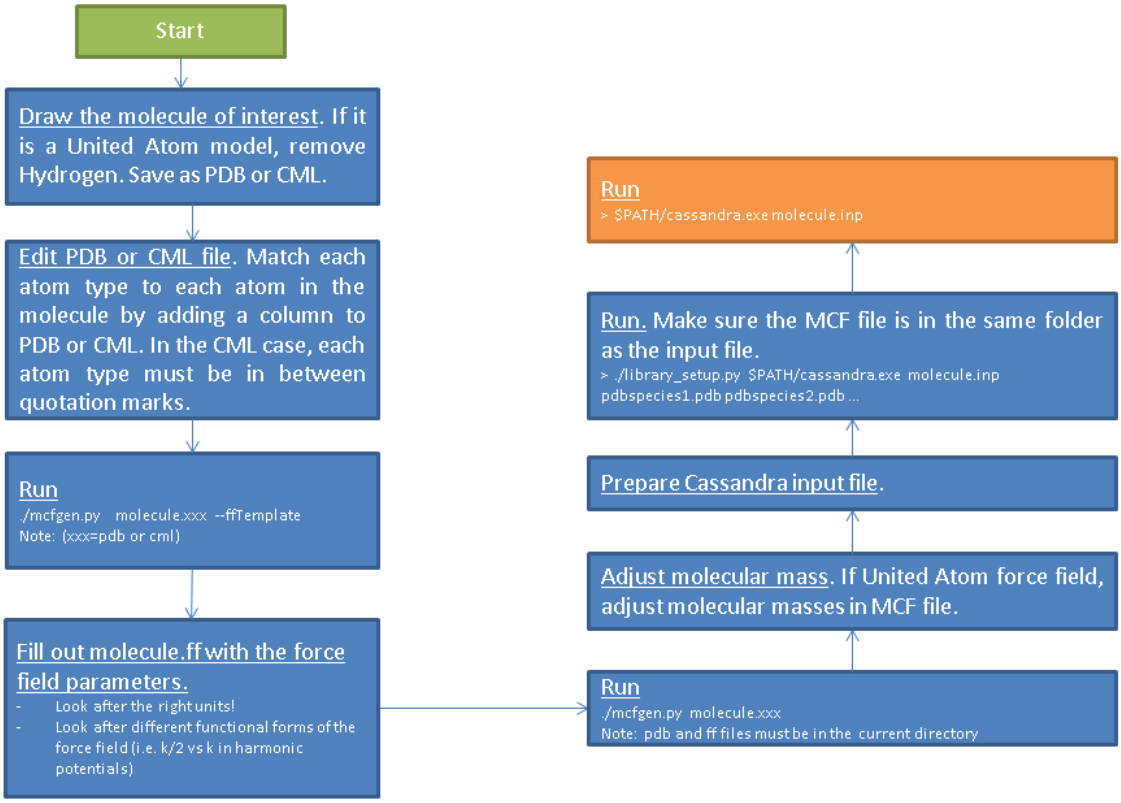


Figure 3.1: Flow diagram representing a typical setup of a Cassandra simulation

provided in the `MCF` directory. Input files for NVT, NPT, GCMC and GEMC ensembles are located in the `Examples` directory which also contains fragment library files for a number of molecules simulated in these ensembles.

## 3.3 Cassandra File Preparation

### 3.3.1 MCF File

One MCF file is required for each unique species in a simulation. A species is defined as a collection of atoms associated with each other through bonds. Thus a molecule is a species as is an ion. If you wanted to simulate sodium sulfate, you would need separate MCF files for the sodium ion and the sulfate ion. MCF files can be created manually or by using the scripts provided with the code, as described in the section 5.1. Instructions for generating an MCF file can also be found in the `Scripts/MCF_Generation/README` file. We will collect MCF files submitted to us by users and will post them on the Cassandra website http://cassandra.nd.edu. If you have an MCF file you would like us to post, send it to ed@nd.edu.

### 3.3.2 Input File

An input file is required for a Cassandra simulation. The input file specifies conditions for the simulation and various keywords required for the simulation in a given ensemble. Please refer to Chapter 4.1 for further details.

### 3.3.3 Fragment Library Generation

Cassandra makes use of reservoir sampling schemes to correctly and efficiently sample the various coupled intramolecular degrees of freedom associated with branch points and rings. For more information, please see Ref. [2]. The molecule is decomposed into fragments that are either branch points or ring groups, each coupled to other fragments via a single dihedral angle. Thus, the total number of fragments of a molecule is the sum of branch points and ring groups in the molecule. The neighboring fragments are connected by two common atoms present in each of the fragments. Note that the ring group contains all the ring atoms and those directly bonded to the ring atoms. For each fragment identified, Cassandra runs a pre–simulation in the gas phase to sample the intramolecular degrees of freedom. A library of a large number of these conformations are stored for use in an actual simulation. *Note that intramolecular degrees of freedom may not be properly sampled in fused-ring systems. We highly encourage anyone simulating such systems to compare the internal angle and dihedral distributions in the fragment library with a constrained bond length molecular dynamics simulation of the relevant fragment in vacuum.*

The gas phase library generation has been automated with the script `library_setup.py` located in the `Scripts/Frag_Library_Setup` directory. Use the following command for generating the fragment library.

```
> python $PATH/Frag_Library_Setup/library_setup.py $PATH/Src/cassandra_executable
input_filename mol1.pdb (mol1.cml) mol2.pdb (mol2.cml) ...
```

where `input_filename` is the name of the input file for the actual simulation and `mol1.pdb mol2.pdb ...` or `mol1.cml mol2.cml ...` correspond to the names of the pdb (or cml) files used to generated the MCF files. Make sure that if a file does not exist in the current working directory, its path relative to the current working directory is specified.

## 3.4 Running a Simulation

To launch a Cassandra simulation, run the following command:

```
> cassandra_executable input_filename
```

The executable will read `input_filename` and execute the instructions. Make sure that the required files (MCF, fragment library files) are located in the directories as given in the input file.

## 3.5 Restarting a Simulation

Restarting a simulation requires either a checkpoint file (*.chk produced by Cassandra) or a configuration file obtained from xyz files generated from a previous simulation. Please refer to Section 4.1.16 to find information about the keywords *checkpoint* and *read_config*.

## 3.6 Cassandra Output Files

Cassandra generates several output files which can be used for later analysis. All have as a prefix the Run_Name specified in the input file. See Chapter 4.1 for details. The type of output is specified by the file name suffix. The following are generated:

- **Log file** (*.log): Contains basic information on what the run is, timing information and reports the various parameters specified by the user. A complete copy of the input file is reproduced. Other important information includes the move acceptance rates. You can use the log file to keep track of what conditions were simulated.
- **Coordinate file** (*.xyz or *.box#.xyz): For each box in the system, a set of xyz coordinates are written out with a frequency specified by the user (Coord_Freq). The file has as a header the number of atoms in the box. Following this, the atomic coordinates of molecule 1 of species 1 are written, then the coordinates of molecule 2 of species 1 are written, etc. After all the coordinates of the molecules of species 1 are written, the coordinates of the molecules of species 2 are written, etc. You can use this

file to do all your structural analysis and post processing.

Note that if you generate your initial configuration using the make_config command, the first "snapshot" of the coordinate file will contain the initial configuration of all the species in the system for a given box. You can use this configuration to check on whether the initial configuration is reasonable, or use it as an input to other codes. Note that the initial configuration will be generated using a configurational biased scheme, so it may be a better starting configuration than if you used other methods.

- **Checkpoint file** (*.chk): A checkpoint file is written every Coord_Freq steps. This can be used to restart a simulation from this point using all of the same information as the run that was used to generate the checkpoint file. To do this, you must use the checkpoint restart option (see Chapter 4.1). It will basically pick up where the simulation left off, using the same random number seed, maximum displacements, etc. This is useful in case your job crashes and you want to continue running a job. You can also use the checkpoint file to start a new simulation using the configuration of the checkpoint file as an initial configuration and the optimized maximum displacements. To do this, use the script read_old.py. You will need to set a new random number seed if you do this. See the documentation in Chapter 4.1 for more details.

- **H-matrix file** (*.H or *.box#.H): This file is written to every Coord_Freq MC steps. The first line is the box volume in angstrom$^3$. The next three lines are the box coordinates in angstrom in an H-matrix form. Since Cassandra only supports cubic boxes at the moment, this is just a diagonal and symmetric matrix, but is included here for later versions that will enable non-orthogonal boxes. After this, a blank line is written. The next line is the box number, and the final line(s) is(are) the species ID and number of molecules for that species in this box. If there are three species, there will be three lines. This output is repeated every Coord_Freq times. This file allows you to compute the density of the box during constant pressure simulations.

- **Property file** (*.prp# or *.box#.prp#): This file lists the instantaneous thermodynamic and state properties for each box. Note that you can have more than one property file (hence the# after 'prp') and more than one box (also why there is a # after 'box'). The user specifies which properties are to be written and in what order, and these are then reproduced in this file. The file is written to every Prop_Freq steps. A header is written to the first two lines to designate what each property is. You may use this file to compute thermodynamic averages.

# Chapter 4

# Files Required to Run Cassandra

## 4.1 Simulation Input File

This is a required file that is given as an argument to the Cassandra executable. Example input files for each ensemble are provided in the Examples directory that can be modified for new simulations. The input file is divided into sections. Each section begins with a section header that starts with a '#', e.g. `# Run_Name`, and ends with a blank line. Section `# Move_Probability_Info` is an exception and terminates with `# Done_Move_Probability_Info`, because subsections e.g. `# Prob_Translation` are separated by blank lines. Comment lines begin with '!' and are ignored. Sections in the input file can be listed in any order, but the order and format of keywords and parameters given in each section are important unless otherwise noted below. Previously, some keywords were capitalized, e.g. `CUBIC`, some contained an initial capital, e.g. `Units`, and some were all lowercase, e.g. `kappa_ins`. New in version 1.2, all keywords are supported in lowercase text; each word in a section header must still begin with an initial capital.

### 4.1.1 Run Name

**# Run_Name**
*Character*

The run name is specified on the next line following the keyword. This name is used as a prefix for all the files produced by the simulation. For example,

```
# Run_Name
dee.out
```

Cassandra will then use `dee.out` as prefix for all output files created.

### 4.1.2 Simulation Type

**# Sim_Type**
*Character*

Sets the ensemble (and thus the suite of moves) of a Cassandra simulation. The following ensembles are supported:

- nvt or nvt_mc (canonical ensemble)
- npt or npt_mc (isothermal-isobaric ensemble)
- gcmc (grand canonical ensemble)
- gemc (Gibbs ensemble)
- gemc_npt (Multi-species Gibbs ensemble)
- nvt_min (canonical ensemble, only moves which lower the energy are accepted)
- fragment or nvt_mc_fragment (canonical ensemble simulation of a fragment)
- ring_fragment or nvt_mc_ring_fragment (canonical ensemble simulation of a ring fragment)

Simulation types `fragment` and `ring_fragment` are used only for generating a fragment library. For example,

```
# Sim_Type
npt
```

will run a Monte Carlo simulation in the isothermal-isobaric ensemble in which the number of molecules of each species $N$, the pressure $P$ and temperature $T$ are held constant.

### 4.1.3 Number of species

**# Nbr_Species**
*Integer*

Total number of species in the simulation. For ionic systems, each ion is counted as a separate species. For example, for a mixture of two species, use the following:

```
# Nbr_Species
2
```

### 4.1.4 VDW Style

**# VDW_Style**
*Character(i,1)* [*Character(i,2) Real(i,3) Real(i,4)/Logical(i,4)*]

This keyword specifies the functional form of repulsion dispersion interactions to be used and if tail corrections are added for box $i$. One line is required for each box. *Character(i,1)* specifies the van der Waals model and can be `lj` for a Lennard-Jones 12-6 potential, `mie` for a Mie potential, or `none` to turn off all repulsion-dispersion interactions. *Character(i,2)* and *Real(i,3)* are required for `lj` or `mie`. *Character(i,2)* specifies how the Lennard-Jones potential is truncated. Options are `cut`, `cut_tail`, `cut_switch`, or `cut_shift`. Refer to Chapter 2 for the functional forms. The other parameters *Real(i,3)* and *Real(i,4)/Logical(i,4)* depend on the selection of *Character(i,2)* as described below:

- **cut**: This option cuts the potential at the distance specified by *Real(i,3)*. The fourth parameter is omitted.

  For example, to simulate one box with a 14 Å cutoff specify the following:

  ```
  # VDW_Style
  lj cut 14.0
  ```

  Similarly, for a two box simulations such as used in the Gibbs ensemble where both boxes have a 14 Å cutoff, use the following:

  ```
  # VDW_Style
  lj cut 14.0
  lj cut 14.0
  ```

- **cut_tail**: This options cuts the potential off at a distance corresponding to *Real(i,3)* and applies analytic tail corrections to the energy and pressure. An optional fourth argument *Logical(i,4)* can be set to 'true', in which case *Real(i,3)* is ignored and the cutoff distance is always set to half of the simulation box length. The cutoff will change during the course of the simulation when attempting volume moves. This option is provided to enable reproduction of literature simulations that use a cut off distance of half the simulation box length, but its use is discouraged.

  For example, to simulate one box with a 14 Å cutoff using tail corrections, specify the following:

  ```
  # VDW_Style
  lj cut_tail 14.0
  ```

  For a two box simulation where the first box has a 14 Å cutoff and the second one has a 20 Å cutoff, use the following:

```
# VDW_Style
lj cut_tail 14.0
lj cut_tail 20.0
```

- **cut_switch**: This option cuts the potential off and smoothly brings the potential to zero using a spline. The potential is cutoff and the spline turned on at a distance specified by *Real(i,3)* ($r_{on}$ in Eq 2.8) and the potential goes to zero at a distance specified by *Real(i,4)* ($r_{off}$ in Eq 2.8).

  For example, a one box simulation using the cut_switch option could be specified as follows:

  ```
  # VDW_Style
  lj cut_switch 12.0 14.0
  ```

  In this case, the Lennard-Jones potential would end at 12.0 Å and be smoothly taken to zero at 14.0 Å. $r_{on} < r_{off}$ or *Real(i,3)* < *Real(i,4)*.

- **cut_shift**: This option cuts the potential off at a distance specified by *Real(i,3)* and shifts the entire potential so that at this distance the potential is zero. The fourth parameter *Real(i,4)/Logical(i,4)* is omitted. The functional form of this potential is given in eq 2.7.

  To perform a two box simulation with a cut_shift option in which both boxes have a 10.5 Å cut-off, use the following:

  ```
  # VDW_Style
  lj cut_shift 10.5
  lj cut_shift 10.5
  ```

- **cut_shift-force**: This option cuts the potential off at a distance specified by *Real(i,3)*. The derivative of the potential is shifted to zero at the cutoff. The result is that both the potential and derivative of the potential smoothly go to zero at the cutoff. The fourth parameter *Real(i,4)/Logical(i,4)* is omitted. The functional form of this potential is given in eq 2.10.

  To perform a two box simulation with a cut_shift-force option in which both boxes have a 10.5 Å cutoff, use the following:

  ```
  # VDW_Style
  lj cut_shift-force 10.5
  lj cut_shift-force 10.5
  ```

**Note: For all options, cutoff distances must be less than or equal to the shortest edge length of a simulation box**.

### 4.1.5 Charge Style

**# Charge_Style**
*Character(i,1)* [*Character(i,2) Real(i,3) Real(i,4)*]

Cassandra allows the use of fixed partial charges on atomic centers using a Coulomb potential of the form given in Eq 2.13. If this section is missing from the input file, the electrostatic energy of the simulation will not be computed. If you do not wish to use a Coulomb potential for box *i*, set *Character(i,1)* to `none`. If `none` is selected for *Character(i,1)* then *Character(i,2)*, *Real(i,3)* and *Real(i,4)* are omitted.

For example,

```
# Charge_Style
none
```

should be used if you have no partial charges and are simulating a single box (or the section can just be omitted).

To compute the electrostatic energy for box *i*, this section must be included and *Character(i,1)* set to `coul`. For this option, *Character(i,2)* can be set to `ewald` if you want to use an Ewald sum to compute Coulombic interactions, `dsf` if you want to use the Damped Shifted Force method by Fennell *et al.*[1], or it can be set to `cut`, in which case the Coulombic interactions will be cut off and the long range interactions ignored. For the Ewald option, *Real(i,3)* is the real space cutoff distance and *Real(i,4)* specifies the accuracy of the Ewald summation. A reasonable value for the accuracy is $10^{-5}$. Note that the number of reciprocal vectors for the Ewald summation is determined in the code based on the accuracy parameter. For more details, see the paper by Fincham [3].

For example,

```
# Charge_Style
coul ewald 12.0 1E-5
```

will use the Ewald sum for a single box. The real space cutoff will be 12 Å and the accuracy will be $10^{-5}$. If you have two boxes, like in a Gibbs ensemble calculation, then you could use the following:

```
# Charge_Style
coul ewald 12.0 1E-5
```

```
coul ewald 30.0 1E-5
```

This will use an Ewald sum for both boxes. In the first box, the real space cutoff will be 12 Å while in the second box a larger cutoff of 30 Å will be used. **Note: When performing Gibbs ensemble simulations of vapor-liquid equilibria, the vapor box is often much larger than the liquid box. In this case, you will want to use a longer real space cutoff for the larger vapor box to avoid using too many reciprocal space vectors.** Also note that the real space cutoffs must always be less than or equal to half of the shortest edge length of a simulation box.

If you wish to use the Damped Shifted Force method, the entry *Real(i,3)* is the electrostatic energy cutoff distance and *Real(i,4)* is an optional entry to specify the damping parameter. If not specified, Cassandra will set this value algorithmically from the cutoff radius. For example,

```
# Charge_Style
coul dsf 12.0 0.20
```

will use the Damped Shifted Force method for a single box. The electrostatic energy cutoff will be set to 12 Å and the damping parameter will be set to 0.20, which is a reasonable value for typical liquid phase simulations.

**Note**: If the cutoff in VDW_Style is set to half of the simulation box length, any cutoff distance specified in the Charge_Style section will default to the half of the simulation box length. In the case of Ewald summation, however, the accuracy will be the same as *Real(i,4)*.

### 4.1.6 Mixing Rule

**# Mixing_Rule**
*Character*

Sets the method by which van der Waals interactions between unlike atoms are calculated. Acceptable options are `lb` for Lorentz-Berthelot, `geometric` for geometric mixing rule and `custom` for allowing the user to provide specific values. To use either `lb` or `geometric` keywords with the Mie potential, all atomtypes must have the same repulsive and dispersive exponents. If this section is missing, `lb` is used as default.

To illustrate the use of the `custom` option, consider a mixture of methane (species 1) and butane (species 2) united atom models using a Lennard-Jones potential. Methane has a single atomtype, CH4. Butane has two atomtypes: pseudoatoms 1 and 4 are type CH3, pseudoatoms 2 and 3 are type CH2. The cross interaction table is as follows:

```
 # Mixing_Rule
custom
CH4 CH3 120.49 3.75
```

```
CH4 CH2 82.51 3.83
CH3 CH2 67.14 3.85
```

The order in which atom types are listed is unimportant, but the atom types must match exactly the types given in each MCF. The Lennard-Jones potential requires two parameters: an energy parameter with units K, and a collision diameter with units Å. The Mie potential requires four parameters: an energy parameter with units K, a collision diameter with units Å, a repulsive exponent, and a dispersive exponent.

### 4.1.7 Starting Seed

**# Seed_Info**
*Integer(1) Integer(2)*

Inputs for the starting random number seeds for the simulation. Cassandra uses a random number generator proposed by L'Ecuyer [4], which takes five seeds to calculate a random number, out of which three are defined internally while two *Integer(1)* and *Integer(2)* are supplied by the user. **When a 'checkpoint' file is used to restart a simulation (see # Start_Type below), the user supplied seeds will be overwritten by those present in the checkpoint file. If # Start_Type is set to 'read_config', then the seeds specified in the input file are used**.

As an example,

```
# Seed_Info
1244432 8263662
```

is an acceptable way of specifying the seeds. Note that two independent simulations can be run using the same input information if different seeds are used. If two simulations having exactly the same input information and the same seeds are run, the results will be identical.

### 4.1.8 Minimum Cutoff

**# Rcutoff_Low**
*Real*

Sets the minimum allowable distance in Å between two atoms. Any MC move bringing two sites closer than this distance will be immediately rejected. It avoids numerical problems associated with random moves that happen to place atoms very close to one another such that they will have unphysically strong repulsion or attraction. This distance must be less than the intramolecular distance of all atoms in a species which are not bonded to one another. For models that use dummy sites without explicitly defining bonds between dummy and atomic sites of the molecules (for example, the TIP4P water model), it is important that the minimum distance is set to be less than the shortest distance between any two sites on the molecule. For

most systems, 1 Å seems to work OK, but for models with dummy sites, a shorter value may be required.

### 4.1.9 Pair Energy Storage

**# Pair_Energy**
*Logical*

Cassandra can use a time saving feature in which the energies between molecules are stored and used during energy evaluations after a move, thereby saving a loop over all molecules. This requires more memory, but it can be faster. The default is to not use this feature. If you wish to use this, set *Logical* to 'true'.

### 4.1.10 Molecule Files

**# Molecule_Files**
*Character(i,1) Integer(i,2)*

This specifies the name of the molecular connectivity file (MCF) and the maximum total number of molecules of a given species specified by this MCF. A separate line is required for each species present in the simulation. *Character(i,1)* is the name of the MCF for species *i*. *Integer(i,2)* is the maximum number of molecules expected for the species.

For example

```
# Molecule_Files
butane.mcf 100
hexane.mcf 20
octane.mcf 5
```

specifies that there are three different species, and the MCFs state the names of the files where information on the three species can be found. Species 1 is butane, species 2 is hexane and species 3 is octane. There can be a maximum of 100 butane molecules, 20 hexane molecules and 5 octane molecules in the total system. The maximum number of molecules specified here will be used to allocate memory for each species, so do not use larger numbers than are needed.

### 4.1.11 Simulation Box

**# Box_Info**
*Integer(1)*
*Character(i)*
*Real(i,1)* [*Real(i,2) Real(i,3)*]

[`restricted_insertion` *Character(1) Real(1) [Real(2)]*]

This section sets parameters for the simulation boxes. *Integer(1)* specifies the total number of boxes in the simulation. Gibbs ensemble simulations must have two boxes. *Character(i)* is the shape of the $i$th simulation box. The supported keywords are `cubic`, `orthogonal`, and `cell_matrix`.

If *Character(i)* is `cubic`, *Real(i,1)* is the length of the box edges in Å. Information for additional boxes is provided in an analogous fashion and is separated from the previous box by a blank line. For a two box simulation, box information is given as:

```
# Box_Info
2
cubic
30.0

cubic
60.0
```

This will construct a 30 x 30 x 30 Å cube and the second a 60 x 60 x 60 Å cube.

The options orthogonal and cell_matrix are only supported for constant volume simulations (i.e. NVT or GCMC) which only have 1 box. If *Character(1)* is `orthogonal`, *Real(1,1) Real(1,2) Real(1,3)* are the length, width and height that define the simulation box. For example,

```
# Box_Info
1
orthogonal
30.0 35.0 40.0
```

This will create a simulation box with dimensions 30.0 x 35.0 x 40.0 Å.

A non-orthogonal box is created by setting *Character(1)* to `cell_matrix`. In this case, three basis vectors are needed to define the simulation box. Each vector is entered as a column of a 3x3 matrix. For example,

```
# Box_Info
1
cell_matrix
30  0  0
 0 35  0
 0  2 40
```

defines a simulation box with basis vectors (30, 0, 0), (0, 35, 2) and (0, 0, 40).

The optional keyword `restricted_insertion` is used to define a region inside the simulation box in which molecules will be inserted at start-up via `make_config` or `add_to_config` or throughout the simulation via

grand canonical insertion moves or Gibbs ensemble swap moves. `restricted_insertion` takes one of several options: `sphere`, `cylinder`, `slitpore`, `interface`. Each option requires additional parameters, as follows.

- `sphere` $R$, where $R$ is the radius of a sphere centered at the origin
- `cylinder` $R$, where $R$ is the radius of a cylinder centered on the z-axis
- `slitpore` $z_{max}$, where $z_{max}$ is half the height of a rectangular prism centered on the $xy$-plane
- `interface` $z_{min}$ $z_{max}$, defines two rectangular prisms that span the box in the $x$ and $y$ directions. One box has bounds $z_{min} < z < z_{max}$ and the other has bounds $-z_{max} < z < -z_{min}$.

For example, to make a spherical droplet with a radius of 5 Å in cubic box 100 Å

```
 # Box_Info
1
cubic
100
restricted_insertion sphere 5.0
```

In addition, the insertion method for each species must be identified in the `Start_Type` or `Move_Probability_Info` sections.

### 4.1.12 Temperature

**# Temperature_Info**
*Real(i)*

*Real(i)* is the temperature in Kelvin for box $i$. For GEMC, the temperature of box 2 will be read from a second line:

```
# Temperature_Info
300.0
300.0
```

### 4.1.13 Pressure

**# Pressure_Info**
*Real(i)*

*Real(i)* is the pressure setpoint in bar for box $i$. For GEMC, the pressure of box 2 will be read from a second line:

```
# Pressure_Info
1.0
1.0
```

If the simulation type does not require an input pressure (e.g., NVT), this section will be ignored.

### 4.1.14 Chemical Potential

**# Chemical_Potential_Info**
*Real(1) ... Real(n)*

where $n$ is the number of insertable species and *Real(i)* is the chemical potential setpoint (shifted by a species-specific constant) of insertable species $i$ in kJ/mol. Each chemical potential will be assigned in the order species appear in the `Molecule_Files` section. For species with insertion method `none`, the chemical potential can be listed as `none` or omitted. This section is only read for grand canonical simulations. See Eq. (7.32) for more information. For example, the adsorption of methane (species 2) in a zeolite (species 1) can be computed by inserting methane molecules into a box with a zeolite crystal. In this example, only one chemical potential (for methane) is required and the following are equivalent:

```
# Chemical_Potential_Info
-35.0
```

```
# Chemical_Potential_Info
none -35.0
```

### 4.1.15 Move Probabilities

**# Move_Probability_Info**
**[subsections]**
**# Done_Probability_Info**

This section specifies the probabilities associated with different types of MC moves to be performed during the simulation. The section begins with the header `# Move_Probability_Info` and is terminated by the footer `# Done_Probability_Info`. All the move probability subsections must be between the section header and footer.

If the move probabilities do not sum to 1.0, then the probability of each move will be divided by the total.

### Translation

**# Prob_Translation**
*Real(1)*
*Real(i,1) ... Real(i,n)* *One line required for each box $i$

where $n$ is the number of species. *Real(1)* is the probability of performing a center of mass translation move. *Real(i,j)* is the maximum displacement in Å of species $j$ in box $i$. This subsection is optional in all ensembles.

For example, if you have three species and two boxes, you could specify the translation probability as

```
# Prob_Translation
0.25
2.0 2.5 1.0
12.0 12.0 12.0
```

This will tell Cassandra to attempt center of mass translations 25% of the total moves. For box 1, the maximum displacement will be 2.0 Å for species 1, 2.5 Å for species 2, and 1.0 Å for species 3. For box 2, the maximum displacement for all species is 12.0 Å.

For a simulation that involves solid frameworks, set the maximum displacement of the solid species to zero. Every molecule in the simulation with a maximum displacement greater than zero has an equal chance of being moved.

### Rotation

**# Prob_Rotation**
*Real(1)*
*Real(i,1) ... Real(i,n)* *One line required for each box $i$

where $n$ is the number of species. The probability of performing a rotation move is specified by *Real(1)* while *Real(i,j)* denotes the maximum rotation for species $j$ in box $i$ in degrees about the x, y or z-axis. The axis will be chosen with uniform probability. This subsection is optional for all ensembles.

For example, if you are simulating a single species in two boxes, you could specify the rotational probability as

```
# Prob_Rotation
0.25
30.0
```

```
180.0
```

Twenty-five percent of the attempted moves will be rotations. Molecules in box 1 will be rotated a maximum of 30° around the x, y, or z-axis. Molecules in box 2 will be rotated a maximum of 180° around the x, y, or z-axis.

If all species are point particles (such as single-site Lennard-Jones particles), this section should be omitted. For a multi-species system, set *Real(i,j)* to zero for point particles and solid frameworks.

Linear molecules are a special case. A molecule is identified as linear if all angles in the MCF are fixed at 180°. If a linear molecule were aligned with the axis of rotation, then the molecular orientation would not be changed. Therefore, linear molecules are rotated by choosing a random unit vector with uniform probability without regard to the molecule's current orientation or the maximum rotation. As with non-linear molecules, if *Real(i,j)* is zero, no molecules of species $j$ will be rotated.

For a single box simulation of a non-linear molecule (species 1), a linear molecule (species 2), and a point particle (species 3), you could specify

```
# Prob_Rotation
0.25
30.0 10.0 0.0
```

Molecules of species 1 will be rotated a maximum of 30° around the x, y or z-axis, molecules of species 2 will be rotated by choosing a random unit vector, and the point particles will not be rotated.

**Angle**

**# Prob_Angle**
*Real(1)*
A molecule will be selected at random and its angle will be perturbed based on its Boltzmann weighted distribution. The probability of attempting this move is the only required input. It is specified by *Real(1)*. Note that this move is rarely needed since the fragment libraries should already provide efficient sampling of angles. This move, however, may improve sampling of angles for large molecules in the case where parts of its fragments are rarely regrown by a regrowth move.

```
 # Prob_Angle
0.3
```

This will tell Cassandra to attempt angle moves 30% of the total moves for all molecules containing angles within a given box.

**Dihedral**

**# Prob_Dihedral**
*Real(1)*
*Real(1) ... Real(n)*

The probability of performing a dihedral move is specified by *Real(1)* while *Real(n)* denotes the maximum width of a dihedral angle displacement for each species. The maximum width is given in degrees. Note that this move is rarely needed since the regrowth moves should already provide efficient sampling of dihedrals. This move, however, may improve sampling of dihedrals for large molecules in the case where the parts of its fragments are rarely regrown (albeit a small maximum width is provided).

```
# Prob_Dihedral
0.3
20 0.0
```

This will tell Cassandra to attempt dihedral moves 30% of the total moves for all molecules containing dihedrals within a given box. The maximum dihedral width will be 20° for species 1 and 0.0° for species 2. Since the maximum dihedral width of species 2 is set to 0.0° in both boxes, no dihedral moves will be attempted on species 2. Note that a single max dihedral width is provided, even if species 1 may contain many dihedrals. This is also true for simulations with more than one box. Also note that the same max dihedral width is used for systems containing more than one box.

**Regrowth**

**# Prob_Regrowth**
*Real(1)*
*Real(2,1) ... Real(2,n)*

where $n$ is the number of species. A regrowth move consists of deleting part of the molecule randomly and then regrowing the deleted part via configurational bias algorithm. This can result in relatively substantial conformational changes for the molecule, but the cost of this move is higher than that of a simple translation or rotation. The probability of attempting a regrowth move is specified by *Real(1)* while *Real(2,i)* specifies the relative probability of performing this move on species $i$. The relative probabilities must sum to 1 otherwise Cassandra will quit with an error. This subsection is optional for all ensembles.

For example, if simulating 70 molecules of species 1 and 30 molecules of species 2, you could specify the following:

```
# Prob_Regrowth
0.3
0.7 0.3
```

Thirty percent of the attempted moves will be regrowth moves. Seventy percent of the regrowth moves will be attempted on a molecule of species 1 and the balance of regrowth moves on a molecule of species 2.

*Real(2,i)* should be set to zero for monatomic, linear, or rigid species, including solid frameworks.

**Volume**

**# Prob_Volume**
*Real(1)*
*Real(2)*
[*Real(3)*]

*Real(1)* is the relative probability of attempting a box volume change. Since volume changes are computationally expensive, this probability should normally not exceed 0.05 and values from 0.01-0.03 are typical. *Real(2)* is the maximum volume displacement in Å$^3$ for box 1. *Real(3)* is the maximum volume displacement in Å$^3$ for box 2, and is only required for GEMC-NPT simulations. The attempted change in box volume is selected from a uniform distribution. This subsection is required for NPT, GEMC-NPT and GEMC-NVT simulations. For example, if you are simulating a liquid with a single box in the NPT ensemble, you would specify the following:

```
# Prob_Volume
0.02
300
```

This will tell Cassandra to attempt volume moves 2% of the total moves. The box volume would be changed by random amounts ranging from -300 Å$^3$ to +300 Å$^3$. For a liquid box 20 Å per side, this would result in a maximum box edge length change of about 0.25 Å, which is a reasonable value. Larger volume changes should be used for vapor boxes. If you wish to perform a GEMC-NPT simulation, you might specify the following:

```
# Prob_Volume
0.02
300
5000
```

This will tell Cassandra to attempt volume moves 2% of the total moves. The first box volume (assumed here to be smaller and of higher density, such as would occur if it were the liquid box) would be changed by random amounts ranging from -300 Å$^3$ to +300 Å$^3$. The second box volume would be changed by random amounts ranging from -5000 Å$^3$ to +5000 Å$^3$. As with all move probabilities, you can experiment with making larger or smaller moves. Note that if the # Run_Type is set to `equilibration`, Cassandra will attempt to optimize the magnitude of the volume change to achieve about 50% acceptance rates.

The volume perturbation move is only supported for cubic boxes.

**Insertion and Deletion Moves**

**# Prob_Insertion**
*Real(1)*
*Character(2,1) ... Character(2,n)*

where $n$ is the number of species. *Real(1)* sets the probability of attempting insetion moves. *Character(2,i)* is the insertion method and can be either `cbmc`, `none`, or `restricted`. If `cbmc`, species $i$ will be inserted by assembling its fragments using configurational bias Monte Carlo. If `none`, species $i$ will not be inserted or deleted. If `restricted`, species $i$ will be assembled using CBMC with the first fragment inserted into the region defined by the `restricted_insertion` keyword in the `Box_Info` section. **Note that restricted insertions should only be used if the relevant molecules cannot escape the restricted region during the simulation. If this condition is not met the acceptance criteria for molecule deletion will be incorrect and the ensemble will not be properly sampled.** This subsection is required for GCMC simulations.

If there is more than one insertable species, each is chosen for an insertion attempt with equal probability. For example, if you are performing a GCMC simulation with two species that can be inserted, you might specify the following

```
# Prob_Insertion
0.1
cbmc cbmc
```

This will tell Cassandra to attempt insertions 10% of the total moves and both species will be inserted using CBMC. If only species 1 is to be inserted or deleted, use

```
# Prob_Insertion
0.1
cbmc none
```

**# Prob_Deletion**
*Real(1)*

*Real(1)* is the probability of attempting to delete a molecule during a simulation, and must match the insertion probability to satisfy microscopic reversibility. The molecule to delete is selected by first choosing a species with uniform probability, and then choosing a molecule of that species with uniform probability. If a species has the insertion method `none`, no attempt is made to delete it. This subsection is required for GCMC simulations.

**# Prob_Swap**
*Real(1)*
*Character(2,1) ... Character(2,n)*
[`prob_swap_species` *Real(3,1) ... Real(3,n)*]
[`prob_swap_from_box` *Real(4,1) ... Real(4,i)*]

where $n$ is the number of species and $i$ is the number of boxes. *Real(1)* is the probability of attempting to transfer a molecule from one box to another. Similar to the `# Prob_Insertion` subsection, *Character(2,i)* is the insertion method and can be `cbmc`, `none`, or `restricted`. If `cbmc`, species $i$ will be inserted by assembling its fragments using configurational bias Monte Carlo. If `none`, species $i$ will not be transferred between boxes. If `restricted`, species $i$ will be assembled using CBMC with the first fragment inserted into the region defined by the `restricted_insertion` keyword in the `Box_Info` section. This subsection is required for a GEMC simulation.

For example, while performing a GEMC simulation for three species the first two of which are exchanged while the third is not, specify the following:

```
# Prob_Swap
0.1
cbmc cbmc none
```

This will tell Cassandra to attempt swap moves 10% of the total moves. Attempts will be made to transfer species 1 and 2 between available boxes while molecules of species 3 will remain in the boxes they are present in at the start of the simulation.

By default, a molecule is chosen for the attempted swap with uniform probability (amongst swappable molecules). As a result, if one species has a much higher mole fraction in the system (e.g. if calculating methane solubility in liquid water), then most attempted swaps will be of the more abundant species. This behavior can be changed by using the optional keywords `prob_swap_species` and `prob_swap_from_box`.

The keyword `prob_swap_species` must be given with $n$ options: *Real(3,j)* is the probability of selecting species $j$. The keyword `prob_swap_from_box` must be given with $i$ options: *Real(4,j)* is the probability of selecting a molecule from box $j$. For example, to select a molecule of species 1 for 90% of attempted swaps and to select box 2 as the donor box for 75% of attempted swaps, use:

```
# Prob_Swap
0.1
cbmc cbmc none
prob_swap_species 0.9 0.1 0.0
prob_swap_from_box 0.25 0.75
```

The probability of selecting a species with insertion method `none` must be 0.

**Flip Move**

**# Prob_Ring**
*Real(1) Real(2)*

This subsection is used when flip moves are to be attempted to sample bond angles and dihedral angles in a ring fragment. For more details on this move see Ref. [2]. Note that this subsection is used only in input files that generate configuration libraries of ring moieties. The input file of the actual simulation would

involve the `# Prob_Regrowth` keyword. The relative probability of attempting a flip move is specified by *Real(1)* while the maximum angular displacement in degrees for the move is given by *Real(2)*. For example, if the flip is to be attempted 30% of the time and the maximum angular displacement for the move is 20° specify the following:

```
# Prob_Ring
0.30 20.0
```

### 4.1.16 Start Type

**# Start_Type**
*Character(1)*
[*Character(2)* ]
[`insertion` *Character(3,1) ... Character(3,n)*]

This section specifies whether Cassandra generates an initial configuration or uses a previously generated configuration to start a simulation. *Character(1)* [*Character(2)*] can be one of four keywords: `make_config`, `read_config`, `add_to_config`, or `checkpoint`.

`make_config` and `add_to_config` are options to construct an initial configuration by inserting a specified number of molecules of each species. Each molecule is inserted using configuration bias Monte Carlo, using `kappa_ins` trial locations for the first fragment and `kappa_dih` trial rotations for each additional fragment. Trial locations and rotations that place two atoms closer than `Rcutoff_Low` Å have zero weight. Otherwise the weight of the trial location is computed as discussed in section 7.3.1 and one trial is selected proportionate to its weight. If all trial locations have zero weight, the insertion is rejected and re-attempted. This process does not utilize a chemical potential nor does it compute the change in energy from inserting the fully assembled molecule. As a result, these routines will allow the user to insert more molecules than are thermodynamically reasonable at finite temperature or finite chemical potentials. This can be particularly problematic for deleting molecules in GCMC and GEMC simulations. When computing the reverse probability for inserting that molecule back into the location it's being deleted from, if the reverse move is more than 708 $k_BT$ uphill, then the acceptance probability blows up.

- **make_config** will generate an initial configuration using a configurational biased scheme. The number of molecules of each species is specified as follows:

  **# Start_Type**
  `make_config` *Integer(1) ... Integer(n)*

  where $n$ is the number of species and *Integer(i)* is the number of molecules of species $i$ to insert into the box. This keyword can be repeated for each box. For example, to generate an initial configu-

ration with 100 molecules of species 1 and 75 molecules of species 2

```
# Start_Type
make_config 100 75
```

If the simulation also has a second box with 25 molecules of species 2 only

```
# Start_Type
make_config 100 75
make_config   0 25
```

Please note the caution regarding `make_config` at the end of this section.

- **read_config** will use the coordinates from an .xyz file. For example, a configuration generated at one temperature may be used to initiate a simulation at another temperature. After `read_config` the number of molecules of each species must be given, followed by the .xyz filename:

  **# Start_Type**
  `read_config` *Integer(1) ... Integer(n) Character(1)*

  where $n$ is the number of species, *Integer(i)* is the number of molecules of species $i$ to read from file *Character(1)*. This keyword can be repeated for each box. For example, to start a simulation using a configuration of 50 molecules each of species 1 and 2:

  ```
  # Start_Type
  read_config 50 50 liquid.xyz
  ```

  If the simulation also has a second box with 10 molecules of species 1 and 90 molecules of species 2:

  ```
  # Start_Type
  read_config 50 50 liquid.xyz
  read_config 10 90 vapor.xyz
  ```

  The .xyz files must have the following format:

  ```
  <number of atoms>
  comment line
  <element> <x> <y> <z>
  ...
  ```

- **add_to_config** will read the coordinates from an .xyz file, but then insert additional molecules. After `add_to_config` specify the number of molecules of each species to be read, followed by the .xyz filename, followed by the number of molecules of each species to be added:

**# Start_Type**
`add_to_config` *Integer(1) ... Integer(n) Character(1) Integer(n+1) ... Integer(2n)*

where $n$ is the number of species, *Integer(1)* through *Integer(n)* are the number of molecules of each species to read from file *Character(1)*, and *Integer(n+1)* through *Integer(2n)* are the number of molecules of each species to add to the configuration. This keyword can be repeated for each box. For example, to start a simulation by reading in a zeolite (speces 1) configuration and adding 30 molecules of methane (species 2):

```
# Start_Type
add_to_config 1 0 MFI.xyz 0 30
```

where the file `MFI.xyz` contains the coordinates of a unit cell of MFI silicalite.
Please note the caution regarding `add_to_config` at the end of this section.

- **checkpoint** this keyword is used to restart a simulation from a checkpoint file. During the course of a simulation, Cassandra periodically generates a checkpoint file (*.chk) containing information about the total number of translation, rotation and volume moves along with the random number seeds and the coordinates of each molecule and its box number at the time the file is written. Cassandra provides the capability of restarting from this state point in the event that a simulation crashes or running a production simulation from an equilibrated configuration. For this purpose, in addition to the `checkpoint` keyword, additional information in the form of the name of the checkpoint file *Character(1)* is required in the following format:

  **# Start_Type**
  `checkpoint` *Character(1)*

  For example, to continue simulations from a checkpoint file 'methane_vle_T148.chk', you might specify:

  ```
  # Start_Type
  checkpoint methane_vle_T148.chk
  ```

  Note that when a `checkpoint` file is used to restart a simulation, the seeds for random number generation supplied by the user will be overwritten by those present in the checkpoint file. By contrast, if # Start_Type is set to `read_config`, then the seeds specified in the input file are used.

Unless starting from a checkpoint file, input files for a multi-box simulation must have one line for each box in the `Start_Type` section. Each line can start with a different keyword. For example, a GEMC simulation of a water(1)-methane(2) mixture can begin from an equilibrated water box and a new vapor box:

```
# Start_Type
read_config 100   0 water.xyz
make_config  50  50
```

The keyword `insertion` is optional and is only meaningful if used in conjunction with the keyword `restricted_insertion` in the `Box_Info` section and either the `make_config` or `add_to_config` keywords in this section. *Character(3,i)* is the insertion method for species $i$ and can be one of the following options: `cbmc`, `none`, or `restricted`. If `cbmc`, species $i$ will be assembled using configurational bias Monte Carlo. If `none`, species $i$ will not be inserted. If `restricted`, species $i$ will be assembled using CBMC with the first fragment inserted into the region defined by the `restricted_insertion` keyword in the `Box_Info` section.

### 4.1.17 Run Type

**# Run_Type**
*Character(1) Integer(1)* [*Integer(2)*]

This section is used to specify whether a given simulation is an equilibration or a production run. For an equilibration run, the maximum translational, rotational, torsional and volume widths (for an NPT or a GEMC simulation) are adjusted to achieve 50% acceptance rates. During a production run, the maximum displacement width for different moves are held constant.

Depending on the type of the simulation, *Character(1)* can be set to either `equilibration` or `production`. For an `equilibration` run, *Integer(1)* denotes the number of MC steps performed for a given thermal move before the corresponding maximum displacement width is updated. *Integer(2)* is the number of MC volume moves after which the volume displacement width is updated. This number is optional if no volume moves are performed during a simulation (for example in an NVT or a GCMC simulation). When the run type is set to `production`, the MC moves refer to the frequency at which the acceptance ratios for various moves will be computed and output to the log file. These acceptance rates should be checked to make sure proper sampling is achieved.

For an NPT equilibration run in which the widths of the thermal move are to be updated after 1000 MC moves and maximum volume displacements after 100 volume moves, specify the following:

```
# Run_Type
equilibration 100 10
```

For an NVT production run in which the acceptance ratios of various thermal moves are printed to the log file after every 250 MC steps of a given thermal move, use the following:

```
# Run_Type
production 250
```

### 4.1.18 Simulation Length

**# Simulation_Length_Info**
`units` *Character(1)*
`prop_freq` *Integer(2)*
`coord_freq` *Integer(3)*
`run` *Integer(4)*
[`steps_per_sweep` *Integer(5)*]
[`block_averages` *Integer(6)*]

This section specifies the frequency at which thermodynamic properties and coordinates are output to a file. The `units` keyword determines the method by which the simulation is terminated and data is output. *Character(1)* can be `minutes`, `steps`, or `sweeps`. Thermodynamic quantities are output every *Integer(2)* units, coordinates are written to the disk every *Integer(3)* units and the simulation will stop after *Integer(4)* units.

If *Character(1)* is `minutes`, then the simulation runs for a specified time. For example, to run a simulation for 60 minutes with thermodynamic properties written every minute and coordinates output every 10 minutes, use:

```
# Simulation_Length_Info
units minutes
prop_freq 1
coord_freq 10
run 60
```

If *Character(1)* is `steps`, the simulation runs for a specified number of MC steps. An MC step is defined as a single MC move, regardless of type and independent of system size. To run a simulation of 50,000 steps such that thermodynamic quantities are printed every 100 MC steps and coordinates are output every 10,000 steps, use:

```
# Simulation_Length_Info
units steps
prop_freq 100
coord_freq 10000
run 50000
```

If *Character(1)* is `sweeps`, the simulation runs for a specified number of MC sweeps. The number of MC steps per sweep can be defined using the optional keyword

steps_per_sweep *Integer(5)*

The default `steps_per_sweep` value is the sum of the weights of each move type. A sweep is typically defined as the number of MC moves needed for every move to be attempted for every molecule. For example, in a water box of 100 molecules in the NPT ensemble, a sweep would be 201 moves—100 translations,

100 rotations and 1 volume change. To run a simulation of 1,000 sweeps with thermodynamic quantities are printed every 100 sweeps and coordinates are output every 100 sweeps, use the following:

```
# Simulation_Length_Info
units sweeps
prop_freq 100
coord_freq 100
run 1000
steps_per_sweep 201
```

The optional keyword `block_avg_freq` switches the thermodynamic output from instantaneous values to block average values, where *Integer(6)* is the number of units per block. The number of blocks is given by *Integer(4)/Integer(6)* and the number of data points per block is *Integer(6)/Integer(2)*. For example, during a run of 1,000,000 steps, with properties computed every 100 steps and averaged every 100,000 steps, specify

```
# Simulation_Length_Info
units steps
run 1000000
block_avg_freq 100000
prop_freq 100
coord_freq 100
```

This simulation will output 10 averages, and each average will be computed from 1000 data points.

### 4.1.19 Property Output

**# Property_Info** *Integer(i)*
*Character(i,j) *One line for each property j*

This section provides information on the properties that are output. More than one section is allowed for multiple boxes. In this case, each section is separated by a blank line. *Integer(i)* is the identity of the box for which the properties are desired. *Character(i,j)* is the property that is to be output. Each property is specified on a separate line.

All energies are in kJ/mol and are extensive, i.e. if the numbers of molecules in a simulation are doubled, the magnitude of the energy will also double. The kJ unit of energy is the right order of magnitude for molar quantities, $\mathcal{O}(10^{23})$ molecules. Cassandra is designed for simulations of $\mathcal{O}(100-1000)$ molecules, which will have much smaller internal energies, $\mathcal{O}(10^{-21})$ kJ. Rather than report energies in zeptojoules or eV, we have opted to multiply the energies by Avogadro's number. Or, equivalently, you can interpret the output energies as the energy for a mole of simulation boxes. To get extensive energies in kJ, divide the output energies by Avogadro's number. To get intensive energies in kJ/mol, divide the output energies by the number of molecules (only strictly valid for single species simulations). The following components of the energy can be output:

`energy_total`: total energy of the system, the sum of `energy_intra` and `energy_inter`

`energy_intra`: intramolecular energy, the sum of the following terms

- `energy_bond`: bond energy
- `energy_angle`: angle energy
- `energy_dihedral`: dihedral energy
- `energy_improper`: improper energy
- `energy_intravdw`: intramolecular van der Waals energy
- `energy_intraq`: intramolecular electrostatic energy. In the case of Ewald and DSF methods, this is the real-space intramolecular electrostatic energy.

`energy_inter`: intermolecular energy, the sum of the following terms

- `energy_intervdw`: intermolecular van der Waals energy
- `energy_lrc`: long range tail correction for the truncated van der Waals energy
- `energy_interq`: intermolecular electrostatic energy. In the case of Ewald and DSF methods, this is the real-space intermolecular electrostatic energy.
- `energy_recip`: electrostatic reciprocal energy, for Ewald and DSF methods
- `energy_self`: electrostatic self energy, for Ewald method

Additional supported keywords are:

`enthalpy`: enthalpy of the system, in kJ/mol (extensive). The enthalpy is computed using the pressure setpoint for isobaric simulations and the computed pressure for all other ensembles.
`pressure`: pressure of the system, in bar
`pressure_xx`: the xx-component of the pressure tensor, in bar
`pressure_yy`: the yy-component of the pressure tensor, in bar
`pressure_zz`: the zz-component of the pressure tensor, in bar
`volume`: volume of the system, in $Å^3$
`nmols`: number of molecules of each species
`density`: number density of each species, in $\#/Å^3$
`mass_density`: density of the system, in $kg/m^3$

For example, if you would like total energy, volume and pressure of a one box system to be written, you may specify the following:

```
# Property_Info 1
energy_total
```

```
volume
pressure
```

For a GEMC-NVT simulation, total energy and density of all the species in box 1 and total energy, density of all the species in box 2 along with the pressure may be output using the following format:

```
# Property_Info 1
energy_total
density

# Property_Info 2
energy_total
density
pressure
```

### 4.1.20 Fragment Files

**# Fragment_Files**
*Character(i) Integer(i)* *One line for each fragment $i$

In this section, information about the fragment library is specified. *Character(i)* gives the location of the fragment library of fragment $i$; *Integer(i)* is the corresponding integer id specifying the type of the fragment. This section is automatically generated by library_setup.py. However, if there is a need to change this section, follow the example given below.

For a simulation involving two species of which the first one contains three distinct fragments and species 2 has two identical fragments, this section might look like:

```
# Fragment_Files
frag_1_1.dat 1
frag_2_1.dat 2
frag_3_1.dat 3
frag_1_2.dat 4
frag_1_2.dat 4
```

This will tell Cassandra to use the files frag_1_1.dat, frag_2_1.dat and frag_3_1.dat for the three fragments of species 1. Since species 2 has two identical fragment, Cassandra will use the same fragment library frag_1_2.dat for these fragments.

### 4.1.21 Verbosity in log file

**# Verbose_Logfile**

*Logical*

This optional section is used to control the level of detail about the simulation setup that is output to the log file. Controlling this can be useful for development purposes. If this section is missing, *Logical* is set to `false` by default. Supported options for *Logical* are `true` or `false`.

### 4.1.22 File Info

**# File_Info**
*Character*

This section is used only while generating a fragment library. Cassandra will use the filename specified in *Character* to store different conformations of the fragment being simulated. Once again, this section is automatically handled by library_setup.py. However, if the user wishes to modify this part, use the following template:

```
# File_Info
frag.dat
```

This will tell Cassandra to store the fragment library in the file named `frag.dat`.

### 4.1.23 CBMC parameters

**# CBMC_Info**
`kappa_ins` *Integer(1)*
`kappa_dih` *Integer(2)*
`rcut_cbmc` *Real(3,1)* [*Real(3,2)*]

Cassandra utilizes a configurational bias methodology based on sampling a library of fragment conformations [2]. This section sets a number of parameters required for biased insertion/deletion (refer to the sections # Prob_Insertion, # Prob_Deletion and # Prob_Swap) and configurational regrowth (# Prob_Regrowth section) of molecules. This section is only required if molecules are regrown, inserted and/or deleted.

Keyword `kappa_ins` is required if the section `# Start_Type` is given with keyword `make_config` or `add_to_config`, or if the section `# Sim_Type` is `gcmc`, `gemc` or `gemc_npt`. For a biased insertion, a fragment is chosen to insert first in proportion to the number of atoms in fragment. For example, to insert a united-atom molecule of ethylbenzene, the ring fragment has 7 pseudoatoms while the other has 3. The ring fragment will be inserted first with a probability of 0.7. By contrast, to insert a united-atom molecule of dodecane, all ten fragments have 3 pseudoatoms and so one is chosen with uniform probability. After choosing a Boltzmann-distributed conformation and an orientation with uniform probability, *Integer(1)* trial positions are generated for the

center-of-mass of the fragment. One of the trial positions is then selected randomly based on the Boltzmann weight of the energy of the trial position.

Keyword `kappa_dih` is required if any species composed of multiple fragments is inserted/deleted or regrown. Additional fragments are added to the growing molecule using *Integer(2)* trial dihedral angles that connect the new fragment to the existing part of molecule.

Keyword `rcut_cbmc` is required if section `# CBMC_Info` is required. For all the trials, energy of the partially grown molecule with itself and surrounding molecules is to be calculated. For this purpose, a short cutoff is used. *Real(3,i)* specifies the cutoff distance in Å for box $i$. A short cutoff is fast, but might miss some overlaps. You can experiment with this value to optimize it for your system.

For a GEMC simulation in which 12 candidate positions are generated for biased insertion/deletion, 10 trials for biased dihedral angle selection and the cutoff for biasing energy calculation is set to 5.0 Å in box 1 and 6.5 Å in box 2, this section would look like:

```
# CBMC_Info
kappa_ins 12
kappa_dih 10
rcut_cbmc 5.0 6.5
```

## 4.2 Molecular Connectivity File

A Molecular Connectivity File (MCF) defines the information related to bonds, angles, dihedrals, impropers fragments and non bonded interactions for a given species. One MCF is required for each species present in the system. The information contained in this file involves the force field parameters, atoms participating in each of the interactions and the functional form used in each potential contribution. The keywords are preceeded by a '#' and comments follow a '!'. Similarly to the input file, the order of the keywords is not important. A complete list of the keywords is provided below.

**Note that parameters for all of the following keywords must be separated by spaces only. Do not use the tab character.**

**Note that MCFs are generated by the script mcfgen.py automatically. The following description is provided for the users who wish to modify the MCF or manually write the MCF.**

### 4.2.1 Atom Info

**# Atom_Info**
*Integer(1)*
*Integer(2) Character(3)*20 Character(4)*6 Real(5) Real(6) Character(7)*20 Optional_Parms Character(fin)*

This keyword specifies the information for non-bonded interactions. It is a required keyword in the MCF. If not specified, the code will abort. The inputs are specified below:

- *Integer(1)*: Total number of atoms in the species.
- *Integer(2)*: Atom index.
- *Character(3)*20*: Atom type up to 20 characters. This string of characters should be unique for each interaction site in the system, i.e. do not use the same atom type for two atoms in the same (or different) species unless the (pseudo)atoms have the same atom types.
- *Character(4)*6*: Atom element name up to 6 characters.
- *Real(5)*: Mass of the atom in amu. Note that for united atom models, this would be the mass of the entire pseudoatom.
- *Real(6)*: Charge on the atom.
- *Character(7)*: The functional form for van der Waals (vdW) interactions. Options are "LJ" for Lennard-Jones, "Mie" for the Mie potential, or "NONE" if the atom type does not have vdW interactions. "LJ" and "Mie" cannot be used in the same simulation. This must match what is given for `# VDW_Style` (subsection 4.1.4) in the input file.
- *Character(fin)*: The final entry on the line is 'ring' only if the atom is part of a ring. For atoms that are not part of rings, leave this field blank. Note that the ring fragments contain all ring atoms (e.g., in cyclohexane, all six carbons), and any atoms bonded directly to the ring (e.g., in cyclohexane, all the hydrogens). However, the 'ring' entry is ONLY appended for those atoms that belong to the ring (e.g., for cyclohexane, only the carbon atoms would have 'ring' appended).

Additional parameters are required for LJ and Mie potentials. For LJ,

- *Real(8)*: The energy parameter in K.
- *Real(9)*: Collision diameter ($\sigma$) in Å.

For Mie,

- *Real(8)*: The energy parameter in K.

- *Real(9)*: Collision diameter ($\sigma$) in Å.
- *Real(10)*: The repulsive exponent.
- *Real(11)*: The dispersive exponent.

**Note that for species with a single fragment, the branch point atom is listed as the first atom.**

For example, for a united atom pentane model:

```
# Atom_Info
5
1 CH3_s1 C 15.0107 0.0 LJ 98.0 3.75
2 CH2_s1 C 14.0107 0.0 LJ 46.0 3.95
3 CH2_s1 C 14.0107 0.0 LJ 46.0 3.95
4 CH2_s1 C 14.0107 0.0 LJ 46.0 3.95
5 CH3_s1 C 15.0107 0.0 LJ 98.0 3.75
```

The number below the keyword `# Atom_Info` specifies a species with 5 interaction sites, consistent with a united atom pentane model. The first column specifies the atom ID of each of the pseudo atoms. The second and third columns provide the atom type and atom name, respectively. The fourth column represents the atomic mass of each pseudoatom. Note that the mass of CH3_s1 is 15.0107 for this united atom model, as it involves a carbon and three hydrogen atoms. The same applies for all other interaction sites. The fifth column contains the partial charges placed on each of these pseudoatoms. The sixth, seventh and eighth columns contain the repulsion-dispersion functional form, the energy parameter and the collision diameter respectively. In this case, the usual Lennard-Jones functional form is used. Note that none of these atoms used the flag 'ring', as no rings are present in this molecule.

For a molecule containing rings, for example cyclohexane:

```
# Atom_Info
6
1 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
2 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
3 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
4 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
5 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
6 CH_s1 C 13.0107 0.0 LJ 52.5 3.91 ring
```

Note the flag 'ring' was appended as the last column for this cyclic molecule.

For the SPC/E water model:

```
# Atom_Info
3
1 O1_s1 O 16.00 -0.8476 LJ 78.20 3.1656
```

```
2 H2_s1 H 1.000 0.4238 NONE
3 H3_s1 H 1.000 0.4238 NONE
```

This is a molecule with a single fragment, so the branch point atom is the first atom in the list.

For a single-site model of CO2 using the Mie potential:

```
# Atom_Info
1
1 CO2 C 44.00 0.0 Mie 361.69 3.741 23.0 6.66
```

where the last two parameters are the repulsive and dispersive exponents, respectively.

### 4.2.2 Bond Info

**# Bond_Info**
*Integer(1)*
*Integer(i,2) Integer(i,3) Integer(i,4) Character(i,5) Real(i,6) Optional_Parms Real(i,7)*

This section provides information on the number of bonds in a molecule and atoms involved in each bond along with its type. It is a required keyword in the MCF. If not specified, the code will abort. The inputs are specified below:

- *Integer(1)*: Total number of bonds in the species. From the next line onwards, the bonds are listed sequentially and information for each bond is included on a separate line.
- *Integer(i,2)*: Index of the $i^{th}$ bond.
- *Integer(i,3) Integer(i,4)*: IDs of the atoms participating in the bond.
- *Character(i,5)*: Type of the bond. At present only 'fixed' is permitted.
- *Real(i,6)*: Specifies the bond length for a particular bond in Å.
- *Real(i,7), Optional*: Specify the allowable deviation from the bond length. Default value is 0.01 Å.

**Note that at present, Cassandra simulations can be carried out only for fixed bond length systems.**

For example, for the water model SPC/E, the `# Bond_Info` section is the following:

```
# Bond_Info
2
1 1 2 fixed 1.0
```

```
2 1 3 fixed 1.0
```

In the above example, two bonds are specified whose fixed length is set to 1.0 Å.

### 4.2.3 Angle Info

**# Angle_Info**
*Integer(1)*
*Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) Character(i,6) Real(i,7) Real(i,8)*

The section lists the information on the angles in the species. It is a required keyword in the MCF. If not specified, the code will abort.

- *Integer(1)*: Number of angles in the species.
- *Integer(i,2)*: Index of the $i^{th}$ angle.
- *Integer(i,3) Integer(i,4) Integer(i,5)*: IDs of the atoms participating in the $i^{th}$ angle. Note that *Integer(i,4)* is the ID of the central atom.
- *Character(i,6)*: Type of the angle. Currently, Cassandra supports 'fixed' and 'harmonic' (Eq. section 2.1) angles. For the 'fixed' option, *Real(i,7)* is the value of the angle and *Real(i,8)* is ignored by the code if specified. In the case of 'harmonic' potential type, *Real(i,7)* specifies the harmonic force constant ($K/rad^2$) while *Real(i,8)* is the nominal bond angle (in degrees).

For example, for a united atom pentane molecule with flexible angles, this section is the following:

```
# Angle_Info
3
1 1 2 3 harmonic 31250.0 114.0
2 2 3 4 harmonic 31250.0 114.0
3 3 4 5 harmonic 31250.0 114.0
```

In the above example, the three angles between the pseudoatoms found in the pentane model are specified. The three angles have an harmonic potential, whose force constant is equal and is set to 31250.0 K/rad$^2$. Finally, the equilibrium angle for these angles is 114.0°.

An example for SPC/E water model with fixed angles is provided below:

```
# Angle_Info
1
1 2 1 3 fixed 109.47
```

This model has only one angle that is set to 109.47°. Note that this angle is fixed, so there is no force constant.

### 4.2.4 Dihedral Info

**# Dihderal_Info**
*Integer(1)*
*Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) Integer(i,6) Character(i,7) Real(i,8) Real(i,9) Real(i,10) Real(i,11)*

This section of the MCF lists the number of dihedral angles and associated information for a given species. It is a required keyword in the MCF. If not specified, the code will abort.

- *Integer(1)*: Lists the number of dihedral angles.
- *Integer(i,2)*: Index of the $i^{th}$ dihedral angle.
- *Integer(i,3): Integer(i,6)* - IDs of the atoms in the $i^{th}$ dihedral angle.
- *Character(i,7)* : Dihedral potential type. Acceptable options are 'OPLS', 'CHARMM', 'harmonic' and 'none'. If 'OPLS' dihedral potential type is selected, then the real numbers *Real(i,8) - Real(i,11)* are the coefficients in the Fourier series (see Eq 2.2). The units are in kJ/mol. For the 'CHARMM' dihedral potential type, three additional parameters are specified: $a_0, a_1$ and $\delta$ (section 2.3). If 'harmonic' dihedral potential type is used, then two additional parameters, $K_{phi}$ and $\phi_0$ (section Eq 2.4), are specified. For the 'none' dihedral potential type, no additional parameters are necessary.

For example, for a united atom pentane molecule using an OPLS dihedral potential type, the dihedrals are specified as follows:

```
# Dihedral_Info
2
1 1 2 3 4 OPLS 0.0 2.95188 -0.5670 6.5794
2 2 3 4 5 OPLS 0.0 2.95188 -0.5670 6.5794
```

In this model two dihedral angles are specified by atoms 1,2,3,4 and 2,3,4,5. This model uses an OPLS functional form and thus four parameters are provided after the OPLS flag.

### 4.2.5 Intramolecular Scaling

**# Intra_Scaling**
*Real(i,1) Real(i,2) Real(i,3) Real(i,4)*
*Real(i,5) Real(i,6) Real(i,7) Real(i,8)*

This section sets the intramolecular scaling for 1-2, 1-3, 1-4 and 1-N interactions within a given species. 1-2 means interactions between atom 1 and another atom 2 directly bonded to it, 1-3 means interactions between atom 1 and other atoms 3 separated from atom 1 by exactly two bonds, etc. The first line corresponds to the VDW scaling: *Real(i,1) Real(i,2) Real(i,3) Real(i,4)* apply to 1-2, 1-3, 1-4 and 1-N interactions, where N corresponds to all atoms separated from atom 1 by more than three bonds. The second line corresponds to the Coulomb scaling: *Real(i,5) Real(i,6) Real(i,7) Real(i,8)* apply to 1-2, 1-3, 1-4 and 1-N interactions. Note that intramolecular scaling applies to all the boxes in the simulation.

For example,

```
# Intra_Scaling
0.0 0.0 0.5 1.0
0.0 0.0 0.5 1.0
```

would turn off 1-2 and 1-3 interactions, would scale the VDW and Coulombic interactions for 1-4 atoms by 50% and would use full interactions for all other atom pairs in the species.

If the # Intra_Scaling section is missing from the MCF, it will be looked for in the input file.

### 4.2.6 Fragment Info

**# Fragment_Info**
*Integer(1)*
*Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) ... Integer(i,2+Integer(i,3))*

This section defines the total number of fragments in a given species. It is an optional keyword. However, if the species is composed of fragments, then this section must be specified. The inputs are specified below:

- *Integer(1)*: Total number of fragments.
- *Integer(i,2)*: Index of the $i^{th}$ fragment.
- *Integer(i,3)*: Number of atoms in the $i^{th}$ fragment.
- *Integer(i,4) ... Integer(i,2+integer(i,3))*: List of the atom IDs in the fragment. The first atom ID is that for the branch point atom. **Atom ordering for the remaining atoms must match the order of atoms in the fragment library files.**

For example, for a pentane united atom model:

```
# Fragment_Info
3
1 3 2 1 3
2 3 3 2 4
3 3 4 3 5
```

This specifies three fragments. Each of these fragments has three atoms. The first atom specified for each of the fragments is the branch point atom.

### 4.2.7 Fragment Connectivity

**# Fragment_Connectivity**
*Integer(1)*
*Integer(i,2) Integer(i,3) Integer(i,4)*

The section lists the fragment connectivity - which fragment is bonded to which other fragment. It is a required keyword if **Fragment_Info** is specified.

- *Integer(1)*: total number of fragment connections.
- *Integer(i,2)*: index of the $i^{th}$ fragment connectivity.
- *Integer(i,3) Integer(i,4)*: fragment IDs participating in the connectivty.

For example, for a pentane united atom model:

```
# Fragment_Connectivity
2
1 1 2
2 2 3
```

In this example, there are three fragments, therefore, two fragment connectivities must be specified. Note that fragment 1 is connected to fragment 2 and fragment 2 is connected to fragment 3.

# Chapter 5

# Utilities

All Python utility scripts require the `future` package and `numpy` package, as listed in `requirements-py.txt`. Both packages are available via `pip` or `conda`.

Please note that `python2` support is deprecated and may be removed in a future release.

## 5.1 Generate a Molecular Connectivity File

The script `mcfgen.py` is a tool that aims to ease the setup of molecular connectivity files from scratch (see section 4.2 to learn more about MCFs), as the generation of these files by hand can be error prone. In this section, a pentane MCF will be generated to demonstrate the use of this tool. The Transferable Potentials for Phase Equilibria (TraPPE) force field will be used to represent the pentane molecular interactions. This force field involves a pairwise-additive 12-6 Lennard-Jones potential to represent the dispersion-repulsion interactions. Additionally, bond angles and dihedral angles are represented through harmonic and OPLS functional forms, respectively. Bond lengths are kept constant. The force field mathematical expression becomes

$$U = \sum_{angles} K_\theta(\theta - \theta_0)^2 + \sum_{dihedrals} \frac{1}{2}K_1[1 + cos(\phi)] + \frac{1}{2}K_2[1 - cos(2\phi)] + \frac{1}{2}K_3[1 + cos(3\phi)] + \frac{1}{2}K_4[1 - cos(4\phi)] + \sum_i \sum_{i>j} 4\epsilon_{ij} \left[ \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{12} - \left( \frac{\sigma_{ij}}{r_{ij}} \right)^{6} \right]$$

First, generate (or obtain) a PDB file or a CML file. To generate a PDB or CML file, software such as

Gaussview or Avogadro can be used. Alternatively, PDB files can be downloaded from the internet (e.g. www.rcsb.org). In this example, a pentane PDB file using the program Gaussview v5.08 was be generated, as shown below.

```
REMARK   1 File created by GaussView 5.0.8
HETATM    1  C           0      -4.277   0.355   0.000                       C
HETATM    2  C           0      -3.763   1.081   1.257                       C
HETATM    3  C           0      -2.223   1.078   1.259                       C
HETATM    4  C           0      -1.710   1.805   2.516                       C
HETATM    5  C           0      -0.170   1.802   2.517                       C
END
CONECT    1    2
CONECT    2    1    3
CONECT    3    2    4
CONECT    4    3    5
CONECT    5    4
```

Append a column containing the atom types.

```
REMARK   1 File created by GaussView 5.0.8
HETATM    1  C           0      -4.277   0.355   0.000                       C CH3
HETATM    2  C           0      -3.763   1.081   1.257                       C CH2
HETATM    3  C           0      -2.223   1.078   1.259                       C CH2
HETATM    4  C           0      -1.710   1.805   2.516                       C CH2
HETATM    5  C           0      -0.170   1.802   2.517                       C CH3
END
CONECT    1    2
CONECT    2    1    3
CONECT    3    2    4
CONECT    4    3    5
CONECT    5    4
```

Avogadro v1.1.1 can also be used to generate CML files. Below is an example of a CML file generated using Avogadro

```
<molecule>
 <atomArray>
  <atom id="a1" elementType="C" x3="-0.367789" y3="-0.161907" z3="0.206019"/>
  <atom id="a2" elementType="C" x3="-1.354811" y3="-1.178938" z3="-0.372241"/>
  <atom id="a3" elementType="C" x3="-2.735586" y3="-0.597632" z3="-0.678858"/>
  <atom id="a4" elementType="C" x3="-3.435276" y3="0.007943" z3="0.526735"/>
  <atom id="a5" elementType="C" x3="1.027694" y3="-0.340782" z3="-0.372648"/>
 </atomArray>
 <bondArray>
  <bond atomRefs2="a1 a2" order="1"/>
  <bond atomRefs2="a3 a4" order="1"/>
  <bond atomRefs2="a3 a2" order="1"/>
  <bond atomRefs2="a5 a1" order="1"/>
 </bondArray>
</molecule>
```

Modify the pentane united atom CML file. Note that the atom type is appended as a last column between quotation marks.

```
<molecule>
 <atomArray>
  <atom id="a1" elementType="C" x3="-0.367789" y3="-0.161907" z3="0.206019"/> "CH2"
  <atom id="a2" elementType="C" x3="-1.354811" y3="-1.178938" z3="-0.372241"/> "CH2"
  <atom id="a3" elementType="C" x3="-2.735586" y3="-0.597632" z3="-0.678858"/> "CH2"
  <atom id="a4" elementType="C" x3="-3.435276" y3="0.007943" z3="0.526735"/> "CH3"
  <atom id="a5" elementType="C" x3="1.027694" y3="-0.340782" z3="-0.372648"/> "CH3"
 </atomArray>
 <bondArray>
  <bond atomRefs2="a1 a2" order="1"/>
  <bond atomRefs2="a3 a4" order="1"/>
  <bond atomRefs2="a3 a2" order="1"/>
  <bond atomRefs2="a5 a1" order="1"/>
 </bondArray>
</molecule>
```

In the terminal, run the following command:

```
>python mcfgen.py pentane.pdb --ffTemplate
```

This command will create an .ff file. The first three sections of the FF file are displayed next. Do not modify these.

```
atomtypes
2

begin atom-atomtype
1 CH3
2 CH2
3 CH2
4 CH2
5 CH3
end atom-atomtype

dihedraltype OPLS
```

The force field parameters for non-bonded (not shown), bonded, angle, dihedral (not shown) and coulombic interactions (not shown) must be entered next to the corresponding keyword. For example, the angle type CH3 CH2 CH2 has an angle of 114.0. This value must be placed next to the “Angle” keyword.

```
bonded
CH2 CH2
Length 1.54
Constant fixed

angles
CH3 CH2 CH2
Angle 114.0
Constant 31250.0
```

For more examples of filled ff files, please refer to the examples contained in the /Scripts/MCF_Generation/ directory. Using the filled .ff file, run:

```
> python mcfgen.py pentane.pdb
```

Check the file newly created pentane.mcf for any possible errors. This example can be found in the directory /Scripts/MCF_Generation/PDB/

Note that if an MCF for a rigid solid is being created, this last step must include the –solid flag, as

```
> python mcfgen.py zeolite.pdb --solid
```

## 5.2 Generate Library of Fragment Configurations

The goal of the script `library_setup.py` is to automate the generation of fragment libraries. As a starting point, the script requires the simulation input file, and the MCF and PDB files for each of the species. To run this script, type

```
> library_setup.py $PATH$/cassandra.exe input_file.inp pdbfilespecies1.pdb pdfilespecies2.pdb
...
```

This script will create the necessary files to create the fragment libraries. It will also run Cassandra to generate these libraries, whose location will be at

```
/species?/frag?/frag?.inp
```

where '?' refers to the species number, for example, species 1, species 2 etc.

Note that the script overwrites the section of the input file where needed (i.e. # Frag_Info) with the aforementioned directory locations.

# Chapter 6

# Simulating Rigid Solids and Surfaces

Simulations involving a rigid solid or surface can be performed in constant volume ensembles (i.e., NVT and GCMC). For example, an adsorption isotherm can be computed with a GCMC simulation in which fluid molecules are inserted into a crystalline solid. In addition to the files described in Chapter 4, the following files are required to run a simulation with a rigid solid or surface:

- a molecular connectivity file with force field parameters for each atom in the solid (*.mcf)
- a fragment library file listing the coordinates of each atom in the solid (*.dat)
- a configuration file with the initial coordinates of the all atoms in the system (*.xyz)

The MCF and fragment library file can be created using the scripts discussed in Sections 5.1 and 5.2. Each of these scripts requires a starting PDB configuration file. The input file also requires specific parameters as given in the following section. Sample input scripts for GCMC simulations of various fluids in silicalite are included in the Examples/GCMC directory of the Cassandra distribution.

## 6.1 Input file

The input file should be completed as described in Section 4.1, but with the following parameters:

- Under the keyword `# Prob_Translation`, the translation width for the solid is 0.
- Under the keyword `# Prob_Rotation`, the rotation width for the solid is 0.
- Under the keyword `# Prob_Regrowth`, the regrowth probability for the solid is 0.

- The volume dimensions under the keyword `# Box_Info` must match the crystal dimensions.
- Under the keyword `# Start_Type`, the `read_config` or `add_to_config` options must be used.

GCMC simulations require the following additional parameters:

- Under the keyword `# Prob_Insertion` insertion method for the solid is `none`.
- Under the keyword `# Chemical_Potential_Info`, no chemical potential is required for the solid.

## 6.2 PDB file

A PDB configuration file for a zeolite can be created in the following ways, among others:

- Manually, with atomic coordinates from the literature. For example, the atomic coordinates of silicalite are published in Ref. [5].
- From a Crystallographic Information File (CIF), which can be downloaded from the Database of Zeolite Structures (http://www.iza-structure.org/databases/). From the home page, click on the menu "Framework Type" and select your zeolite. The website will display structural information about the zeolite and will have a link to download a CIF. The CIF contains information about the zeolite structure such as cell parameters, space groups, T and O atom coordinates. A CIF can be converted into a PDB file using either Mercury or VESTA, both of which are available to download for free. For example, using VESTA:
  1. From the File menu, click Open. Then download the CIF (e.g. MFI.cif)
  2. From the Objects menu, click Boundary. Enter the desired number of replicas along each axis. To output a single unit cell, enter -0 to 1 in the x, y and z ranges. To output a 2x2x2 crystal, enter -1 to 1 in the x, y and z ranges.
  3. From the File menu, click Export Data. Enter a filename ending with .pdb (e.g. MFI.pdb)

## 6.3 Molecular connectivity file

Since the solid structure will be rigid, no bond distances, angle parameters or dihedral parameters are needed in the MCF. The PDB file for the rigid solid does not list CONECT information, so the `mcfgen.py` script will not include bond, angle, or dihedral sections in the force field template (*.ff) or MCF. The number of fragments will be zero. Only nonbonded parameters are needed.

## 6.4 Fragment library files

The `library_setup.py` script will not create a fragment library since the MCF lists zero fragments.

## 6.5 Configuration xyz file

A simulation is initiated from an xyz file using the `read_config` or `add_to_config` options. Section 4.1.16 details the `read_config` and `add_to_config` options.

# Chapter 7

# Implementing the Metropolis Acceptance Criteria

All Monte Carlo moves are implemented in Cassandra to preserve detailed balance between each pair of microstates $m$ and $n$

$$\Pi_{mn}\ \alpha_{mn}\ p_m = \Pi_{nm}\ \alpha_{nm}\ p_n \tag{7.1}$$

where $\Pi_{mn}$ is the probability of accepting the move from microstate $m$ to microstate $n$, $\alpha_{mn}$ is the probability of attempting the move that will form $n$ from $m$, and $p_m$ is the probability of $m$ in the ensemble of interest.

In Cassandra, detailed balance is enforced via the Metropolis criterion

$$\Pi_{mn} = \min\left(1, \frac{\alpha_{nm}}{\alpha_{mn}}\frac{p_n}{p_m}\right) \tag{7.2}$$

The ratio in Eq. (7.2) will often involve an exponential, e.g. $e^{-\beta\Delta U}$. To preserve precision in the energy calculation, the acceptance probability is computed

$$\Pi_{mn} = \exp\left\{-\max\left[0, \ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right)\right]\right\} \tag{7.3}$$

The logarithm, defined in code as ln_pacc, is tested in the function accept_or_reject() which is defined in file accept_or_reject.f90. If ln_pacc is greater than 0 and less than a maximum numerical value, $\Pi_{mn}$ is computed and compared to a random number.

Code 7.1: accept_or_reject.f90

```
47  accept_or_reject = .FALSE.
48
49  IF (ln_pacc <= 0.0_DP) THEN
50
51     accept_or_reject = .TRUE.
52
53  ELSE IF ( ln_pacc < max_kBT) THEN
54
55     pacc = DEXP(-ln_pacc)
56
57     IF ( rranf() <= pacc ) THEN
58
59        accept_or_reject = .TRUE.
60
61     END IF
62
63  END IF
```

## 7.1 Canonical Monte Carlo

In the canonical ensemble, the number of molecules $N$, the volume $V$ and temperature $T$ are all constant. The position, orientation and conformation of a semi-flexible molecule with fixed bond-lengths containing $M$ atoms is given by a $2M+1$-dimensional vector $\mathbf{q}$. The positions, orientations and conformations of all $N$ molecules are denoted $\mathbf{q}^N$.

The probability of observing microstate $m$ with configuration $\mathbf{q}_m^N$ is

$$p_m = \frac{e^{-\beta U(\mathbf{q}_m^N)}}{Z(N,V,T)}\, d\mathbf{q}^N \tag{7.4}$$

where $\beta$ is the inverse temperature $1/k_BT$, $U$ is the potential energy, the differential volume $d\mathbf{q}^N$ is included to make $p_m$ dimensionless and $Z$ is the configurational partition function

$$Z(N,V,T) = \int e^{-\beta U(\mathbf{q}^N)} d\mathbf{q}^N. \tag{7.5}$$

The integral is over all $N(2M+1)$ degrees of freedom. The ratio of microstate probabilities follows from Eq. (7.4)

$$\begin{aligned} \frac{p_m}{p_n} &= \frac{e^{-\beta U(\mathbf{q}_m^N)} d\mathbf{q}^N / Z(N,V,T)}{e^{-\beta U(\mathbf{q}_n^N)} d\mathbf{q}^N / Z(N,V,T)} \\ &= e^{\beta(U_n - U_m)} = e^{\beta \Delta U} \end{aligned} \tag{7.6}$$

Table 7.1: Variable symbols and code names for translating and rotating a molecule

| Symbol | Code name |
|---|---|
| $\beta$ | beta(ibox) |
| $\Delta U$ | delta_e |

The configurational partition function $Z$ and differential volume $d\mathbf{q}^N$ both cancel, leaving only the ratio of Boltzmann factors.

New configurations are generated by attempting moves that translate, rotate and regrow a randomly selected molecule.

### 7.1.1 Translating a Molecule

A molecule is translated by moving its center of mass in each Cartesian direction by a random amount chosen from the uniform distribution on the interval [-$\delta r_{max}, \delta r_{max}$]. The maximum displacement $\delta r_{max}$ must be given in the input file. The translation move is symmetric in forward and reverse directions. That is, either microstate $n$ can be formed from microstate $m$ and vice versa by moving one molecule within $\delta r_{max}$ in each Cartesian direction, or microstate $n$ cannot be formed at all. As a result, $\alpha_{mn} = \alpha_{nm}$.

The acceptance probability for a translation move follows from Eq. (7.6)

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \ln\left(\frac{p_m}{p_n}\right) = \beta\Delta U \tag{7.7}$$

In Cassandra, the translation move is implemented in the subroutine Translate defined in move_translate.f90. The relevant lines from version 1.2 are quoted below. The variable names in the move_translate.f90 code are identified with the symbols from Eq. (7.7) in Table 7.1.

Code 7.2: move_translate.f90

```
266 ln_pacc = beta(ibox) * delta_e
267 accept = accept_or_reject(ln_pacc)
```

### 7.1.2 Rotating a Molecule

A linear molecule is rotated differently than a nonlinear molecule. A molecule is identified as linear if it is composed of 2 atoms or if all the angles are rigid with a bond angle of 180°. If the molecule is linear:

1. Pick three random angles: $\phi$ on $[-\pi, \pi]$, cos($\theta$) on [-1,1], and $\psi$ on $[-\pi, \pi]$.

2. With the origin at the molecule's center of mass, rotate by $\phi$ around $z$, rotate by $\theta$ around $x'$, and rotate by $\psi$ around $z'$, as shown in Fig. 7.1.

Even though three angles are randomly chosen, the probability of the resulting orientation is $d\cos(\theta)d\phi/4\pi$.

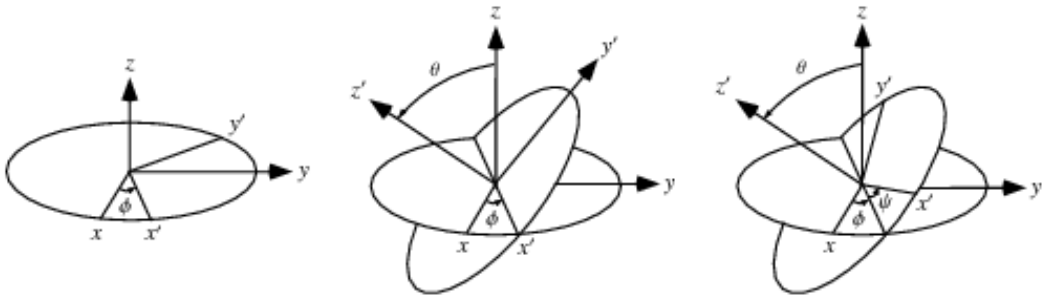


Figure 7.1: Procedure for rotating linear molecules.Image from mathworld.wolfram.com/EulerAngles.html.

If the molecule is nonlinear:

1. Randomly select an axis: $x$, $y$, or $z$.

2. Choose a random angular displacement $\delta\theta$ from $[-\delta\theta_{max}, \delta\theta_{max}]$. $\delta\theta_{max}$ must be given in the input file.

3. Rotate the molecule around a vector parallel to the selected axis and through its center of mass by $\delta\theta$.

In either case, the rotation move is symmetric, $\alpha_{mn} = \alpha_{nm}$, and the acceptance criteria is given by Eq. (7.7). The rotation move is implemented in subroutine Rotate defined in move_rotate.f90.

Code 7.3: move_rotate.f90

```
268 ln_pacc = beta(ibox) * delta_e
269 accept = accept_or_reject(ln_pacc)
```

### 7.1.3 Regrowing a Molecule

Internal degrees of freedom in flexible molecules are sampled by deleting one or more fragments from the molecule and replacing the deleted fragments with conformations from a reservoir of fragment conformations. If the molecule consists of only a single fragment (e.g, water, all atom methane, united atom propane, all atom cyclohexane), the entire molecule is deleted and replaced as follows:

1. Randomly select a molecule $i$ with uniform probability $1/N$, record its center-of-mass position and delete it.
2. Select a molecular conformation with Boltzmann probability $e^{-\beta U(\mathbf{q}_{int,n}^{(i)})}/Z_{int}$, where $\mathbf{q}_{int,n}^{(i)}$ are the internal bond or improper angles of molecule $i$ in microstate $n$ and $Z_{int}$ is the configurational partition function over internal degrees of freedom (see Eq. (7.13)).
3. Pick three random angles: $\phi$ on $[-\pi, \pi]$, $\cos(\theta)$ on [-1,1], and $\psi$ on $[-\pi, \pi]$. Rotate the molecule as shown in Fig. 7.1. The probability of the resulting orientation is $d\mathbf{q}_{rot}/Z_{rot}$, which for a nonlinear molecule is $d\cos(\theta)d\phi d\psi/8\pi^2$.
4. Place the molecule with the selected conformation and orientation at the same center-of-mass position as the deleted molecule.

Regrowing a monoatomic particle has no effect. Regrowing a linear molecule is the same as rotating it. The overall probability $\alpha_{mn}$ of regrowing a molecule with the selected orientation and conformation is

$$\alpha_{mn} = \frac{1}{N}\frac{d\mathbf{q}_{rot}}{Z_{rot}}\frac{e^{-\beta U(\mathbf{q}_n^{(i)})}d\mathbf{q}_{int}}{Z_{int}} \tag{7.8}$$

where $\mathbf{q}_n^{(i)}$ denotes the position, orientation and conformation of molecule $i$ in microstate $n$ and $U(\mathbf{q}_n^{(i)})$ is the potential energy of the isolated molecule $i$, i.e. the intramolecular potential energy. The reverse probability $\alpha_{nm}$ is identical except for the intramolecular potential energy $U(\mathbf{q}_m^{(i)})$ of molecule $i$ in microstate $m$. Using Eqs. (7.6) and (7.8), the acceptance criteria for the regrowth of a single fragment molecule is

$$\begin{aligned}\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) &= \beta\left[\left(U(\mathbf{q}_n^N) - U(\mathbf{q}_m^N)\right) - \left(U(\mathbf{q}_n^{(i)}) - U(\mathbf{q}_m^{(i)})\right)\right] \\ &= \beta\Delta U - \beta\Delta U_{int}^{(i)} = \beta\Delta U_{inter}^{(i)}\end{aligned} \tag{7.9}$$

Only the change in the intermolecular potential energy between molecule $i$ and the other $N-1$ molecules contributes to the acceptance criteria. The code that implements Eq. (7.9) is shown in Code 7.7 in Section 7.3.3.

If the molecule consists of more than one fragment (e.g., all atom ethane, all atom toluene, united atom butane), a bond is cut and the severed fragments are regrown using Configurational Bias Monte Carlo (CBMC). See Section 7.3.3 for more details.

### 7.1.4 Canonical Partition Function

In Sections 7.1.1-7.1.2, the microstate probability is normalized by the configuration partition function $Z$ because the only relevant degrees of freedom are configurational. In other ensembles, the full partition

function $Q$ appears, integrated over both configuration space $\mathbf{q}^N$ and momenta space $\mathbf{p}_q^N$

$$Q(N,V,T) = \frac{1}{h^{N(2M+1)}N!}\int e^{-\beta H(\mathbf{p}_q^N,\mathbf{q}^N)}\; d\mathbf{p}_q^N d\mathbf{q}^N \tag{7.10}$$

where the $2M+1$ momenta $\mathbf{p}_q$ are conjugate to the generalized coordinates $\mathbf{q}$. The momenta and configuration integrals are separable, and the single molecule momenta integrals are all identical.

$$\begin{aligned} Q(N,V,T) &= \frac{1}{N!}\left[\int e^{-\beta U(\mathbf{q}^N)}d\mathbf{q}^N\right]\left[\frac{1}{h^{2M+1}}\int e^{-\beta K(\mathbf{p}_q)}\; d\mathbf{p}_q\right]^N \\ &= \frac{1}{N!}Z(N,V,T)\left[\frac{Q(1,V,T)}{Z(1,V,T)}\right]^N \end{aligned} \tag{7.11}$$

where $Q(1,V,T)$ is the partition function of a single molecule in a box. The center of mass integrals for a single molecule are separable from the integrals over rotational and internal degrees of freedom:

$$Q(1,V,T) = Q_{com}Q_{rot+int} = V\Lambda^{-3}Q_{rot+int} \tag{7.12}$$

where $\Lambda$ is the de Broglie wavelength of the molecule and the rotational and internal momenta integrals in $Q_{rot+int}$ are not separable since the moments of inertia will depend on the conformation adopted by the molecule. The configurational partition function is further separable into center of mass (translational), orientational and internal degrees of freedom:

$$Z(1,V,T) = VZ_{rot}Z_{int} \tag{7.13}$$

where the volume $V$ is the translational partition function and $Z_{rot}$ equals $4\pi$ for a linear molecule and $8\pi^2$ for a nonlinear molecule.

## 7.2 Isothermal-Isobaric Monte Carlo

In the isothermal-isobaric ensemble, the number of particles $N$, the pressure $P$ and temperature $T$ are all constant while the volume $V$ and energy $E$ fluctuate. The partition function is

$$\Delta(N,P,T) = \int e^{-\beta PV}Q(N,V,T)dV \tag{7.14}$$

where $Q$ is dimensionless and $\Delta$ has dimensions of volume. The kinetic contribution to $\Delta$ is independent of the pressure or volume and consequently separable from the configurational contribution, $\Delta_Z$

$$\Delta_Z(N,P,T) = \int e^{-\beta PV} Z(N,V,T) dV \tag{7.15}$$

The probability of the system having volume $V$ is

$$p(V) = \frac{Z(N,V,T)e^{-\beta PV}}{\Delta_Z(N,P,T)} dV \tag{7.16}$$

The probability of observing microstate $m$ with configuration $\mathbf{q}_m^N$ and volume $V_m$ is

$$\begin{aligned} p_m &= \frac{e^{-\beta U(\mathbf{q}_m^N)} d\mathbf{q}_m^N}{Z(N,V_m,T)} \; \frac{Q(N,V_m,T)e^{-\beta PV_m} dV}{\Delta(N,P,T)} \\ &= \frac{e^{-\beta U_m - \beta PV_m}}{\Delta_Z(N,P,T)} \; d\mathbf{q}_m^N dV \end{aligned} \tag{7.17}$$

where the differential element $d\mathbf{q}_m^N$ has subscript $m$ becuase it scales with the volume $V_m$. The ratio of microstate probabilities is

$$\frac{p_m}{p_n} = e^{\beta(U_n - U_m) + \beta P(V_n - V_m)} \left( \frac{d\mathbf{q}_m}{d\mathbf{q}_n} \right)^N = e^{\beta \Delta U + \beta P \Delta V} \left( \frac{d\mathbf{q}_m}{d\mathbf{q}_n} \right)^N \tag{7.18}$$

### 7.2.1 Scaling the Volume

In Cassandra, new volumes are sampled as follows:

1. Pick a random volume $\Delta V$ with uniform probability from the interval $[-\delta V_{max}, \delta V_{max}]$. The trial volume is $V + \Delta V$.
2. Scale the box lengths uniformly.
3. Scale the center of mass of each molecule uniformly.

The probability of selecting $\Delta V$ is the same as selecting $-\Delta V$ which makes scaling the volume symmetric, $\alpha_{mn} = \alpha_{nm}$. Scaling the configurations changes the differential element $d\mathbf{q}_m^N$ surrounding configuration $\mathbf{q}_m^N$.

Table 7.2: Variable symbols and code names for volume scaling move.

| Symbol | Code name |
|---|---|
| $\beta$ | beta(this_box) |
| $\Delta U$ | delta_e |
| $P$ | pressure(this_box) |
| $\Delta V$ | delta_volume |
| $N$ | total_molecules |
| $V_n$ | box_list(this_box)%volume |
| $V_m$ | box_list_old%volume |

Only the molecular centers of mass change, so we can separate $d\mathbf{q}$ into 3 center of mass coordinates $d\mathbf{r}_{com}$ and $2M$-2 orientational and internal coordinates $d\mathbf{q}_{rot+int}$. The scaled center of mass positions are held constant, making $d\mathbf{r}_{com} = V d\mathbf{s}_{com}$. The acceptance probability for a volume scaling move is

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \ln\left(\frac{p_m}{p_n}\right) = \beta\Delta U + \beta P\Delta V + N\ln\left(\frac{V_m}{V_n}\right) \tag{7.19}$$

The volume scaling move is implemented in subroutine Volume_Change defined in move_volume.f90.

Code 7.4: move_volume.f90

```
424 ln_pacc = beta(this_box) * delta_e &
425         + beta(this_box) * pressure(this_box) * delta_volume &
426         - total_molecules * DLOG(box_list(this_box)%volume/box_list_old%volume)
427 accept = accept_or_reject(ln_pacc)
```

## 7.3 Grand Canonical Monte Carlo

In the grand canonical ensemble, the chemical potential $\mu$, the volume $V$ and temperature $T$ are held constant while the number of molecules $N$ and energy $E$ fluctuate. The partition function is

$$\Xi(\mu, V, T) = \sum_{N=0}^{\infty} Q(N, V, T)\ e^{\beta\mu N} \tag{7.20}$$

The probability of the system having $N$ molecules is

$$p(N) = \frac{Q(N, V, T)e^{\beta\mu N}}{\Xi(\mu, V, T)} \tag{7.21}$$

The probability of observing microstate $m$ with $N_m$ molecules and configuration $\mathbf{q}_m^{N_m}$ is

$$\begin{aligned} p_m &= \frac{e^{-\beta U(\mathbf{q}_m^{N_m})} d\mathbf{q}^{N_m}}{Z(N_m,V,T)} \; \frac{Q(N_m,V,T) e^{\beta\mu N_m}}{\Xi(\mu,V,T)} \\ &= \frac{e^{-\beta U_m + \beta\mu N_m}}{\Xi(\mu,V,T)} \; \left[\frac{Q(1,V,T)}{Z(1,V,T)}\, d\mathbf{q}\right]^{N_m} \end{aligned} \tag{7.22}$$

Note that Eq. (7.22) does not contain the factorial $N_m!$ that accounts for indistinguishable particles. In a simulation, particles *are* distinguishable: they are numbered and specific particles are picked for MC moves. The ratio of microstate probabilities is

$$\frac{p_m}{p_n} = e^{\beta\Delta U - \beta\mu\Delta N} \; \left[\frac{Q(1,V,T)}{Z(1,V,T)}\, d\mathbf{q}\right]^{-\Delta N} \tag{7.23}$$

Alternatively, Eq. (7.23) can be recast to use the fugacity $f$ instead of the chemical potential $\mu$. The relationship between $\mu$ and $f$ is

$$\mu = -k_B T \ln\left(\frac{Q(1,V,T)}{N}\right) = -k_B T \; \ln\left(\frac{Q(1,V,T)}{\beta f V}\right) \tag{7.24}$$

Inserting Eq. (7.24) into Eq. (7.23) yields

$$\frac{p_m}{p_n} = e^{\beta\Delta U} \; \left[\frac{\beta f V}{Z(1,V,T)}\, d\mathbf{q}\right]^{-\Delta N} \tag{7.25}$$

Fluctuations in the number of molecules are achieved by inserting and deleting molecules. A successful insertion increases the number of molecules from $N$ to $N + 1$, i.e. $\Delta N = 1$. A successful deletion decreases the number of molecules from $N$ to $N$ - 1, i.e. $\Delta N = -1$.

Random insertions and deletions (see Section 7.6) in the liquid phase typically have very high $\Delta U$ due to core overlap and dangling bonds, respectively, making the probability of acceptance very low. Instead, insertions in Cassandra are attempted using Configurational Bias Monte Carlo.

### 7.3.1 Inserting a Molecule with Configurational Bias Monte Carlo

In Configurational Bias Monte Carlo (CBMC), the molecular conformation of the inserted molecule is molded to the insertion cavity. First, the molecule is parsed into fragments such that each fragment is composed

of (a) a central atom and the atoms directly bonded to it (see Fig. 7.2), or (b) a ring of atoms and all the atoms directly bonded to them. Then, a position, orientation and molecular conformation of the molecule to be inserted are selected via the following steps:

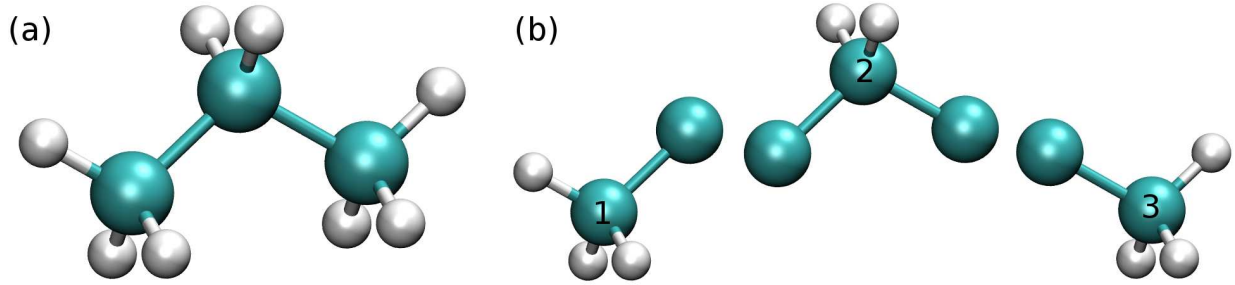


Figure 7.2: (a) An all-atom model of propane. (b) The same model as in (a), but parsed into three fragments.

1. Select the order in which each fragment of the $(N+1)$th molecule will be placed. The probability of the resulting sequence is $p_{seq}$. (See example in Table. 7.3.)
   (a) The first fragment $i$ is chosen with uniform probability $1/N_{frag}$.
   (b) Subsequent fragments must be connected to a previously chosen fragment and are chosen with the uniform probability $1/N_{cnxn}$, where the number of connections $N_{cnxn} = \sum_{ij} \delta_{ij} h_i (1 - h_j)$ is summed over all fragments $i$ and $j$. $h_i$ is 1 if fragment $i$ has been previously chosen and 0 otherwise. $\delta_{ij}$ is 1 if fragments $i$ and $j$ are connected and 0 otherwise.
2. Select a conformation for fragment $i$ with Boltzmann probability $e^{-\beta U(\mathbf{q}_{frag_i})} d\mathbf{q}_{frag_i} / Z_{frag_i}$, where $\mathbf{q}_{frag_i}$ are the internal degrees of freedom (angles and/or impropers) associated with fragment $i$.
3. Select an orientation with uniform probability $d\mathbf{q}_{rot}/Z_{rot}$.
4. Select a coordinate for the center of mass (COM) of fragment $i$:
   (a) Select $\kappa_{ins}$ trial coordinates $\mathbf{r}_k$, each with uniform probability $d\mathbf{r}/V$. Since one of the trial coordinates will be selected later, the individual probabilities are additive. The probability of the collection of trial coordinates is $\kappa_{ins} d\mathbf{r}/V$.
   (b) Compute the change in potential energy $\Delta U_k^{ins}$ of inserting fragment $i$ at each position $\mathbf{r}_k$ into configuration $\mathbf{q}_m^N$.
   (c) Select one of the trial coordinates with probability $e^{-\beta \Delta U_k^{ins}} / \sum_k e^{-\beta \Delta U_k^{ins}}$.
5. For each additional fragment $j$:
   (a) Select a fragment conformation with Boltzmann probability $e^{-\beta U(\mathbf{q}_{frag_j})} d\mathbf{q}_{frag_j} / Z_{frag_j}$
   (b) Select the first of $\kappa_{dih}$ trial dihedrals $\phi_0$ with uniform probability from the interval $[0, \frac{2\pi}{\kappa_{dih}})$. Additional trial dihedrals are equally spaced around the unit circle, $\phi_k = \phi_{k-1} + 2\pi/\kappa_{dih}$. The probability of selecting $\phi_0$ is $\kappa_{dih} d\phi / 2\pi$.

Table 7.3: Possible sequences and probabilities for inserting the fragments of the all-atom model of propane shown in Fig. 7.2.

| **Sequence** | $p_{seq}$ |
|---|---|
| 1 2 3 | 1/3 |
| 2 1 3 | 1/6 |
| 2 3 1 | 1/6 |
| 3 2 1 | 1/3 |

(c) Compute the change in potential energy $\Delta U_k^{dih}$ of attaching fragment $j$ to the growing molecule with each dihedral $\phi_k$.

(d) Select one of the trial dihedrals with probability $e^{-\beta\Delta U_k^{dih}} / \sum_k e^{-\beta\Delta U_k^{dih}}$.

The overall probability $\alpha_{mn}$ of attempting the insertion with the selected position, orientation and conformation is

$$\alpha_{mn} = p_{seq} \frac{d\mathbf{q}_{rot}}{Z_{rot}} \frac{\kappa_{ins} d\mathbf{r}}{V} \frac{e^{-\beta\Delta U_k^{ins}}}{\sum_k e^{-\beta\Delta U_k^{ins}}} \times \left[ \prod_{i=1}^{N_{frag}} \frac{e^{-\beta U(\mathbf{q}_{frag_i})} d\mathbf{q}_{frag_i}}{Z_{frag_i}} \right] \left[ \prod_{j=1}^{N_{frag}-1} \frac{\kappa_{dih} d\phi}{2\pi} \frac{e^{-\beta\Delta U_k^{dih}}}{\sum_k e^{-\beta\Delta U_k^{dih}}} \right] \tag{7.26}$$

$$= p_{seq}\ p_{bias} \frac{e^{-\beta U(\mathbf{q}_{frag})} d\mathbf{q}}{V Z_{rot} Z_{frag} \Omega_{dih}} \tag{7.27}$$

where $Z_{frag} = \prod_i Z_{frag_i}$ is the configurational partition function over degrees of freedom internal to each fragment, $U(\mathbf{q}_{frag}) = \sum_i U(\mathbf{q}_{frag_i})$ is the summed potential energy of each of the (disconnected) fragments, $\Omega_{dih} = (2\pi)^{N_{frag}-1}$ and $p_{bias}$ is

$$p_{bias} = \frac{\kappa_{ins}\ e^{-\beta\Delta U_k^{ins}}}{\sum_k e^{-\beta\Delta U_k^{ins}}} \left[ \prod_{j=1}^{N_{frag}-1} \frac{\kappa_{dih}\ e^{-\beta\Delta U_k^{dih}}}{\sum_k e^{-\beta\Delta U_k^{dih}}} \right] \tag{7.28}$$

Note that the term $V Z_{rot} Z_{frag} \Omega_{dih}$ in the denominator of Eq. (7.27) differs from $Z(1, V, T) = V Z_{rot} Z_{int}$.

In the reverse move, 1 of the $N + 1$ particles is randomly selected for deletion. The probability $\alpha_{nm}$ of picking the molecule we just inserted is

$$\alpha_{nm} = \frac{1}{N+1} \tag{7.29}$$

Combining Eqs. (7.27) and (7.29) with Eq. (7.23) or Eq. (7.25) gives the acceptance probability for a CBMC insertion move

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U - U(\mathbf{q}_{frag,n}^{(N+1)})\right] - \beta\mu' + \ln\left(\frac{(N+1)\Lambda^3}{V}\right) + \ln\left(p_{seq}p_{bias}\right) \tag{7.30}$$

$$= \beta\left[\Delta U - U(\mathbf{q}_{frag,n}^{(N+1)})\right] + \ln\left(\frac{N+1}{\beta f' V}\right) + \ln\left(p_{seq}p_{bias}\right) \tag{7.31}$$

where $\mu'$ and $f'$ are, respectively, a shifted chemical potential and a skewed fugacity,

$$\mu' = \mu + k_B T \ln\left(Q_{rot+int}\frac{Z_{frag}\Omega_{dih}}{Z_{int}}\right) \tag{7.32}$$

$$f' = f\frac{Z_{frag}\Omega_{dih}}{Z_{int}} \tag{7.33}$$

All of the terms in Eqs. (7.32) and (7.33) are intensive. GCMC simulations using Eqs. (7.30) and (7.31) will converge to the same average density regardless of the simulation volume $V$. However, the values of $\mu'$ or $f'$ that correspond to the converged density will *not* match tabulated values of $\mu$ or $f$ computed from experimental data.

Note that the term $Z^{IG}/\Omega$ from Macedonia *et al* [6]. would be equivalent to $Z_{int}/\Omega_{frag}\Omega_{dih}$ in the nomenclature used here. The configurational partition function of the disconnected fragments integrates over a Boltzmann factor, $Z_{frag} = \int e^{-\beta U(\mathbf{q}_{frag})} d\mathbf{q}_{frag}$, whereas the term $\Omega_{frag} = \int d\mathbf{q}_{frag}$ does not.

In Cassandra, the insertion move is implemented in the subroutine Insertion in move_insert.f90. The relevant lines from version 1.2 are quoted below. The variable names in the move_insert.f90 code are identified with symbols in Table 7.4.

Table 7.4: Variable symbols and code names for inserting a molecule

| Symbol | Code name |
|---|---|
| $\beta$ | beta(ibox) |
| $\Delta U$ | dE |
| $U(\mathbf{q}_{frag})$ | dE_frag |
| $\ln(p_{seq}p_{bias})$ | ln_pbias |
| $\mu'$ | species_list(is)%chem_potential |
| $N$ | nmols(is,this_box) |
| $V$ | box_list(this_box)%volume |
| $\Lambda$ | species_list(is)%de_broglie(this_box) |

Code 7.5: move_insert.f90

```
335   ! change in energy less energy used to bias selection of fragments
336   dE_frag = E_angle + nrg_ring_frag_tot
337   ln_pacc = beta(ibox) * (dE - dE_frag)
338
339   ! chemical potential
340   ln_pacc = ln_pacc - species_list(is)%chem_potential * beta(ibox)
341
342   ! bias from CBMC
343   ln_pacc = ln_pacc + ln_pbias
344
345   ! density
346   ln_pacc = ln_pacc + DLOG(REAL(nmols(is,ibox),DP)) &
347                     + 3.0_DP*DLOG(species_list(is)%de_broglie(ibox)) &
348                     - DLOG(box_list(ibox)%volume)
349
350   accept = accept_or_reject(ln_pacc)
```

Note that GCMC simulations using fugacities are currently not supported in Cassandra. This feature will be implemented in a future release.

### 7.3.2 Deleting a Molecule that was Inserted via Configurational Bias Monte Carlo

The probability $\alpha_{mn}$ of choosing a molecule to delete is

$$\alpha_{mn} = \frac{1}{N} \tag{7.34}$$

The probability of the reverse move $\alpha_{nm}$ requires knowledge of the sequence and biasing probabilities $p_{seq}$ and $p_{bias}$ that would have been used to place the molecule if it was being inserted. $p_{seq}$ and $p_{bias}$ can be calculated using the following procedure:

1. Select the fragment order using the same procedure for inserting a molecule. The probability of the resulting sequence is $p_{seq}$.
2. The first fragment in the sequence is fragment $j$. Calculate the intramolecular potential energy of fragment $j$'s current conformation, $U(\mathbf{q}_{frag_j})$. The probability of this conformation is Boltzmann $e^{-\beta U(\mathbf{q}_{frag_j})} d\mathbf{q}_{frag_j}/Z_{frag_j}$.
3. The probability of the fragment's current orientation is $d\mathbf{q}_{rot}/Z_{rot}$.
4. Calculate the weight of the fragment's current center of mass (COM) coordinates:
   (a) Compute the interaction potential energy $\Delta U^{ins}$ between fragment $j$ and the other $N-1$ molecules.
   (b) Select $\kappa_{ins}-1$ trial coordinates $\mathbf{r}_k$, each with uniform probability $d\mathbf{r}/V$.
   (c) Calculate the weight of the fragment's current COM amongst the trial coordinates, $e^{-\beta\Delta U^{ins}}/\sum_k e^{-\beta\Delta U_k^{ins}}$.
5. For each additional fragment $j$:
   (a) Calculate the intramolecular potential energy of fragment $j$'s current conformation, $U(\mathbf{q}_{frag_j})$. The weight of this conformation in the Boltzmann distribution is $e^{-\beta U(\mathbf{q}_{frag_j})} d\mathbf{q}_{frag_j}/Z_{frag_j}$.
   (b) Calculate the interaction potential energy $\Delta U^{dih}$ between fragment $j$, on the one hand, and fragments $i$ through $j-1$ and the other $N-1$ molecules.
   (c) Calculate the current dihedral $\phi_0$ of fragment $j$. Compute the interaction potential energy $\Delta U_k^{dih}$ at $\kappa_{dih}-1$ trial dihedrals $\phi_k = \phi_{k-1} + 2\pi/\kappa_{dih}$.
   (d) Compute the weight of $\phi_0$ amongst the trial dihedrals, $e^{-\beta\Delta U^{dih}}/\sum_k e^{-\beta\Delta U_k^{dih}}$.

The overall probability $\alpha_{nm}$ is

$$\alpha_{nm} = p_{seq}\ p_{bias}\ \frac{e^{-\beta U(\mathbf{q}_{frag})} d\mathbf{q}}{V Z_{rot} Z_{frag} \Omega_{dih}}. \tag{7.35}$$

The acceptance criteria for deleting a molecule inserted via CBMC is

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U + U(\mathbf{q}^{(i)}_{frag,m})\right] + \beta\mu' + \ln\left(\frac{V}{N\Lambda^3}\right) - \ln\left(p_{seq}p_{bias}\right) \tag{7.36}$$

$$= \beta\left[\Delta U + U(\mathbf{q}^{(i)}_{frag,m})\right] + \ln\left(\frac{\beta f' V}{N}\right) - \ln\left(p_{seq}p_{bias}\right) \tag{7.37}$$

Table 7.5: Variable symbols and code names for deleting a molecule

| Symbol | Code name |
|---|---|
| $\beta$ | beta(ibox) |
| $\Delta U$ | dE |
| $U(\mathbf{q}_{frag})$ | dE_frag |
| $ln(p_{seq}p_{bias})$ | ln_pbias |
| $\mu'$ | species_list(is)%chem_potential |
| $N$ | nmols(is,this_box) |
| $V$ | box_list(this_box)%volume |
| $\Lambda$ | species_list(is)%de_broglie(this_box) |

In Cassandra, the deletion move is implemented in the subroutine Deletion in move_delete.f90. The relevant lines are quoted below. The variable names in move_delete.f90 code are identified with symbols in Table 7.5.

Code 7.6: move_delete.f90

```
317   ! change in energy less energy used to bias fragment selection
318   dE_frag = - E_angle - nrg_ring_frag_tot
319   ln_pacc = beta(ibox) * (dE - dE_frag)
320
321   ! chemical potential
322   ln_pacc = ln_pacc + beta(ibox) * species_list(is)%chem_potential
323
324   ! CBMC bias probability
325   ln_pacc = ln_pacc - ln_pbias
326
327   ! dimensionless density
328   ln_pacc = ln_pacc + DLOG(box_list(ibox)%volume) &
329                     - DLOG(REAL(nmols(is,ibox),DP)) &
330                     - 3.0_DP*DLOG(species_list(is)%de_broglie(ibox))
331
332   accept = accept_or_reject(ln_pacc)
```

Note that GCMC simulations using fugacities are currently not supported in Cassandra. This feature will be implemented in a future release.

### 7.3.3 Regrowing a Molecule with Configurational Bias Monte Carlo

Regrowing a molecule that has more than one fragment is a combination deletion and insertion move. Starting from microstate $m$:

1. Randomly select a molecule with uniform probability $1/N$.

2. Randomly select a bond to cut on the selected molecule with uniform probability $1/N_{bonds}$.
3. Delete the fragments on one side of the bond or the other with equal probability. The number of deleted fragments is $N_{del}$.
4. Reinsert the deleted fragments using the CBMC procedures for selecting the order of inserting the fragments, choosing a fragment conformation, and a connecting dihedral value (see Section 7.3.1).

The overall probability $\alpha_{mn}$ of attempting to regrow the molecule with the selected conformation is

$$\begin{aligned}\alpha_{mn} &= \frac{p_{seq}}{NN_{bonds}} \left[\prod_{j=1}^{N_{del}} \frac{e^{-\beta U(\mathbf{q}_{frag_j}^{(i)})} d\mathbf{q}_{frag_j}}{Z_{frag_j}}\right] \left[\prod_{j=1}^{N_{del}} \frac{\kappa_{dih} d\phi}{2\pi} \frac{e^{-\beta \Delta U_k^{dih}}}{\sum_k e^{-\beta \Delta U_k^{dih}}}\right] \\ &= \frac{p_{seq}}{NN_{bonds}} \frac{e^{-\beta U(\mathbf{q}_{del,n}^{(i)})} d\mathbf{q}}{Z_{del}\Omega_{del}} p_{forward} \end{aligned} \tag{7.38}$$

where $Z_{del} = \prod_i Z_{frag_j}$ is the configurational partition function over degrees of freedom internal to the deleted fragments, $U(\mathbf{q}_{del,n}^{(i)}) = \sum_j U(\mathbf{q}_{frag_j})$ is the summed potential energy of each deleted fragment with the conformations in microstate $n$, $\Omega_{del} = (2\pi)^{N_{del}}$ and $p_{forward}$ is the biasing probability

$$p_{forward} = \prod_{j=1}^{N_{del}} \frac{\kappa_{dih}\ e^{-\beta \Delta U_k^{dih}}}{\sum_k e^{-\beta \Delta U_k^{dih}}} \tag{7.39}$$

The reverse move is identical except for the potential energy of the deleted fragments $U(\mathbf{q}_{del,m}^{(i)})$ in microstate $m$ and the biasing probability $p_{reverse}$ which will depend on the values of the connecting dihedrals. Using Eqs. (7.6) and (7.38), the acceptance criteria is:

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\left(U(\mathbf{q}_n^N) - U(\mathbf{q}_{del,n}^{(i)})\right) - \left(U(\mathbf{q}_m^N) - U(\mathbf{q}_{del,m}^{(i)})\right)\right] + \ln\left(\frac{p_{forward}}{p_{reverse}}\right) \tag{7.40}$$

Eq. (7.40) is implemented in subroutine cut_N_grow() in file move_regrow.f90.

Code 7.7: move_regrow.f90

```
426   ln_pacc = beta(ibox) * (delta_e_n - nrg_ring_frag_forward) &
427           - beta(ibox) * (delta_e_o - nrg_ring_frag_reverse) &
428           + ln_pfor - ln_prev
429
430   accept = accept_or_reject(ln_pacc)
```

Table 7.6: Variable symbols and code names for regrowing a molecule

| Symbol | Code name |
|---|---|
| $\beta$ | beta(ibox) |
| $U(\mathbf{q}_n^N) - U(\mathbf{q}_{del,n}^{(i)})$ | delta_e_n - nrg_ring_frag_forward |
| $U(\mathbf{q}_m^N) - U(\mathbf{q}_{del,m}^{(i)})$ | delta_e_o - nrg_ring_frag_reverse |
| $ln(p_{forward})$ | ln_pfor |
| $ln(p_{reverse})$ | ln_prev |

## 7.4 Gibbs Ensemble Monte Carlo

The Gibbs Ensemble Monte Carlo method is a standard technique for studying phase equilibria of pure fluids and mixtures. It is often used to study vapor-liquid equilibria due to its intuitive physical basis. In Cassandra, the NVT and NPT versions of the Gibbs Ensemble (GEMC-NVT and GEMC-NPT) are implemented. The GEMC-NVT method is suitable for simulating vapor liquid equilibria of pure systems, since pure substances require the specification of only one intensive variable (temperature) to completely specify a state of two phases. By contrast, mixtures require the specification of an additional degree of freedom (pressure). Thus, in the GEMC-NPT method, the pressure is specified in addition to temperature.

The partition functions and microstate probabilities are derived for GEMC-NVT and GEMC-NPT in sections 7.4.1 and 7.4.2, respectively. In both GEMC-NVT and GEMC-NPT, thermal equilibrium is attained by performing translation, rotation and regrowth moves. The acceptance rules for these moves are identical to those presented in sections 7.1.1, 7.1.2, 7.1.3 and 7.3.3. Pressure equilibrium is achieved by exchanging volume, in the case of GEMC-NVT, and independently changing the volume of each box, in the case of GEMC-NPT. The acceptance rule for the exchanging volume in GEMC-NVT is derived and its Cassandra implementation is presented in section 7.4.3. The acceptance rule for swapping a molecule in either GEMC-NVT or GEMC-NPT are derived in section 7.4.4.

### 7.4.1 Gibbs Ensemble-NVT

In the GEMC-NVT method, there are two boxes A and B. To achieve phase equilibrium, the boxes are allowed to exchange volume and particles under the constraint of constant total volume ($V^t = V^A + V^B$) and constant number of particles ($N^t = N^A + N^B$). The partition function is

$$Q_{GE}\left(N^t, V^t, T\right) = \sum_{N^A=0}^{N^t} \int_0^{V^t} dV^A \; Q(N^A, V^A, T) \; Q(N^t - N^A, V^t - V^A, T) \tag{7.41}$$

where $Q(N, V, T)$ is the canonical partition function given in Eq. 7.10. Since both boxes are maintained

at the same temperature the kinetic contribution of each molecule is independent of the box in which it is located. The configurational partition function $Z_{GE}$ is defined by separating the momenta integrals from the configurational integrals, volume integrals and molecular sums

$$Z_{GE}\left(N^t, V^t, T\right) = \sum_{N^A=0}^{N^t} \int_0^{V^t} dV^A \; Z(N^A, V^A, T) \; Z(N^t - N^A, V^t - V^A, T) \tag{7.42}$$

The probability of microstate $m$ in the NVT Gibbs ensemble is

$$p_m = \frac{e^{-\beta U^A\left(\mathbf{q}^{N^A}\right) - \beta U^B\left(\mathbf{q}^{N^B}\right)} d\mathbf{q}^{N^A} d\mathbf{q}^{N^B} dV^A}{Z_{GE}(N^t, V^t, T)} \tag{7.43}$$

Note that the molecule number factorials are not included in equation 7.43, as particles are distinguishable in a simulation (see also equation 7.22).

For two microstates $m$ and $n$ that differ only by a thermal move of a molecule in box A, the ratio of microstate probabilities is

$$\frac{p_m}{p_n} = e^{\beta \Delta U^A} \tag{7.44}$$

similar to Eq. 7.6. As a result, thermal moves have the same acceptance rule in GEMC-NVT as they do in other ensembles. The differential elements $d\mathbf{q}$ will likewise cancel from the acceptance criteria when swapping a molecule between boxes. When exchanging volume, however, the differential elements will reduce to a ratio of the old volume to the new, as shown in section 7.4.3.

### 7.4.2 Gibbs Ensemble-NPT

The GEMC-NPT method is only valid for sampling phase equilibria in multicomponent systems. It is similar to GEMC-NVT, except that the volume of each box fluctuates independently. Consequently, the total volume of the system is not constant and the pressure must be specified in addition to the temperature. This is consistent with the Gibbs phase rule for mixtures, which requires the specification of two intensive variables (e.g. pressure and temperature) to fully specify a state with two phases.

The partition function is

$$\Delta_{GE}\left(\{N^t\},P,T\right)=\sum_{N_1^A=0}^{N_1^t}...\sum_{N_s^A=0}^{N_s^t}\Delta(\{N^A\},P,T)\;\Delta(\{N^t-N^A\},P,T) \tag{7.45}$$

where $\{N\}$ is the number of molecules of each species, $\Delta(N,P,T)$ is the multicomponent analog to Eq. 7.14, and there is a separate sum for each species over the number of molecules in box A. The kinetic contribution to $\Delta_{GE}$ can be separated giving the configurational partition function

$$\Delta_{Z,GE}\left(N^t,P,T\right)=\sum_{N_1^A=0}^{N_1^t}...\sum_{N_s^A=0}^{N_s^t}\Delta_Z(N^A,P,T)\;\Delta_Z(N^t-N^A,P,T) \tag{7.46}$$

where $\Delta_Z(N,P,T)$ is the multicomponent analog to Eq. 7.15. The probability of microstate $m$ in this ensemble is

$$p_m=\frac{e^{-\beta U^A-\beta U^B-\beta PV^A-\beta PV^B}dV^AdV^B}{\Delta_{Z,GE}(N^t,P,T)}\prod_{s=1}^{N_{species}}\left[d\mathbf{q}_s^A\right]^{N_s^A}\left[d\mathbf{q}_s^B\right]^{N_s^B} \tag{7.47}$$

Similar to GEMC-NVT, the ratio of probabilities between microstates that differ by only a thermal move in box A is

$$\frac{p_m}{p_n}=e^{\beta\Delta U^A} \tag{7.48}$$

Volume changes are only attempted on one box at a time. The ratio of probabilities between microstates that differ only by the volume of box A is

$$\frac{p_m}{p_n}=e^{\beta\Delta U^A}+\left(\frac{V_m^A}{V_n^A}\right)^{N^A} \tag{7.49}$$

similar to Eq. 7.18. As a result, volume moves in GEMC-NPT have the same acceptance criteria as in the NPT ensemble (see Eq. 7.19).

Table 7.7: Variable symbols and code names for the volume scaling move in the GEMC-NVT method.

| Symbol | Code name |
|---|---|
| $\beta^A$ | beta(box1) |
| $\beta^B$ | beta(box2) |
| $\Delta U^A$ | delta_e_1 |
| $\Delta U^B$ | delta_e_2 |
| $N^A$ | tot_mol_box_1 |
| $N^B$ | tot_mol_box_2 |
| $V_m^A$ | box_list(box1)%volume |
| $V_m^B$ | box_list(box2)%volume |
| $V_n^A$ | box_list_old_1%volume |
| $V_n^B$ | box_list_old_2%volume |

### 7.4.3 Volume Exchange Moves

In GEMC-NVT, volume is exchanged between the two boxes to achieve pressure equilibrium using a symmetric volume move, $\alpha_{mn} = \alpha_{nm}$. If box A is shrunk by $\Delta V$, then box B grows by $\Delta V$ and vice versa. $\Delta V$ is chosen from a uniform distribution with probability $1/\delta V_{max}$, where $\delta V_{max}$ is an adjustable parameter. The scaled center of mass positions of each molecule are held constant, introducing a ratio of volumes into the acceptance criteria similar to Eq. 7.19.

The acceptance rule is derived from equation 7.43 and yields

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \ln\left(\frac{p_m}{p_n}\right) = \beta\Delta U^A + \beta\Delta U^B + N^A \ln\left(\frac{V_m^A}{V_n^A}\right) + N^B \ln\left(\frac{V_m^B}{V_n^B}\right) \tag{7.50}$$

This acceptance rule is implemented in the file move_vol_swap.f90 as follows

Code 7.8: move_vol_swap.f90

```
402 ln_pacc = beta(box_grw) * delta_e_1 + beta(box_shk) * delta_e_2 &
403         - REAL(SUM(nmols(:,box_grw)),DP) * DLOG( box_list(box_grw)%volume /
              box_list_old_1%volume) &
404         - REAL(SUM(nmols(:,box_shk)),DP) * DLOG( box_list(box_shk)%volume /
              box_list_old_2%volume)
```

### 7.4.4 Molecule Exchange Moves

In either GEMC-NVT or GEMC-NPT, molecules are swapped between the two boxes to equalize the chemical potential of each species. The ratio of probabilities for microstates that differ only by swapping a molecule of species $s$ from box *out* to box *in* is

$$\frac{p_m}{p_n} = e^{\beta\Delta U^A + \beta\Delta U^B} \frac{d\mathbf{q}_s^{out}}{d\mathbf{q}_s^{in}} \tag{7.51}$$

where the differential elements $d\mathbf{q}$ will cancel from the acceptance criteria by similar terms in $\alpha_{mn}/\alpha_{nm}$. The particle swap is not symmetric since each molecule is inserted and deleted using configurational bias. The forward probability $\alpha_{mn}$ follows from the steps used to swap a molecule:

1. Pick a box *out* with probability $p_{box}$, where $p_{box}$ is
   (a) the ratio of molecules in box, $N^{out}/N^t$ (default)
   (b) a fixed probability given in the input file
2. If necessary, pick a species $s$ with probability $p_{spec}$, where $p_{spec}$ is
   (a) the ratio of molecules of species $s$ in box *out*, $N_s^{out}/N^{out}$ (default)
   (b) a fixed probability given in the input file
3. Pick a molecule of species $s$ from the box *out* with uniform probability, $1/N_s^{out}$
4. Insert molecule in box *in* using protocol presented in section 7.3.1

If the default probabilities are used at each step, then a swap is attempted for each molecule with uniform probability

$$\frac{N^{out}}{N^t}\frac{N_s^{out}}{N^{out}}\frac{1}{N_s^{out}} = \frac{1}{N^t} \tag{7.52}$$

The attempt probability of generating configuration $n$

$$\alpha_{mn} = p_{out,m}p_{spec,m}\frac{1}{N_{s,m}^{out}}p_{seq}\ p_{bias,n}\ \frac{e^{-\beta U^{in}(\mathbf{q}_{frag,n})}d\mathbf{q}_s^{in}}{V^{in}Z_{rot}Z_{frag}\Omega_{dih}} \tag{7.53}$$

Table 7.8: Variable symbols and code names for the particle transfer move in the GEMC-NVT method.

| **Symbol** | **Code name** |
|---|---|
| $\beta^A$ | beta(box_out) |
| $\beta^B$ | beta(box_in) |
| $\Delta U^A$ | -delta_e_out |
| $\Delta U^B$ | delta_e_in |
| $U^{in}(\mathbf{q}_{frag,n})$ | e_angle_in + nrg_ring_frag_in |
| $U^{out}(\mathbf{q}_{frag,m})$ | e_angle_out + nrg_ring_frag_out |
| $V^{out}$ | box_list(box_out)%volume |
| $V^{in}$ | box_list(box_in)%volume |
| $ln(p_{bias,n})$ | ln_pfor |
| $ln(p_{bias,m})$ | ln_prev |
| $p_{out,m}p_{spec,m}$ | P_forward |
| $p_{out,n}p_{spec,n}$ | P_reverse |

where $p_{bias}$ is defined in Eq. 7.28. The reverse probability $\alpha_{nm}$ is calculated similarly. The acceptance rule is

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \ln\left(\frac{p_{out,m}}{p_{out,n}}\frac{p_{spec,m}}{p_{spec,n}}\frac{p_{bias,n}}{p_{bias,m}}\frac{N^{in}_{s,n}+1}{N^{out}_{s,m}}\frac{V^{out}}{V^{in}}\right) - \beta U^{in}(\mathbf{q}_{frag,n}) + \beta U^{out}(\mathbf{q}_{frag,m}) + \beta\Delta U^{out} + \beta\Delta U^{in} \tag{7.54}$$

where $p_{seq}$ does not appear since the same fragment regrowth sequence is used in the forward and reverse moves. The molecule swap move is implemented in the file move_mol_swap.f90 as follows

Code 7.9: move_mol_swap.f90

```
587 delta_e_in_pacc = delta_e_in
588 delta_e_out_pacc = delta_e_out
589
590 delta_e_in_pacc = delta_e_in_pacc - e_angle_in - nrg_ring_frag_in
591 delta_e_out_pacc = delta_e_out_pacc - e_angle_out - nrg_ring_frag_out
```

```
598 ln_pacc = beta(box_in)*delta_e_in_pacc - beta(box_out)*delta_e_out_pacc
599
600 ln_pacc = ln_pacc - DLOG(box_list(box_in)%volume) &
601                   + DLOG(box_list(box_out)%volume) &
602                   - DLOG(REAL(nmols(is,box_out),DP)) &
603                   + DLOG(REAL(nmols(is,box_in) + 1, DP))
604
605 ln_pacc = ln_pacc + ln_pfor - ln_prev &
606                   + DLOG(P_forward / P_reverse)
607
608 accept = accept_or_reject(ln_pacc)
```

## 7.5 Multicomponent Systems

Excluding section 7.4.2, the acceptance rules for all the Monte Carlo techniques expressed in this chapter have been developed for pure component systems. The Monte Carlo moves and acceptance criteria for multicomponent systems are straightforward extensions of the pure component moves. The only modification needed to translate, rotate and regrow molecules is to first select a species. In these moves, a species is selected randomly in proportion to its mole fraction $N_i/N$. When inserting and deleting a molecule, the mole fractions of each species change. In these cases, a species in a multicomponent system is selected instead with uniform probability $1/N_{species}$. In either case, species selection is symmetric for both forward and reverse moves and so cancels from the acceptance criterion.

## 7.6 Appendix

### 7.6.1 Inserting a Molecule Randomly

To insert a molecule, a position, orientation and molecular conformation must each be selected. The probability of inserting the new molecule at a random location is $d\mathbf{r}/V$, where $d\mathbf{r}$ is a Cartesian volume element of a single atom. The probability of choosing the molecule orientation is $d\mathbf{q}_{rot}/Z_{rot}$, which for a linear molecule is $d\cos(\theta)d\phi/4\pi$ and for a nonlinear molecule is $d\cos(\theta)d\phi d\psi/8\pi^2$. The probability of the molecule conformation only depends on the remaining $2M-5$ internal bond angles, dihedral angles and improper

angles $\mathbf{q}_{int}$. A thermal ensemble of configurations is Boltzmann distributed $e^{-\beta U(\mathbf{q}_{int})}/Z_{int}$. The overall probability $\alpha_{mn}$ is

$$\alpha_{mn} = \frac{d\mathbf{r}}{V}\,\frac{d\mathbf{q}_{rot}}{Z_{rot}}\,\frac{e^{-\beta U(\mathbf{q}_{int,N+1,n})}}{Z_{int}}\,d\mathbf{q}_{int} = \frac{e^{-\beta U(\mathbf{q_{N+1,n}})}}{Z(1,V,T)}\,d\mathbf{q}. \tag{7.55}$$

where we have used Eq. (7.13) to recover $Z(1,V,T)$ and recognized that only internal degrees of freedom contribute to the potential energy of the isolated $N+1$th molecule in microstate $n$, $U(\mathbf{q}_{N+1,n}) = U(\mathbf{q}_{int,N+1,n})$. For a point particle with no rotational or internal degrees of freedom, $\alpha_{mn}$ reduces to $d\mathbf{r}/V$. For molecules with internal flexibility, a library of configurations distributed according to $e^{-\beta U(\mathbf{q}_{int})}/Z_{int}$ can be generated from a single molecule MC simulation. In the reverse move, 1 of the $N+1$ particles is randomly selected for deletion. The probability $\alpha_{nm}$ of picking the molecule we just inserted is

$$\alpha_{nm} = \frac{1}{N+1} \tag{7.56}$$

The acceptance probability for a random insertion move is

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U - U(\mathbf{q}_{N+1})\right] - \beta\mu + \ln\left(\frac{N+1}{Q(1,V,T)}\right) \tag{7.57}$$

where $U(\mathbf{q}_{N+1})$ is the intramolecular potential energy of the inserted molecule. $Q(1,V,T)$ is typically not known *a priori*, nor is it easily estimated. Substituting Eq. (7.12) into Eq. (7.57) and absorbing $Q_{rot+int}$ into a shifted chemical potential $\mu'$

$$\mu' = \mu - k_B T \ln(Q_{rot+int}) \tag{7.58}$$

gives the acceptance criteria for inserting a molecule

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U - U(\mathbf{q}_{N+1})\right] - \beta\mu' + \ln\left(\frac{(N+1)\Lambda^3}{V}\right). \tag{7.59}$$

The terms absorbed into $\mu'$ are intensive and therefore GCMC simulations using Eq. (7.59) will converge to a specific average density. However, the value of $\mu'$ that corresponds to the converged density will *not* match tabulated values of $\mu$ computed from experimental data.

Substituting Eq. (7.24) into Eq. (7.57) gives

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U - U(\mathbf{q}_{N+1})\right] + \ln\left(\frac{N+1}{\beta f V}\right) \tag{7.60}$$

where no terms have been absorbed into the fugacity $f$. Note also that the partition function has completely been eliminated from the acceptance criteria.

### 7.6.2 Deleting a Molecule Inserted Randomly

The probability $\alpha_{mn}$ of choosing a molecule to delete is

$$\alpha_{mn} = \frac{1}{N} \tag{7.61}$$

The probability $\alpha_{nm}$ of inserting that molecule back in is

$$\alpha_{nm} = \frac{e^{-\beta U(\mathbf{q})}}{Z(1,V,T)}\, d\mathbf{q} \tag{7.62}$$

The acceptance probability for deleting a molecule inserted randomly is

$$\ln\left(\frac{\alpha_{mn}}{\alpha_{nm}}\frac{p_m}{p_n}\right) = \beta\left[\Delta U + U(\mathbf{q}_N)\right] + \beta\mu' + \ln\left(\frac{V}{N\Lambda^3}\right) \tag{7.63}$$

$$= \beta\left[\Delta U + U(\mathbf{q}_N)\right] + \ln\left(\frac{\beta f V}{N}\right) \tag{7.64}$$

Note that in $\Delta U$ is defined differently in Eqs. (7.59) and (7.60) than in Eqs. (7.63) and (7.64). In the former, the new configuration has more molecules, $\Delta U = U(\mathbf{q}_n^{N+1}) - U(\mathbf{q}_m^N)$. In the latter, the new configuration has fewer molecules, $\Delta U = U(\mathbf{q}_n^{N-1}) - U(\mathbf{q}_m^N)$.

# Bibliography


[1] Christopher J. Fennell and J. Daniel Gezelter. Is the ewald summation still necessary? pairwise alternatives to the accepted standard for long-range electrostatics. *The Journal of Chemical Physics*, 124(23), 2006.

[2] J. K. Shah and E. J. Maginn. A General And Efficient Monte Carlo Method for Sampling Intramolecular Degrees of Freedom of Branched and Cyclic Molecules. *J. Chem. Phys.*, 135:134121, 2011.

[3] D. Fincham. Optimization of the Ewald Sum For Large Systems. *Mol. Sim*, 13:1–9, 1994.

[4] P. L'Ecuyer. Tables of Maximally Equidistributed Combined LFSR Generators. *Math. Comput.*, 68:261–269, 1999.

[5] S. L. Lawton D. H. Olson, G. T. Kokotailo and W. M. Meier. Crystal Structure and Structure-Related Properties of ZSM-5. *J. Phys. Chem.*, 85:2238–2243, 1981.

[6] M.D. Macedonia and E.J. Maginn. A biased grand canonical Monte Carlo method for simulating adsorption using all-atom and branched united atom models. *Molecular Physics*, 96(9):1375–1390, 1999.