Faunus

May 19, 2022

Contents

CHAPTER 1

Introduction

1.1 About

Faunus is a general Monte Carlo simulation code, designed to be flexible, easy to use, and to modify. The code is written in C++ and Python bindings are available. The development is a team effort with, in reverse chronological order, many valiant contributions from:

Marco Polimeni, Vidar Aspelin, Stefan Hervø-Hansen, Richard Chudoba, Niels Kouwenhoven, Coralie Pasquier, Lukáš Sukeník, Giulio Tesei, Alexei Abrikossov, João Henriques, Björn Stenqvist, Axel Thuresson, Robert Vácha, Magnus Ullner, Chris Evers, Anıl Kurut, André Teixeira, Christophe Labbez, Ondrej Marsalek, Martin Trulsson, Björn Persson, Mikael Lund

Should you find Faunus useful, please consider supporting us by crediting:

- Stenqvist *et al. Molecular Simulation 2013, 39:1233* [citing articles].
- Lund, M. *et al. Source Code Biol. Med., 2008, 3:1* [citing articles].

1.2 Quick Start

Simulations are set up using YAML or JSON files, see for example minimal.yml, for a Metropolis Monte Carlo simulation of charged Lennard-Jones particles in a cubic PBC box. Running with

```
yason.py minimal.yml | faunus
```

produces an output file, `out.json`, with move statistics, system properties etc. The script `yason.py` merely converts from YAML to JSON as the former, easier to read, is used in all examples. For more examples, see the `examples` folder.

1.3 Getting Help

To open the user-guide in a browser, type:

```
faunus-manual
```

If you have questions, comments, or need help, create a ticket on our Github issue page.

CHAPTER 2

Installing

2.1 Using Conda

For macOS and Linux x86-64, precompiled binary packages are available via (mini)conda:

```
conda config --add channels conda-forge
conda install faunus
```

In addition to the `faunus` executable, this installs a set of examples in `share/faunus`, as well as python bindings. To *update* an existing installation, use

```
faunus --version      # show version string
conda search faunus   # show (new) revisions
conda upgrade faunus
```

For the adventurous, sporadically updated development versions can be installed with `conda install -c teokem faunus`. Starting from version 2.1, we adhere to semantic versioning.

2.2 Building from source code

Faunus is continuously tested on macOS/Linux, but should compile on most unix operating systems and possibly under Cygwin (Windows).

2.2.1 Requirements

- CMake 3.11+
- C/C++17 compiler (Clang 5+, GCC 7+, etc.)
- Python 3.6+ with the following packages:
  - `jinja2`, `ruamel_yaml` or `yaml`

The following are optional:

- `jsonschema` (for validating input)
- `pandoc` (for building manual)
- `pypandoc` (for building manual)
- `BeautifulSoup4` (for building manual)
- Message Passing Interface (MPI)

macOS tip: Apple's developer tools, Xcode, include clang and CMake can be installed with an Installer package from Kitware, or using MacPorts, Homebrew, or (mini)conda

2.2.2 Compiling

Download the latest release or the developer branch and build using cmake:

```
cd faunus
cmake . [OPTIONS]
make faunus
make usagetips # requires `pandoc`, `pypandoc`, `BeautifulSoup4`
```

Use `make help` to see all build targets.

The following options are available:

2.2.3 Compiling the Manual

Pandoc is required to build the HTML manual:

```
make manual_html
```

In addition to pandoc, a TeX Live installation containing XeLaTeX is required to build the PDF manual. The manual is supposed to be typeset with EB Garamond, Garamond Math and Fira Code fonts thus they have to be available in your system. Alternatively, you can tweak the font options in the `header.md` file.

```
make manual
```

2.2.4 Python libraries in odd locations

Should there be multiple compilers or python distributions, be specific:

```
CC=/opt/bin/clang CXX=/opt/bin/clang++ cmake . \
  -DPYTHON_EXECUTABLE=/opt/bin/python3 \
  -DPYTHON_INCLUDE_DIR=/opt/include/python3.6 \
  -DPYTHON_LIBRARY=/opt/lib/libpython3.6.dylib
```

For solving python issues on macOS, the linked python library can be probed and, if needed, renamed:

```
otool -L pyfaunus.so
install_name_tool -change libpython3.6.dylib \
  $HOME/miniconda/lib/libpython3.6.dylib pyfaunus.so
```

2.2.5 Resetting the build system

To change the compiler or for another reason reset the build system, do:

```
make clean
rm -fR CMakeCache.txt CMakeFiles _deps
```

CHAPTER 3

Development

3.1 Code Style

If you plan to contribute to Faunus it is recommended to activate the pre-commit hook for automatic styling of all changes:

```
cd faunus
./scripts/git-pre-commit-format install
```

This requires `clang-format` which may also be directly used in IDE's such as CLion. In the top-level directory of Faunus you will find the style configuration file `.clang-format`

Also, adhere to the following naming conventions:

3.2 Creating a conda package (development usage)

The basic steps for creating a conda package is outlined below, albeit details depend on the build environment. See also the `.travis.yml` configuration file in the main repository.

```
conda config --add channels conda-forge
conda install conda-build anaconda-client
cd scripts/
conda-build .
anaconda login
anaconda upload -u USER ... # see output from build step
```

Instead of uploading to anaconda.org, install a local copy directly after the build step above:

```
conda install -c USER faunus --use-local
```

CHAPTER 4

Running Simulations

The main program for running simulations is `faunus` and should be available from the command line after installation. For general usage, type:

```
faunus --help
```

Input is read either from `stdin` or from a JSON formatted file. Some examples:

```
faunus < input.json              # input from stdin
faunus -i in.json -o out.json -q # file input/output and be quiet
```

Via the script `yason.py`, see below, YAML formatted input can be passed:

```
yason.py in.yml | faunus # from yaml
```

4.1 Input and Output

Natively, input and output are JSON formatted:

```
{ "atomlist": [
        { "Na+": { "q": 1.0, "mw": 22.99 } }
    ]
}
```

However, via the helper script `yason.py`, JSON can be converted to/from YAML which is less verbose, more readable and therefore used throughout the documentation:

```
atomlist:
    - Na+: { q: 1.0, mw: 22.99 }
```

Generating several input files for a parameter scan, it can be helpful to use an input *template file*,

```
# template.yml
geometry: {type: cylinder, length: {{length}}, radius: {{radius}}}
```

which can be populated using python:

```
from jinja2 import Template
with open('template.yml') as f:
    output = Template(f.read()).render(
             length = 200,
             radius = 50 )
    print(output)
```

4.1.1 Post-Processing

JSON formatted output can conveniently be converted to syntax highlighted YAML for better readability:

```
yason.py --color out.json
```

For further processing of output or input, JSON (and YAML) can be read by most programming languages. For example in python:

```
import json
with open('out.json') as f:
    d = json.load(f) # --> dict
    print( d['atomlist'][0]["Na+"]["mw"] ) # --> 22.99
    #              ^      ^    ^     ^
    #              |      |    |     |
    #              |      |    |     get mol. weight value
    #              |      |    key is the atom name
    #              |      fist object in array
    #              atomlist is an array of objects
```

4.2 Restarting

Restart files generated by the analysis function `savestate` contains the last system state (positions, groups etc.). To start from the previously saved state, use:

```
faunus --input in.json --state state.json
```

4.3 Diagnostics

Faunus writes various status and diagnostic messages to the standard error output. The amount of messages can be control with the `--verbosity` (`-v`) option ranging from completely suppressed messages to tracing all operations. Only warnings and errors are shown by default. It may be useful to increase the verbosity level when debugging to show status and debug information.

```
faunus --verbosity 5 --input in.json
```

Note this is an experimental feature, covering only a fraction of actions so far.

Tip: Redirect the standard error output to a log file.

```
faunus -v 5 -i in.json 2>> error.log
```

4.4 Parallelization

By default, Monte Carlo moves and energy evaluations run in *serial* and are not sped up by OpenMP/MPI as described below. Pragmas for non-bonded interactions can relatively easily be added, but this currently requires source modifications. We are working to make this user-controllable and in the meantime consider using an embarrassingly parallel scheme via different random seeds (provided that your system equilibrates quickly).

4.4.1 OpenMP

Some routines in Faunus can run in parallel using multiple threads. The only prerequisite is that Faunus was compiled with OpenMP support (which is default). The number of threads is controlled with an environment variable. The following example demonstrates how to run Faunus using 4 threads:

```
export OMP_NUM_THREADS=4
faunus -i in.json
```

4.4.2 Message Passing Interface (MPI)

Only few routines in Faunus are currently parallelisable using MPI, for example parallel tempering, and penalty function energies.

Running with MPI spawns `nproc` processes that may or may not communicate with each other. If `nproc>1`, input and output files are prefixed with `mpi{rank}.` where `{rank}` is the rank or process number, starting from zero.

The following starts two processes, reading input from `mpi0.in.json` and `mpi1.in.json`. All output files, including those from any analysis are prefixed with `mpi0.` and `mpi1.`.

```
mpirun -np 2 ./faunus -i in.json
```

If all processes take the same input:

```
mpirun -np 2 ./faunus --nopfx --input in.json
mpirun -np 2 --stdin all ./faunus < in.json
```

4.5 Python Interface

An increasing part of the C++ API is exposed to Python. For instance:

```
import pyfaunus
help(pyfaunus)
```

For more examples, see `pythontest.py`. Note that the interface is under development and subject to change.

CHAPTER 5

Topology

The topology describes atomic and molecular properties as well as processes and reactions.

5.1 Global Properties

The following keywords control temperature, simulation box size etc., and must be placed outer-most in the input file.

```
temperature: 298.15  # system temperature (K)
geometry:
  type: cuboid       # Cuboidal simulation container
  length: [40,40,40] # cuboid dimensions (array or number)
mcloop:              # number of MC steps (macro × micro)
  macro: 5           # Number of outer MC steps
  micro: 100         # Number of inner MC steps; total = 5 × 100 = 500
random:              # seed for pseudo random number generator
  seed: fixed        # "fixed" (default) or "hardware" (non-deterministic)
```

5.1.1 Geometry

Below is a list of possible geometries, specified by `type`, for the simulation container, indicating if and in which directions periodic boundary conditions (PBC) are applied. Origin ($0,0,0$) is always placed in the geometric *center* of the simulation container. Particles are always kept inside the simulation container with an external potential that is zero if inside; infinity if outside.

5.1.2 Simulation Steps

The variables `macro` and `micro` are positive integers and their product defines the total number simulations steps. In each step a random Monte Carlo move is drawn from a weighted distribution. For each `macro` step, all analysis methods are, if befitting, instructed to flush buffered data to disk and may also trigger terminal output. For this reason `macro` is typically set lower than `micro`.

5.2 Atom Properties

Atoms are the smallest possible particle entities with properties defined below.

A filename (`.json`) may be given instead of an atom definition to load from an external atom list. Atoms are loaded in the given order, and if it occurs more than once, the latest entry is used.

Example:

```
atomlist:
  - Na: {q:  1.0, sigma: 4, eps: 0.05, dp: 0.4}
  - Ow: {q: -0.8476, eps: 0.65, sigma: 3.165, mw: 16}
  - my-external-atomlist.json
  - ...
```

5.3 Molecule Properties

A molecule is a collection of atoms, but need not be associated as real molecules. Two particular modes can be specified:

1. If `atomic=true` the atoms in the molecule are unassociated and is typically used to define salt particles or other non-aggregated species. No structure is required, and the molecular center of mass (COM) is unspecified.
2. If `atomic=false` the molecule resembles a real molecule and a structure or trajectory is *required*.

Properties of molecules and their default values:

Example:

```
moleculelist:
  - salt: {atoms: [Na,Cl], atomic: true}
  - water:
      structure: water.xyz
      bondlist:
        - harmonic: {index: [0,1], k: 100, req: 1.5}
        - ...
  - carbon_dioxide:
      structure:
        - O: [-1.162,0,0]
        - C: [0,0,0]
        - O: [1.162,0,0]
      bondlist:
        - harmonic: {index: [1,0], k: 8443, req: 1.162}
        - harmonic: {index: [1,2], k: 8443, req: 1.162}
        - harmonic_torsion: {index: [0,1,2], k: 451.9, aeq: 180}
      excluded_neighbours: 2 # generates an exclusionlist as shown below
      exclusionlist: [ [0,1], [1,2], [0,2] ] # redundant in this topology
  - ...
```

5.3.1 Structure Loading Policies

When giving structures using the `structure` keyword, the following policies apply:

- `structure` can be a file name: `file.@` where `@=xyz|pqr|aam`
- `structure` can be an *array* of atom names and their positions: `- Mg:  [2.0,0.1,2.0]`

- `structure` can be a FASTA sequence: `{fasta: [AAAAAAAK], k: 2.0; req: 7.0}` which generates a linear chain of harmonically connected atoms. FASTA letters are translated into three letter residue names which *must* be defined in `atomlist`. Special letters: `n=NTR`, `c=CTR`, `a=ANK`.
- Radii in files are *ignored*; `atomlist` definitions are used.
- By default, charges in files are *used*; `atomlist` definitions are ignored. Use `keepcharges=False` to override.
- A warning is issued if radii/charges differ in files and `atomlist`.
- Box dimensions in files are ignored.

5.3.2 Nonbonded Interaction Exclusion

Some nonbonded interactions between atoms within a molecule may be excluded in the topology. Force fields almost always exclude nonbonded interactions between directly bonded atoms. However other nonbonded interactions may be excluded as well; refer to your force field parametrization. If a molecule contains overlapping hard spheres, e.g., if the bond length is shorter than the spheres diameter, it is necessary to exclude corresponding nonbonded interactions to avoid infinite energies.

The excluded nonbonded interactions can be given as an explicit list of atom pairs `excludelist`, or they can be deduced from the molecules topology using the `excluded_neighbours=n` option: If the atoms are `n` or less bonds apart from each other in the molecule, the nonbonded interactions between them are excluded. Both options `excluded_neighbours` and `exclusionlist` can be used together making a union.

5.4 Initial Configuration

Upon starting a simulation, an initial configuration is required and must be specified in the section `insertmolecules` as a list of valid molecule names. Molecules are inserted in the given order and may be `inactive`. If a group is marked `atomic`, its `atoms` are inserted `N` times.

Example:

```
insertmolecules:
  - salt:  { molarity: 0.1 }
  - water: { N: 256 }
  - water: { N: 1, inactive: true }
```

The following keywords for each molecule type are available:

A filename with positions for the `N` molecules can be given with `positions`. The file must contain exactly `N`-times molecular positions that must all fit within the simulation box. Only *positions* from the file are copied; all other information is ignored.

For `implicit` molecules, only `N` should be given and the molecules are never inserted into the simulation box.

The `molarity` keyword is an alternative to `N` and uses the initial volume to calculate the number of molecules to insert. `N` and `molarity` are mutually exclusive.

5.4.1 Overlap Check

Random insertion is repeated until there is no overlap with the simulation container boundaries. Overlap between particles is ignored and for i.e. hard-sphere potentials the initial energy may be infinite.

5.5 Equilibrium Reactions

Faunus supports density fluctuations, coupled to chemical equilibria with explicit and/or implicit particles via their chemical potentials as defined in the `reactionlist` detailed below, as well as in `atomlist` and `moleculelist`. The level of flexibility is very high and reactions can be freely composed.

The move involves deletion and insertion of reactants and products and it is therefore important that simulations are started with a sufficiently high number of initial molecules in `insertmolecules`. If not, the `rcmc` move will attempt to issue warnings with suggestions how to fix it.

5.5.1 Reaction format

The initial key describes a transformation of reactants (left of =) into products (right of =) that may be a mix of atomic and molecular species.

- all species, +, and = must be surrounded by white-space
- atom and molecule names cannot overlap
- species can be repeated to match the desired stoichiometry, e.g. `A + A = C`

Available keywords:

The `neutral` keyword is needed for molecular groups containing titratable atoms. If `neutral` is set to true, the activity of the neutral molecule should be specified in `moleculelist`.

5.5.2 Example: Grand Canonical Salt Particles

This illustrates how to maintain constant chemical potential of salt ions:

```
atomlist:
  - na: {q: 1.0, ...}  # note that atom names must differ
  - cl: {q: -1.0, ...} # from molecule names
moleculelist:
  - Na+: {atoms: [na], atomic: true, activity: 0.1}
  - Cl-: {atoms: [cl], atomic: true, activity: 0.1}
reactionlist:
  - = Na+ + Cl+: {} # note: molecules, not atoms
moves:
  - rcmc: {} # activate speciation move
```

The same setup can be used also for molecular molecules, *i.e.* molecules with `atomic: false`.

5.5.3 Example: Acid/base titration with *implicit* protons

An *implicit* atomic reactant or product is included in the reaction but not explicitly in the simulation cell. Common use-cases are acid-base equilibria where the proton concentration is often very low:

```
atomlist:
  - H+: {implicit: true, activity: 0.00001} # pH 5
  - COO-: {q: -1.0, ...}
  - COOH: {q: 0.0, ...}
reactionlist:
  - "COOH = COO- + H+": {pK: 4.8} # not electroneutral!
```

where we set `pK` equal to the `pKa`: $$ K_a = \frac{ a_{\mathrm{COO^-}} a_{\mathrm{H^+}} }{ a_{\mathrm{COOH}} }. $$ To simulate at a given constant pH, H+ is specified as an implicit atom of activity $10^{-\mathrm{pH}}$ and the equilibrium is modified accordingly (in this case K is divided by $a_{\mathrm{H^+}}$). It is important to note that this reaction violates *electroneutrality* and should be used only with Hamiltonians where this is allowed. This could for example be in systems with salt screened Yukawa interactions.

5.5.4 Example: Acid/base titration coupled with Grand Canonical Salt

To respect electroneutrality when swapping species, we can associate the titration move with an artificial insertion or deletion of salt ions. These ions should be present under constant chemical potential and we therefore couple to a grand canonical salt bath:

```
atomlist:
  - H+: {implicit: true, activity: 0.00001} # pH 5
  - COO-: {q: -1.0, ...}
  - COOH: {q: 0.0, ...}
  - na: {q: 1.0, ...}
  - cl: {q: -1.0, ...}
moleculelist:
  - Na+: {atoms: [na], atomic: true, activity: 0.1}
  - Cl-: {atoms: [cl], atomic: true, activity: 0.1}
reactionlist:
  - COOH + Cl- = COO- + H+: {pK: 4.8} # electroneutral!
  - COOH = Na+ + COO- + H+: {pK: 4.8} # electroneutral!
  - = Na+ + Cl-: {} # grand canonical salt
```

For the first reaction, K is divided by both $a_{\mathrm{H^+}}$ and $a_{\mathrm{Cl^-}}$, so that the final equilibrium constant used by the speciation move is $$ K' = \frac{K_a}{a_{ \mathrm{H^+} } a_{ \mathrm{Cl^-} } } = \frac{ a_{\mathrm{COO^-}} }{ a_{\mathrm{COOH}} a_{\mathrm{Cl^-}} }. $$ In an ideal system, the involvement of Na or Cl in the acid-base reaction is inconsequential for the equilibrium, since the Grand Canonical ensemble ensures constant salt activity.

5.5.5 Example: Precipitation of Calcium Hydroxide using *implicit* molecules

Here we introduce a solid phase of Ca(OH)2 and its solubility product to predict the amount of dissolved calcium and hydroxide ions. Note that we start from an empty simulation box (both ions are inactive) and the solid phase is treated *implicitly*, i.e. it never inters the simulation box. Additional coupled reactions can naturally be introduced in order to study complex equilibrium systems under influence of intermolecular interactions.

```
moleculelist:
    - Ca(OH)2: {implicit: true} # this molecule is implicit
    - Ca++: {atoms: [ca++], atomic: true}
    - OH-: {atoms: [oh-], atomic: true}
insertmolecules:
    - Ca++: {N: 200, inactive: true}
    - OH-: {N: 400, inactive: true}
    - Ca(OH)2: {N: 200} # not actually inserted!
reactionlist:
    - "Ca(OH)2 = Ca++ + OH- + OH-": {pK: 5.19}
```

5.5.6 Example: Swapping between molecular conformations

The following can be used to alternate between different molecular conformations

```
moleculelist:
  - A: {atomic: false, structure: ...}
  - B: {atomic: false, structure: ...}
reactionlist:
  - A = B: {lnK: 0.69} # K=2, "B" twice as likely as "A"
```

CHAPTER 6

Energy

The system energy, or Hamiltonian, consists of a sum of potential energy terms,

$$ \mathcal{H}_{sys} = U_1 + U_2 + \dots $$

The energy terms are specified in `energy` at the top level input and evaluated in the order given. For example:

```
energy:
    - isobaric: {P/atm: 1}
    - sasa: {molarity: 0.2, radius: 1.4 }
    - confine: {type: sphere, radius: 10, molecules: [water]}
    - nonbonded:
        default: # applied to all atoms
        - lennardjones: {mixing: LB}
        - coulomb: {type: plain, epsr: 1}
        Na CH:   # overwrite specific atom pairs
        - wca: { mixing: LB }

    - maxenergy: 100
    - ...
```

The keyword `maxenergy` can be used to skip further energy evaluation if a term returns a large energy change (in kT), which will likely lead to rejection. The default value is *infinity*.

Energies in MC may contain implicit degrees of freedom, *i.e.* be temperature-dependent, effective potentials. This is inconsequential for sampling density of states, but care should be taken when interpreting derived functions such as energies, entropies, pressure etc.

6.1 Infinite and NaN Energies

In case one or more potential energy terms of the system Hamiltonian returns infinite or NaN energies, a set of conditions exists to evaluate the acceptance of the proposed move:

- always reject if new energy is NaN (i.e. division by zero)

- always accept if energy change is from NaN to finite energy
- always accept if the energy *difference* is NaN (i.e. from infinity to minus infinity)

These conditions should be carefully considered if equilibrating a system far from equilibrium.

6.2 External Pressure

This adds the following pressure term (see i.e. Frenkel and Smith, Chapter 5.4) to the Hamiltonian, appropriate for MC moves in $\ln V$:

$$ U = PV - k_BT\left (N + 1 \right) \ln V $$

where N is the total number of molecules and atomic species.

6.3 Nonbonded Interactions

This term loops over pairs of atoms, i, and j, summing a given pair-wise additive potential, u_{ij},

$$ U = \sum_{i=0}^{N-1}\sum_{j=i+1}^N u_{ij}(\textbf{r}_j-\textbf{r}_i)$$

Using `nonbonded`, potentials can be arbitrarily mixed and customized for specific particle combinations. `nonbonded_splined` internally *splines* the combined potential in an interval `[rmin,rmax]` determined by the following policies:

- `rmin` is decreased towards zero until the potential reaches `u_at_rmin=20` kT
- `rmax` is increased until the potential reaches `u_at_rmax=1e-6` kT

If outside the interval, infinity or zero is returned, respectively. Finally, the spline precision can be controlled with `utol=1e-5` kT.

Below is a description of possible nonbonded methods. For simple potentials, the hard coded variants are often the fastest option. For better performance, it is recommended to use `nonbonded_splined` in place of the more robust `nonbonded` method. To check that the combined potential is splined correctly, set `to_disk=true` to print to `A-B_tabulated.dat` the exact and splined combined potentials between species A and B.

6.3.1 Mass Center Cutoffs

For cutoff based pair-potentials working between large molecules, it can be efficient to use mass center cutoffs between molecular groups, thus skipping all pair-interactions. A single cutoff can be used between all molecules (`default`), or specified for specific combinations:

```
- nonbonded:
    cutoff_g2g:
      default: 40
      protein water: 60
```

6.4 Electrostatics

This is a multipurpose potential that handles several electrostatic methods. Beyond a spherical real-space cutoff, R_c, the potential is zero while if below,

$$ \tilde{u}^{(zz)}_{ij}(\bar{r}) = \frac{e^2 z_i z_j }{ 4\pi\epsilon_0\epsilon_r |\bar{r}| }\mathcal{S}(q) $$

where $\bar{r} = \bar{r}_j - \bar{r}_i$, and tilde indicate that a short-range function $\mathcal{S}(q=|\bar{r}|/R_c)$ is used to trucate the interactions. The available short-range functions are:

Internally $\mathcal{S}(q)$ is *splined* whereby all types evaluate at similar speed. For the `poisson` potential,

$$ \acute{q} = \frac{1-\exp\left(2\kappa R_c q\right)}{1-\exp\left(2\kappa R_c\right)} $$

which as the inverse Debye length, $\kappa\to 0$ gives $\acute{q}=q$. The `poisson` scheme can generate a number of other truncated pair-potentials found in the litterature, depending on `C` and `D`. Thus, for an infinite Debye length, the following holds:

6.4.1 Debye Screening Length

A background screening due to implicit ions can be added by specifying the keyword `debyelength` to the schemes

- `yukawa`
- `ewald`
- `poisson`

The former is an alias for `poisson` with `C=1`, and `D=-1` which gives a plain and shifted Coulomb potential with exponential screening. If `shift=false`, the potential is left unshifted and any given cutoff is ignored and instead set to infinity.

6.4.2 Multipoles

If `type=coulomb` is replaced with `type=multipole` then the electrostatic energy will in addition to monopole-monopole interactions include contributions from monopole-dipole, and dipole-dipole interactions. Multipolar properties of each particle is specified in the Topology. The `zahn` and `fennell` approaches have undefined dipolar self-energies and are therefore not recommended for such systems.

The ion-dipole interaction is described by

$$ \tilde{u}^{(z\mu)}_{ij}(\bar{r}) = -\frac{ez_i\left(\mu_j\cdot \hat{r}\right) }{|\bar{r}|^2} \left(\mathcal{S}(q) - q\mathcal{S}^{\prime}(q) \right) $$

where $\hat{r} = \bar{r}/|\bar{r}|$, and the dipole-dipole interaction by

$$ \tilde{u}^{\mu\mu}_{ij}(\bar{r}) = -\left (\frac{3 (\boldsymbol{\mu}_i \cdot \hat{r}) \left(\boldsymbol{\mu}_j\cdot\hat{r}\right) - \boldsymbol{\mu}_i\cdot\boldsymbol{\mu}_j }{|\bar{r}|^3}\right) \left(\mathcal{S}(q) - q\mathcal{S}^{\prime}(q) + \frac{q^2}{3}\mathcal{S}^{\prime\prime}(q) \right) - \frac{\left(\boldsymbol{\mu}_i\cdot\boldsymbol{\mu}_j\right)}{|\bar{r}|^3}\frac{q^2}{3}\mathcal{S}^{\prime\prime}(q). $$

Warning:

- The `zahn` and `fennell` approaches have undefined dipolar self-energies (see next section) and are therefore not recommended for dipolar systems.

6.4.3 Self-energies

When using `coulomb` or `multipole`, an electrostatic self-energy term is automatically added to the Hamiltonian. The monopole and dipole contributions are evaluated according to

$$ U_{self} = -\frac{1}{2}\sum_i^N\sum_{\ast\in{z,\mu}} \lim_{|\bar{r}_{ii}|\to 0}\left(u^{(\ast\ast)}_{ii}(\bar{r}_{ii})

- \tilde{u}^{(\ast\ast)}_{ii}(\bar{r}_{ii}) \right) $$

where no tilde indicates that $\mathcal{S}(q)\equiv 1$ for any q.

6.4.4 Ewald Summation

If type is `ewald`, terms from reciprocal space and surface energies are automatically added (in addition to the previously mentioned self- and real space-energy) to the Hamiltonian which activates the additional keywords:

The added energy terms are:

$$ U_{\text{reciprocal}} = \frac{2\pi f}{V} \sum_{ {\bf k} \ne {\bf 0}} A_k \vert Q^{q\mu} \vert^2 $$

$$ U_{\text{surface}} = \frac{1}{4\pi\varepsilon_0\varepsilon_r}\frac{ 2\pi }{ (2\varepsilon_{surf} + 1) V } \left(\left|\sum_{j}q_j\bar{r}_j\right|^2 + 2 \sum_j q_i \bar{r}_j \cdot \sum_j \boldsymbol{\mu}_j + \left| \sum_j \boldsymbol{\mu}_j \right|^2 \right) $$

where

$$ f = \frac{1}{4\pi\varepsilon_0\varepsilon_r} \quad\quad V=L_xL_yL_z $$

$$ A_k = \frac{e^{-(k^2 + \kappa^2)/4\alpha^2}}{k^2} \quad \quad Q^{q\mu} = Q^{q} + Q^{\mu} $$

$$ Q^{q} = \sum_{j}q_je^{i({\bf k}\cdot {\bf r}_j)} \quad Q^{\mu} = \sum_{j}i({\boldsymbol{\mu}}_j\cdot {\bf k}) e^{i({\bf k}\cdot {\bf r}_j)} $$

$$ \bar{k} = 2\pi\left(\frac{n_x}{L_x} , \frac{n_y}{L_y} ,\frac{n_z}{L_z} \right)\quad \bar{n} \in \mathbb{Z}^3 $$

Like many other electrostatic methods, the Ewald scheme also adds a self-energy term as described above. In the case of isotropic periodic boundaries (`ipbc=true`), the orientational degeneracy of the periodic unit cell is exploited to mimic an isotropic environment, reducing the number of wave-vectors to one fourth compared with 3D PBC Ewald. For point charges, IPBC introduce the modification,

$$ Q^q = \sum_j q_j \prod_{\alpha\in{x,y,z}} \cos \left(\frac{2\pi}{L_{\alpha}} n_{\alpha} r_{\alpha,j} \right) $$

while for point dipoles (currently unavailable),

$$ Q^{\mu} = \sum_j \bar{\mu}_j \cdot \nabla_j \left(\prod_{ \alpha \in { x,y,z } } \cos \left (\frac{2\pi}{L_{\alpha}} n_{\alpha} \bar{r}_{\alpha,j} \right) \right) $$

6.4.5 Mean-Field Correction

For cuboidal slit geometries, a correcting mean-field, external potential, $\varphi(z)$, from charges outside the box can be iteratively generated by averaging the charge density, $\rho(z)$, in dz-thick slices along z. This correction assumes that all charges interact with a plain Coulomb potential and that a cubic cutoff is used via the minimum image convention.

To enable the correction, use the `akesson` keyword at the top level of `energy`:

The density is updated every `nstep` energy calls, while the external potential can be updated slower (`nphi`) since it affects the ensemble. A reasonable value of `nstep` is system dependent and can be a rather large value. Updating the external potential on the fly leads to energy drifts that decrease for consecutive runs. Production runs should always be performed with `fixed=true` and a well converged $\rho(z)$.

At the end of simulation, `file` is overwritten unless `fixed=true`.

6.5 Pair Potentials

In addition to the Coulombic pair-potentials described above, a number of other pair-potentials can be used. Through the C++ API or the custom potential explained below, it is easy to add new potentials.

6.5.1 Charge-Nonpolar

The energy when the field from a point charge, z_i, induces a dipole in a polarizable particle of unit-less excess polarizability, $\alpha_j=\left (\frac{\epsilon_j-\epsilon_r}{\epsilon_j+2\epsilon_r}\right) a_j^3$, is

$$ \beta u_{ij} = -\frac{\lambda_B z_i^2 \alpha_j}{2r_{ij}^4} $$

where a_j is the radius of the non-polar particle and α_j is set in the atom topology, `alphax`. For non-polar particles in a polar medium, α_i is a negative number. For more information, see J. Israelachvili's book, Chapter 5.

Charge-polarizability products for each pair of species is evaluated once during construction and based on the defined atom types.

6.5.2 Cosine Attraction

An attractive potential used for coarse grained lipids and with the form,

$$ \beta u(r) = -\epsilon \cos^2 \left (\frac{\pi(r-r_c)}{2w_c} \right) $$

for $r_c\leq r \leq r_c+w_c$. For $r<r_c$, $\beta u=-\epsilon$, while zero for $r>r_c+w_c$.

6.5.3 Assorted Short Ranged Potentials

The potentials below are often used to keep particles apart and/or to introduce stickiness. The atomic interaction parameters, e.g., σ_i and ϵ_i, are taken from the topology.

If several potentials are used together and different values for the coefficients are desired, an aliasing of the parameters' names can be introduced. For example by specifying `sigma: sigma_hs`, the potential uses the atomic value `sigma_hs` instead of `sigma`, as shown in example below. To avoid possible conflicts of parameters' names with future keywords of Faunus, we recommend following naming scheme: `property_pot`, where `property` is either `sigma` or `eps` and `pot` stands for the potential abbreviation, i.e, `hs`, `hz`, `lj`, `sw`, and `wca`.

Mixing (combination) rules can be specified to automatically parametrize heterogeneous interactions. If not described otherwise, the same rule is applied to all atomic parameters used by the potential. No meaningful defaults are defined yet, hence always specify the mixing rule explicitly, e.g., `arithmetic` for `hardsphere`.

For convenience, the abbreviation `LB` can be used instead of `lorentz_berthelot`.

Custom parameter values can be specified to override the mixing rule for a given pair, as shown in the example bellow.

```
- lennardjones:
    mixing: LB
    custom:
      - Na Cl: {eps: 0.2, sigma: 2}
      - K Cl: { ... }
- hertz:
    mixing: LB
    eps: eps_hz
    custom:
```

(continues on next page)

(continued from previous page)

```
      - Na Cl: {eps_hz: 0.2, sigma: 2}
- hardsphere:
    mixing: arithmetic
    sigma: sigma_hs
    custom:
      - Na Cl: {sigma_hs: 2}
```

6.5.4 SASA (pair potential)

This calculates the surface area of two intersecting particles or radii R and r to estimate an energy based on transfer-free-energies (TFE) and surface tension. The total surface area is calculated as

$$ A = 4\pi \left (R^2 + r^2 \right) - 2\pi \left (Rh_1 + rh_2 \right) $$

where h_1 and h_2 are the heights of the spherical caps comprising the lens formed by the overlapping spheres. For complete overlap, or when far apart, the full area of the bigger sphere or the sum of both spheres are returned. The pair-energy is calculated as:

$$ u_{ij} = A \left (\gamma_{ij} + c_s \varepsilon_{\text{tfe},ij} \right) $$

where γ_{ij} and $\varepsilon_{\text{tfe},ij}$ are the arithmetic means of `tension` and `tfe` provided in the atomlist.

Note that SASA is strictly not additive and this pair-potential is merely a poor-mans way of approximately taking into account ion-specificity and hydrophobic/hydrophilic interactions. Faunus offers also a full, albeit yet experimental implementation of [Solvent Accessible Surface Area] energy.

6.5.5 Custom

This takes a user-defined expression and a list of constants to produce a runtime, custom pair-potential. While perhaps not as computationally efficient as hard-coded potentials, it is a convenient way to access alien potentials. Used in combination with `nonbonded_splined` there is no overhead since all potentials are splined.

The following illustrates how to define a Yukawa potential:

```
custom:
    function: lB * q1 * q2 / r * exp( -r/D ) # in kT
    constants:
        lB: 7.1  # Bjerrum length
        D: 30    # Debye length
```

The function is passed using the efficient ExprTk library and a rich set of mathematical functions and logic is available. In addition to user-defined constants, the following symbols are defined:

6.6 Custom External Potential

This applies a custom external potential to atoms or molecular mass centra using the ExprTk library syntax.

In addition to user-defined `constants`, the following symbols are available:

If `com=true`, charge refers to the molecular net-charge, and `x,y,z` the mass-center coordinates. The following illustrates how to confine molecules in a spherical shell of radius, *r*, and thickness *dr*:

```
customexternal:
    molecules: [water]
    com: true
    constants: {radius: 15, dr: 3}
    function: >
        var r2 := x^2 + y^2 + z^2;
        if ( r2 < radius^2 )
           1000 * ( radius-sqrt(r2) )^2;
        else if ( r2 > (radius+dr)^2 )
           1000 * ( radius+dr-sqrt(r2) )^2;
        else
           0;
```

6.6.1 Gouy Chapman

By setting `function=gouychapman`, an electric potential from a uniformly, charged plane in a 1:1 salt solution is added; see *e.g.* the book *Colloidal Domain* by Evans and Wennerström, 1999. If a surface potential, φ_0 is specified,

$$ \rho = \sqrt{\frac{2 c_0}{\pi \lambda_B} } \sinh (\beta e \varphi_0 / 2) $$ while if instead a surface charge density, ρ, is given, $$ \beta e \varphi_0 = 2\mbox{asinh} \left (\rho \sqrt{\frac{\pi \lambda_B} {2 c_0}} \right) $$ where λ_B is the Bjerrum length. With $\Gamma_0 = \tanh{ \beta e \varphi_0 / 4 }$ the final, non-linearized external potential is: $$ \beta e \phi_i = 2 \ln \left (\frac{1+\Gamma_0e^{-\kappa r_{z,i}}}{1-\Gamma_0 e^{-\kappa r_{z,i}}} \right) $$ where z_i is the particle charge; e is the electron unit charge; κ is the inverse Debye length; and $r_{z,i}$ is the distance from the charged xy-plane which is always placed at the minimum z-value of the simulation container (normally a slit geometry). Fluctuations of the simulation cell dimensions are respected.

The following parameters should be given under `constants`; the keywords `rho`, `rhoinv`, and `phi0` are mutually exclusive.

6.7 Bonded Interactions

Bonds and angular potentials are added via the keyword `bondlist` either directly in a molecule definition (topology) for intra-molecular bonds, or in `energy->bonded` where the latter can be used to add inter-molecular bonds:

```
moleculelist:
    - water: # TIP3P
        structure: "water.xyz"
        bondlist: # index relative to molecule
            - harmonic: { index: [0,1], k: 5024, req: 0.9572 }
            - harmonic: { index: [0,2], k: 5024, req: 0.9572 }
            - harmonic_torsion: { index: [1,0,2], k: 628, aeq: 104.52 }
energy:
    - bonded:
        bondlist: # absolute index; can be between molecules
           - harmonic: { index: [56,921], k: 10, req: 15 }
```

$\mu V T$ ensembles and Widom insertion are currently unsupported for molecules with bonds.

The following shows the possible bonded potential types:

6.7.1 Harmonic

$$ u(r) = \frac{1}{2}k(r-r_{\mathrm{eq}})^2 $$

6.7.2 Finite Extensible Nonlinear Elastic

Finite extensible nonlinear elastic potential long range repulsive potential.

$$ u(r) = \begin{cases} -\frac{1}{2} k r_{\mathrm{max}}^2 \ln \left [1-(r/r_{\mathrm{max}})^2 \right], & \text{if } r < r_{\mathrm{max}} \ \infty, & \text{if } r \geq r_{\mathrm{max}} \end{cases} $$

It is recommended to only use the potential if the initial configuration is near equilibrium, which prevalently depends on the value of `rmax`. Should one insist on conducting simulations far from equilibrium, a large displacement parameter is recommended to reach finite energies.

6.7.3 Finite Extensible Nonlinear Elastic + WCA

Finite extensible nonlinear elastic potential long range repulsive potential combined with the short ranged Weeks-Chandler-Andersen (wca) repulsive potential. This potential is particularly useful in combination with the `nonbonded_cached` energy.

$$ u(r) = \begin{cases} -\frac{1}{2} k r_{\mathrm{max}}^2 \ln \left [1-(r/r_{\mathrm{max}})^2 \right] + u_{\mathrm{wca}}, & \text{if } 0 < r \leq 2^{1/6}\sigma \ -\frac{1}{2} k r_{\mathrm{max}}^2 \ln \left [1-(r/r_{\mathrm{max}})^2 \right], & \text{if } 2^{1/6}\sigma < r < r_{\mathrm{max}} \ \infty, & \text{if } r \geq r_{\mathrm{max}} \end{cases} $$

It is recommended to only use this potential if the initial configuration is near equilibrium, which prevalently depends on the value of `rmax`. Should one insist on conducting simulations far from equilibrium, a large displacement parameter is recommended to reach finite energies.

6.7.4 Harmonic torsion

$$ u(r) = \frac{1}{2}k(\alpha - \alpha_{\mathrm{eq}})^2 $$

6.7.5 Cosine based torsion (GROMOS-96)

$$ u(r) = \frac{1}{2}k(\cos(\alpha) - \cos(\alpha_{\mathrm{eq}}))^2 $$

6.7.6 Proper periodic dihedral

$$ u(r) = k(1 + \cos(n\phi - \phi_{\mathrm{syn}})) $$

6.8 Geometrical Confinement

Confines `molecules` in a given region of the simulation container by applying a harmonic potential on exterior atom positions, $\mathbf{r}_i$:

$$ U = \frac{1}{2} k \sum_{i}^{\text{exterior}} f_i $$

where f_i is a function that depends on the confinement `type`, and k is a spring constant. The latter may be *infinite* which renders the exterior region strictly inaccessible. During equilibration it is advised to use a *finite* spring

constant to drive exterior particles inside the region. Should you insist on equilibrating with $k=\infty$, ensure that displacement parameters are large enough to transport molecules inside the allowed region, or all moves may be rejected. Further, some analysis routines have undefined behavior for configurations with infinite energies.

Available values for `type` and their additional keywords:

The `scale` option will ensure that the confining radius is scaled whenever the simulation volume is scaled. This could for example be during a virtual volume move (analysis) or a volume move in the NPT ensemble.

where $\mathbf{d}=(1,1,0)$ and $\circ$ is the entrywise (Hadamard) product.

where δr are distances to the confining, cuboidal faces. Note that the elements of `low` must be smaller than or equal to the corresponding elements of `high`.

6.9 Solvent Accessible Surface Area

Note that the implementation of Solvent Accessible Surface Area potential is considered *experimental*. The code is untested, unoptimized, and the configuration syntax below can change. The FreeSASA library option has to be enabled when [compiling].

Calculates the free energy contribution due to

1. atomic surface tension
2. co-solute concentration (typically electrolytes)

via a SASA calculation for each atom, as implemented in the FreeSASA library.

The energy term is:

$$ U = \sum_i^N A_{\text{sasa},i} \left (\gamma_i + c_s \varepsilon_{\text{tfe},i} \right) $$

where c_s is the molar concentration of the co-solute; γ_i is the atomic surface tension; and $\varepsilon_{\text{tfe},i}$ the atomic transfer free energy, both specified in the atom topology with `tension` and `tfe`, respectively.

6.10 Penalty Function

This is a version of the flat histogram or Wang-Landau sampling method where an automatically generated bias or penalty function, $f(\mathcal{X}^d)$, is applied to the system along a one dimensional ($d=1$) or two dimensional ($d=2$) reaction coordinate, $\mathcal{X}^d$, so that the configurational integral reads,

$$ Z(\mathcal{X}^d) = e^{-\beta f(\mathcal{X}^d)} \int e^{-\beta \mathcal{H}(\mathcal{R}, \mathcal{X}^d)} d \mathcal{R}. $$

where $\mathcal{R}$ denotes configurational space at a given $\mathcal{X}$. For every visit to a state along the coordinate, a small penalty energy, f_0, is added to $f(\mathcal{X}^d)$ until Z is equal for all $\mathcal{X}$. Thus, during simulation the free energy landscape is flattened, while the true free energy is simply the negative of the generated bias function,

$$ \beta A(\mathcal{X}^d) = -\beta f(\mathcal{X}^d) = -\ln\int e^{-\beta \mathcal{H}(\mathcal{R}, \mathcal{X}^d)} d \mathcal{R}. $$

Flat histogram methods are often attributed to Wang and Landau (2001) but the idea appears in earlier works, for example by Hunter and Reinhardt (1995) and Engkvist and Karlström (1996).

To reduce fluctuations, f_0 can be periodically reduced (`update`, `scale`) as f converges. At the end of simulation, the penalty function is saved to disk as an array ($d=1$) or matrix ($d=2$). Should the penalty function file be

available when starting a new simulation, it is automatically loaded and used as an initial guess. This can also be used to run simulations with a *constant bias* by setting $f_0=0$.

Example setup where the x and y positions of atom 0 are penalized to achieve uniform sampling:

```
energy:
- penalty:
    f0: 0.5
    scale: 0.9
    update: 1000
    file: penalty.dat
    coords:
    - atom: {index: 0, property: "x", range: [-2.0,2.0], resolution: 0.1}
    - atom: {index: 0, property: "y", range: [-2.0,2.0], resolution: 0.1}
```

Options:

The coordinate, $\mathcal{X}$, can be freely composed by one or two of the types listed in the next section (via `coords`).

6.10.1 Reaction Coordinates

The following reaction coordinates can be used for penalising the energy and can further be used when analysing the system (see Analysis). Please notice that atom id's are determined by the order of appearance in the `atomlist` whereas molecular id's follow the order of insertion specified in `insertmolecules`.

Atom Properties

Molecule Properties

Notes:

- the molecular dipole moment is defined with respect to the mass-center
- for `angle`, the principal axis is the eigenvector corresponding to the smallest eigenvalue of the gyration tensor
- `Rinner` can be used to calculate the inner radius of cylindrical or spherical vesicles. $d^2=\bf{r} \cdot$`dir` where $\bf{r}$ is the position vector
- `L/R` can be used to calculate the bending modulus of a cylindrical lipid vesicle
- `Rg` is calculated as the square-root of the sum of the eigenvalues of the gyration tensor, S. $$ S = \frac{1}{\sum_{i=1}^{N} m_{i}} \sum_{i=1}^{N} m_{i} \bf{t_i} \bf{t_i^T} $$ where $\bf{t_i} = \bf{r_i} - \bf{cm}$, $\bf{r_i}$ is the coordinate of the ith atom, m_i is the mass of the ith atom, $\bf{cm}$ is the mass center of the group and N is the number of atoms in the molecule.

System Properties

The enclosing cuboid is the smallest cuboid that can contain the geometry. For example, for a cylindrical simulation container, `Lz` is the height and `Lx=Ly` is the diameter.

6.10.2 Multiple Walkers with MPI

If compiled with MPI, the master process collects the bias function from all nodes upon penalty function `update`. The *average* is then re-distributed, offering linear parallelization of the free energy sampling. It is crucial that the

walk in coordinate space differs in the different processes, e.g., by specifying a different random number seed; start configuration; or displacement parameter. File output and input are prefixed with `mpi{rank}`.

The following starts all MPI processes with the same input file, and the MPI prefix is automatically appended to all other input and output:

```
yason.py input.yml | mpirun --np 6 --stdin all faunus -s state.json
```

Here, each process automatically looks for `mpi{nproc}.state.json`.

6.11 Constraining the system

Reaction coordinates can be used to constrain the system within a `range` using the `constrain` energy term. Stepping outside the range results in an inifinite energy, forcing rejection. For example,

```
energy:
    - constrain: {type: molecule, index: 0, property: end2end, range: [0,200]}
```

Tip: placing `constrain` at the *top* of the energy list is more efficient as the remaining energy terms are skipped should an infinite energy arise.

CHAPTER 7

Monte Carlo Moves

A simulation can have an arbitrary number of MC moves operating on molecules, atoms, the volume, or any other parameter affecting the system energy. Moves are specified in the `moves` section at the top level input. For example:

```
moves:
    - moltransrot: { molecule: water, dp: 2.0, repeat: N,
                     dprot: 1.0, dir: [1,1,0] }
    - volume: { dV: 0.01 }
    - ...

random: { seed: hardware }
```

The pseudo-random number engine used for MC moves can be seeded in three ways,

The last option is used to restore the state of the engine as saved along with normal simulation output as a string containing a lenghty list of numbers. If initialization from a previously saved state fails – this may happen if generated on another operating system – a warning is issued and the seed falls back to `fixed`.

7.1 Translation and Rotation

The following moves are for translation and rotation of atoms, molecules, or clusters. The `dir` keyword restricts translational directions which by default is set to `[1,1,1]`, meaning translation by a unit vector, randomly picked on a sphere, and scaled by a random number in the interval `[0, dp]`. If `dir=[1,1,0]` the unit vector is instead picked on a circle (here *x*, *y*) and if `dir=[0,0,1]` on a line (here z).

7.1.1 Molecular

This will simultaneously translate and rotate a molecular group by the following operation

$$ \textbf{r}^N_{trial} = \mbox{Rot}(\textbf{r}^N) + \delta $$

where $\mbox{Rot}$ rotates `dprot`$\cdot \left (\zeta-\frac{1}{2} \right)$ radians around a random unit vector emanating from the mass center, ζ is a random number in the interval $[0,1[$, and δ is a random unit vector

scaled by a random number in the interval `[0,dp]`. A predefined axis of rotation can be specified as `dirrot`. For example, setting `dirrot` to [1,0,0], [0,1,0] or [0,0,1] results in rotations about the $x-$, $y-$, and $z-$axis, respectively. Upon MC movement, the mean squared displacement will be tracked.

7.1.2 Atomic

As `moltransrot` but instead of operating on the molecular mass center, this translates and rotates individual atoms in the group. The repeat is set to the number of atoms in the specified group and the displacement parameters `dp` and `dprot` for the individual atoms are taken from the atom properties defined in the topology. Atomic *rotation* affects only anisotropic particles such as dipoles, spherocylinders, quadrupoles etc.

7.1.3 Cluster Move

This will attempt to rotate and translate clusters of molecular `molecules` defined by a distance `threshold` between mass centers. The `threshold` can be specified as a single number or as a complete list of combinations. For simulations where small molecules cluster around large macro-molecules, it can be useful to use the `satellites` keyword which denotes a list of molecules that can be part of a cluster, but cannot be the cluster nucleus or starting point. All molecules listed in `satellites` must be part of `molecules`. A predefined axis of rotation can be specified as `dirrot`. For example, setting `dirrot` to [1,0,0], [0,1,0] or [0,0,1] results in rotations about the $x-$, $y-$, or $z-$axis, respectively.

The move is associated with bias, such that the cluster size and composition remain unaltered. If a cluster is larger than half the simulation box length, only translation will be attempted.

Example:

```
cluster:
   molecules: [protein, cations]
   satellites: [cations]
   threshold:
      protein protein: 25
      protein cations: 15
      cations cations: 0
   dp: 3
   dprot: 1
```

7.2 Internal Degrees of Freedom

7.2.1 Charge Move

This performs a fractional charge move on a specific atom.

Limitations: This move changes the particle charge and therefore cannot be used with splined pair-potentials where the initial charges from are read from `atomlist`. Instead, use a hard-coded variant like `nonbonded_coulombl j` etc.

7.2.2 Conformational Swap

This will swap between different molecular conformations as defined in the Molecule Properties with `traj` and `trajweight` If defined, the weight distribution is respected, otherwise all conformations have equal intrinsic weight. Upon insertion, the new conformation is randomly oriented and placed on top of the mass-center of an exising molecule. That is, there is no mass center movement.

7.2.3 Pivot

Performs a rotation around a random, harmonic bond vector in `molecule`, moving all atoms either before *or* after the bond with equal probability. Current implementation assumes unbranched chains with all atoms as links, i.e., no side chains are present. For long polymers (compared to the box size), a large displacement parameter may cause problems with mass center calculation in periodic systems. This can be caught with the `sanity` analysis and should it occur, try one of the following:

- enable `skiplarge`
- decrease `dprot`
- increase the simulation container.

The first option will simply reject troublesome configurations and the final output contains information of the skipped fraction. Skipping is unphysical so make sure the skipped fraction is small.

The default value of `repeat` is the number of harmonic bonds in the `molecule` (multiplied by the number of molecules).

Limitations: Chain bonds have to be ordered sequentially in the topology.

7.2.4 Crankshaft

Performs a rotation of a chain segment between two randomly selected atoms in the `molecule`.

The default value of `repeat` is the number of atoms in the `molecule` minus two (multiplied by the number of molecules).

7.3 Parallel Tempering

We consider an extended ensemble, consisting of n sub-systems or replicas, each in a distinct thermodynamic state (different Hamiltonians) and with the total energy

$$ U = \sum_i^n\mathcal{H}_i(\mathcal{R}_i) $$

The parallel tempering move performs a swap move where coordinate spaces (positions, volume) between random, neighboring sub-systems, i and j, are exchanged,

$$ \mathcal{R}_i^{\prime} = \mathcal{R}_j \quad \text{and} \quad \mathcal{R}_j^{\prime} = \mathcal{R}_i $$

and the energy change of the *extended ensemble*, $\Delta U_{i\leftrightarrow j}$, is used in the Metropolis acceptance criteria.

Parallel tempering requires compilation with MPI and the number of replicas, n, exactly matches the number of processes. Each replica prefixes input and output files with `mpi0.`, `mpi1.`, etc. and only exchange between neighboring processes is performed.

Parallel tempering is currently limited to systems with constant number of particles, N.

7.4 Volume Move

Performs a random walk in logarithmic volume,

$$ V^{\prime} = e^{\ln V + \left (\zeta-\frac{1}{2} \right)\cdot dV } $$

and scales:

1. molecular mass centers
2. positions of free atoms (groups with `atomic=true`)

by $(V^{\prime}/V)^{1/3}$. This is typically used for the NPT ensemble, and for this an additional pressure term should be added to the Hamiltonian. In the case of `isochoric` scaling, the total volume is kept constant and `dV` refers to an area change and reported output statistics on *volume* should be regarded as *area*. The table below explains the scaling behavior in different geometries:

Warning: Untested for cylinders, slits.

7.5 Reactive Canonical Monte Carlo

The speciation move handles density fluctuations and particle transformations and is the main move for particle insertion, deletion, and swapping used in (semi)-grand canonical ensembles. A reaction from `reactionlist` is randomly picked from the topology and is either propagated forward or backward. In Faunus, the total number of atoms and molecules is constant, but these can be either *active* or *inactive*. Deleting a molecule simply deactivates it, while insertion *vice versa* activates an inactive molecule. Thus, it is important that the *capacity* or reservoir of particles (active plus inactive) is sufficiently large to allow for fluctuations. This is ensured using `insertmolecules` (see Topology). A runtime warning will be given, should you run low on particles. Besides deleting/inserting molecules (mono- or polyatomic), the speciation move performs reactions involving a single-atom ID transformation (*e.g.*, acid-base reactions). In this case, an particle of type A (part of a mono- or polyatomic molecule) is randomly picked from the system and all its properties, except its position, are replaced with those of an atom of type B. Such ID transormations can also involve the addition/deletion of molecules or *implicit* atoms. For a reaction $$ \sum_i \nu_i M_i = 0 $$ where M_i is the chemical symbol and ν_i is the stoichiometric coefficient of species i (positive for products and negative for reagents), the contribution of a speciation move to the energy change is $$ \beta \Delta U = - \sum_i \ln{ \left (\frac{ N_i! }{(N_i+\nu_i)!} V^{\nu_i} \right) } - \ln{ \prod_i a_i^{\nu_i} }, $$ where N_i is the number of particles of species i in the current state and a_i is the activity of species i.

For more information, see the Topology section and doi:10/fqcpg3.

CHAPTER 8

Analysis

Faunus can perform on-the-fly analysis during simulation by allowing for an arbitrary number of analysis functions to be added. The list of analysis is defined in the `analysis` section at the top level input:

```
analysis:
    - systemenergy: {file: energy.dat, nstep: 500, nskip: 2000}
    - xtcfile: {file: traj.xtc, nstep: 1000}
    - widom:  {molecule: water, ninsert: 20, nstep: 50}
    - molrdf: {name1: water, name2: water, nstep: 100,
               dr: 0.1, dim: 3, file: rdf.dat}
    - ...
```

All analysis methods support the `nstep` keyword that defines the interval between sampling points and the `nskip` keyword that defines the number of initial steps that are excluded from the analysis. In addition all analysis provide output statistics of number of sample points, and the relative run-time spent on the analysis.

8.1 Density

8.1.1 Bulk Density

This calculates the average density, $\langle N_i/V \rangle$ of molecules and atoms which may fluctuate in *e.g.* the isobaric ensemble or the Grand Canonical ensemble. For atomic groups, densities of individual atom types are reported. The analysis also files probability density distributions of atomic and polyatomic molecules as well as of atoms involved in id transformations, *e.g.*, acid-base equilibria. The filename format is `rho-@name.dat`.

8.1.2 Density Profile

Calculates the summed density of `atoms` in spherical, cylindrical or planar shells around `origo` which by default is the center of the simulation box:

$$ \rho(r) = \frac{\langle N(r) \rangle}{V(r)} $$

The sum of coefficients in `dir` determines the volume element normalisation:

This can be used to obtain charge profiles, measure excess pressure etc.

8.1.3 Density Slice

Calculates the density in cuboidal slices of thickness *dz* along the *z* axis. If an atom name is specified for the option `atomcom`, the *z*-position of each atom is calculated with respect to the center of mass of the atoms of the given type.

8.2 Structure

8.2.1 Atomic $g(r)$

Samples the pair correlation function between atom id's *i* and *j*,

$$ g_{ij}(r) = \frac{ N_{ij}(r) }{ \sum_{r=0}^{\infty} N_{ij}(r) } \cdot \frac{ \langle V \rangle }{ V(r) } $$

where $N_{ij}(r)$ is the number of observed pairs, accumulated over the entire ensemble, in the separation interval $[r, r+dr]$ and $V(r)$ is the corresponding volume element which depends on dimensionality, `dim`.

By specifying `slicedir`, the RDF is calculated only for atoms within a slice of given `thickness`. For example, with `slicedir=[0,0,1]` and `thickness=2`, the RDF is calculated for atoms with *z*-coordinates differing by less than 2 Å. This quasi-2D RDF in the *xy*-plane should be normalized with `dim=2`.

8.2.2 Molecular $g(r)$

Same as `atomrdf` but for molecular mass-centers.

8.2.3 Dipole-dipole Correlation

Sample the dipole-dipole angular correlation, $\langle \pmb{\hat{\mu}}(0)\cdot \pmb{\hat{\mu}}(r) \rangle$, between dipolar atoms and as a function of separation, *r*. In addition, the radial distribution function, $g(r)$ is sampled and saved to `{file}.gofr.dat`.

8.2.4 Structure Factor

The isotropically averaged static structure factor between N point scatterers is calculated using the Debye formula,

$$ S(q) = 1 + \frac{2}{N} \left \langle \sum_{i=1}^{N-1}\sum_{j=i+1}^N \frac{\sin(qr_{ij})}{qr_{ij}} \right \rangle $$

The selected `molecules` can be treated either as single point scatterers (`com=true`) or as a group of individual point scatterers of equal intensity, i.e., with a form factor of unity.

The computation of the structure factor is rather computationally intensive task, scaling quadratically with the number of particles and linearly with the number of scattering vector mesh points. If OpenMP is available, multiple threads may be utilized in parallel to speed it up the analysis.

The `explicit` scheme is recommended for cuboids with PBC and the calculation is performed by explicitly averaging the following equation over the 3+6+4 directions obtained by permuting the crystallographic index `[100]`, `[110]`, `[111]` to define the scattering vector $\mathbf{q} = 2\pi p/L(h,k,l)$ where $p=1,2,\dots,p_{max}$.

$$ S(q) = \frac{1}{N} \left < \left (\sum_i^N \sin(\mathbf{qr}_i) \right)^2 + \left (\sum_j^N \cos(\mathbf{qr}_j) \right)^2 \right > $$

The sampled q-interval is always $\left [2\pi/L,, 2\pi p_{max} \sqrt{3} / L \right]$, L being the box side length. Currently only cubic boxes are supported. For more information, see doi:10.1063/1.449987.

8.2.5 Atomic Inertia Eigenvalues

This calculates the inertia eigenvalues for all particles having a given id. The inertia tensor is defined as

$$ I = \sum_{i=1}^N m_i (| \bf{t_i} |^2 \mathrm{I} - \bf{t_i} \bf{t_i}^T) $$

where $\bf{t_i} = \bf{r_i} - \bf{cm}$, $\bf{r_i}$ is the coordinate of the ith particle, $\bf{cm}$ is the position of the mass center of the whole group of atoms, m_i is the molecular weight of the ith particle, $\bf{I}$ is the identity matrix and N is the number of atoms.

8.2.6 Inertia Tensor

This calculates the inertia eigenvalues and the principal axis for a range of atoms within a molecular group of given index. Atom coordinates are considered with respect to the mass center of the group. For protein complex, the analysis can be used to calculate the principal axes of the constituent monomers, all originating at the mass center of the complex. The inertia tensor is defined as

$$ I = \sum_{i=1}^N m_i (| \bf{t_i} |^2 \mathrm{I} - \bf{t_i} \bf{t_i}^T) $$

where $\bf{t_i} = \bf{r_i} - \bf{cm}$, $\bf{r_i}$ is the coordinate of the ith particle, $\bf{cm}$ is the position of the mass center of the whole group of atoms, m_i is the molecular weight of the ith particle, $\bf{I}$ is the identity matrix and N is the number of atoms.

8.2.7 Polymer Shape

This calculates the radius of gyration, end-to-end distance, and related fluctuations for all groups defined in `molecules`.

8.3 Charge Properties

8.3.1 Molecular Multipoles

Calculates average molecular multipolar moments and their fluctuations.

8.3.2 Multipole Moments

For a range of atoms within a molecular group of given index, this calculates the total charge and dipole moment, as well as the eigenvalues and the major axis of the quadrupole tensor. Atom coordinates are considered with respect to the mass center of the group. For a protein complex, the analysis can be used to calculate, e.g., the dipole vectors of the constituent monomers, all originating at the mass center of the complex. The quadrupole tensor is defined as

$$ Q = \frac{1}{2} \sum_{i=1}^N q_i (3 \bf{t_i} \bf{t_i}^T - | \bf{t_i} |^2 \mathrm{I}) $$

where $\bf{t_i} = \bf{r_i} - \bf{cm}$, $\bf{r_i}$ is the coordinate of the ith particle, $\bf{cm}$ is the position of the mass center of the whole group of atoms, q_i is the charge of the ith particle, $\bf{I}$ is the identity matrix and N is the number of atoms.

8.3.3 Electric Multipole Distribution

This will analyse the electrostatic energy between two groups as a function of their mass center separation. Sampling consists of the following:

1. The exact electrostatic energy is calculated by explicitly summing Coulomb interactions between charged particles
2. Each group - assumed to be a molecule - is translated into a multipole (monopole, dipole, quadrupole)
3. Multipolar interaction energies are calculated, summed, and tabulated together with the exact electrostatic interaction energy. Ideally (infinite number of terms) the multipoles should capture full electrostatics

The points 1-3 above will be done as a function of group-to-group mass center separation, R and moments on molecule a and b with charges q_i in position $\boldsymbol{r}_i$ with respect to the mass center are calculated according to:

$$ q_{a/b} = \sum_i q_i \quad \quad \boldsymbol{\mu}_{a/b} = \sum_i q_i\mathbf{r_i} $$

$$ \boldsymbol{Q}_{a/b} = \frac{1}{2} \sum_i q_i\mathbf{r_i} \mathbf{r_i}^T $$

And, omitting prefactors here, the energy between molecule a and b at R is:

$$ u_{\text{ion-ion}} = \frac{q_aq_b}{R} \quad \quad u_{\text{ion-dip}} = \frac{q_a \boldsymbol{\mu}_b \boldsymbol{R}}{R^3} + \dots $$

$$ u_{\text{dip-dip}} = \frac{\boldsymbol{\mu_a}\boldsymbol{\mu_b} }{ R^3 } - \frac{3 (\boldsymbol{\mu_a} \cdot \boldsymbol{R}) (\boldsymbol{\mu_b}\cdot\boldsymbol{R}) }{R^5} $$

$$ u_{\text{ion-quad}} = \frac{ q_a \boldsymbol{R}^T \boldsymbol{Q}_b \boldsymbol{R} }{R^5}-\frac{q_a \mbox{tr}(\boldsymbol{Q}_b) }{R^3}+ \dots $$

$$ u_{\text{total}} = u_{\text{ion-ion}} + u_{\text{ion-dip}} + u_{\text{dip-dip}} + u_{\text{ion-quad}} $$

$$ u_{\text{exact}} = \sum_i^a\sum_j^b \frac{q_iq_j}{ | \boldsymbol{r_i} - \boldsymbol{r_j} | } $$

During simulation, the above terms are thermally averaged over angles, co-solute degrees of freedom etc. Note also that the moments are defined with respect to the *mass* center, not *charge* center. While for globular macromolecules the difference between the two is often small, the latter is more appropriate and is planned for a future update.

The input keywords are:

8.3.4 Charge Fluctuations

For a given molecule, this calculates the average charge and standard deviation per atom, and the most probable species (atom name) averaged over all present molecules. A PQR file of a random molecule with average charges and most probable atomic species can be saved.

8.4 Reaction Coordinate

This saves a given reaction coordinate as a function of steps. The generated output `file` has three columns:

1. step number
2. the value of the reaction coordinate
3. the cummulative average of all preceding values.

Optional gzip compression can be enabled by suffixing the filename with `.gz`, thereby reducing the output file size significantly. The folowing example reports the mass center z coordinate of the first molecule every 100th steps:

```
- reactioncoordinate:
    {nstep: 100, file: cmz.dat.gz, type: molecule, index: 0, property: com_z}
```

In the next example, the angle between the principal molecular axis and the xy-plane is reported by diagonalising the gyration tensor to find the principal moments:

```
- reactioncoordinate:
    {nstep: 100, file: angle.dat.gz, type: molecule, index: 0, property: angle, dir:␣
↪[0,0,1]}
```

8.4.1 Processing

In the above examples we stored two properties as a function of steps. To join the two files and calculate the *average angle* as a function of the mass center coordinate, z, the following python code may be used:

```
import numpy as np
from scipy.stats import binned_statistic

def joinRC(filename1, filename2, bins):
    x = np.loadtxt(filename1, usecols=[1])
    y = np.loadtxt(filename2, usecols=[1])
    means, edges, bins = binned_statistic(x, y, 'mean', bins)
    return (edges[:-1] + edges[1:]) / 2, means

cmz, angle = joinRC('cmz.dat.gz', 'angle.dat.gz', 100)
np.diff(cmz) # --> cmz resolution; control w. `bins`
```

Note that Numpy automatically detects and decompresses `.gz` files. Further, the command line tools `zcat`, `zless` etc. are useful for handling compressed files.

8.5 System Sanity

It is wise to always assert that the simulation is internally sane. This analysis checks the following and aborts if insane:

- all particles are inside the simulation boundaries
- molecular mass centers are correct
- ...more to be added.

To envoke, use for example `- sanity: {nstep: 1}` by default, `nstep=-1`, meaning it will be run at the end of simulation, only. This is not a particularly time-consuming analysis and we recommend that it is enabled for all simulations.

8.6 System Energy

Calculates the energy contributions from all terms in the Hamiltonian and outputs to a file as a function of steps. If filename ends with `.csv`, a comma separated value file will be saved, otherwise a simple space separated file with a single hash commented header line. All units in k_BT.

8.7 Perturbations

8.7.1 Virtual Volume Move

Performs a virtual volume move by scaling the simulation volume to $V+\Delta V$ along with molecular mass centers and atomic positions. The excess pressure is evaluated as a Widom average:

$$ p^{ex} = \frac{k_BT}{\Delta V} \ln \left\langle e^{-\delta u / k_BT} \right\rangle_{NVT} $$

For more advanced applications of volume perturbations - pressure tensors, surface tension etc., see here.

8.7.2 Virtual Translate Move

Performs a virtual displacement, dL, of a single `molecule` in the direction `dir` and measure the force by perturbation,

$$ f = \frac {k_BT \ln \langle \exp{\left (-dU/k_BT \right)} \rangle_0 }{ dL } $$

8.7.3 Widom Insertion

This will insert a non-perturbing ghost molecule into the system and calculate a Widom average to measure the free energy of the insertion process, *i.e.* the excess chemical potential:

$$ \mu^{ex} = -k_BT \ln \left\langle e^{-\delta u/k_BT} \right\rangle_0 $$

where δu is the energy change of the perturbation and the average runs over the *unperturbed* ensemble. If the molecule has `atomic=true`, δu includes the internal energy of the inserted group. This is useful for example to calculate the excess activity coefficient of a neutral salt pair. Upon insertion, random positions and orientations are generated. For use with rod-like particles on surfaces, the `absz` keyword may be used to ensure orientations on only one half-sphere.

Exactly *one inactive* `molecule` must be added to the simulation using the `inactive` keyword when inserting the initial molecules in the topology.

8.8 Positions and Trajectories

8.8.1 Save State

Saves the current configuration or the system state to file. For grand canonical simulations, the PQR file format sets charges and radii of inactive particles to zero and positions them in one corner of the box.

If the suffix is `json` (text) or `ubj` (binary), a single state file that can be used to restart the simulation is saved with the following information:

- topology: atom, molecule, and reaction definitions
- particle and group properties incl. positions
- geometry
- state of random number generator (if `saverandom=true`)

8.8.2 Space Trajectory (experimental)

Save all particle and group information to a compressed, binary trajectory format. The following properties are saved:

- all particle properties (id, position, charge, dipole etc.)
- all group properties (id, size, capacity etc.)
- todo: geometry, energy

The file suffix must be either `.traj` (uncomressed) or `.ztraj` (compressed). For the latter, the file size is reduced by roughly a factor of two using zlib compression.

8.8.3 XTC trajectory

Generates a Gromacs XTC trajectory file with particle positions and box dimensions as a function of steps. Both *active* and *inactive* atoms are saved.

8.8.4 Charge-Radius trajectory

Most trajectory file formats do not support a fluctuating number of particles. For each `nstep`, this analysis files charge and radius information for all particles. Inactive particles are included with *zero* charge and radius.

Using a helper script for VMD (see `scripts/`) this information can be loaded to visualise flutuating charges and or number of particles. The script should be sourced from the VMD console after loading the trajectory, or invoked when launching VMD:

```
vmd confout.pqr traj.xtc -e scripts/vmd-qrtraj.tcl
```