AtomsMM

Release 0.1.0

Charlles R. A. Abreu

Apr 11, 2020

CONTENTS

1 Overview **1**
- 1.1 Installation . . . 1
- 1.2 Documentation . . . 1
- 1.3 Development . . . 1

2 Installation **3**

3 Usage **5**

4 Python API **7**
- 4.1 computers . . . 7
- 4.2 forces . . . 8
- 4.3 integrators . . . 12
- 4.4 propagators . . . 18
- 4.5 reporters . . . 35
- 4.6 systems . . . 39
- 4.7 utils . . . 43

5 Contributing **45**
- 5.1 Bug reports . . . 45
- 5.2 Documentation improvements . . . 45
- 5.3 Feature requests and feedback . . . 45
- 5.4 Development . . . 46

6 Authors **49**

7 Changelog **51**

8 Glossary **53**

9 Bibliography **55**

10 Indices and tables **57**

Bibliography **59**

Python Module Index **61**

Index **63**

CHAPTER ONE

OVERVIEW

AtomsMM is an OpenMM customization developed by the ATOMS group at UFRJ/Brazil.

- Free software: MIT license

1.1 Installation

```
git clone https://github.com/atoms-ufrj/atomsmm.git
cd atomsmm
python setup.py install
```

1.2 Documentation

https://atomsmm.readthedocs.io/

1.3 Development

To run the all tests run:

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append`
	`tox`

Other	`PYTEST_ADDOPTS=--cov-append tox`

CHAPTER TWO

INSTALLATION

At the command line:

```
git clone https://github.com/atoms-ufrj/atomsmm.git
cd atomsmm
python setup.py install
```

CHAPTER THREE

USAGE

To use AtomsMM in a project:

```
import atomsmm
```

CHAPTER

FOUR

PYTHON API

4.1 computers

class atomsmm.computers.**PressureComputer**(*system*, *topology*, *platform*, *properties={}*, *temperature=None*)

Bases: `simtk.openmm.openmm.Context`

An OpenMM Context extension aimed at computing properties of a system related to isotropic volume variations.

Parameters

- **system** (*openmm.System*) – The system…
- **topology** (*openmm.app.Topology*) – The topology…
- **platform** (*openmm.Platform*) – The platform…
- **properties** (*dict(), optional, default=dict()*) – The properties…
- **temperature** (*unit.Quantity, optional, default=None*) – The bath temperature used to compute pressures using the equipartition expectations of kinetic energies. It this is *None*, then the instantaneous kinetic energies will be employed.

get_atomic_pressure()

Returns the unconstrained atomic pressure of a system:

$$P = \frac{2K + W}{3V},$$

where W is the unconstrained atomic virial (see `get_atomic_virial()`), K is the total kinetic energy of all atoms, and V is the box volume. If keyword *temperature* was employed in the `PressureComputer` creation, then the instantaneous kinetic energy is replaced by its equipartition-theorem average $\langle K \rangle = 3N_\mathrm{atoms} k_B T/2$, where T is the heat-bath temperature, thus making P independent of the atomic velocities.

Warning: The resulting pressure should not be used to compute the thermodynamic pressure of a system with constraints. For this, one can use `get_molecular_pressure()` instead.

get_atomic_virial()

Returns the unconstrained atomic virial of the system.

Considering full scaling of atomic coordinates in a box volume change (i.e. without any distance constraints), the internal virial of the system is given by

$$W = -\sum_{i,j} r_{ij} E'(r_{ij}),$$

where $E'(r)$ is the derivative of the interaction potential as a function of the distance between two atoms. Such interaction includes van der Waals, Coulomb, and bond-stretching contributions. Angles and dihedrals are not considered because they are invariant to full atomic coordinate scaling.

> **Warning:** The resulting virial should not be used to compute the thermodynamic pressure of a system with constraints. For this, one can use `get_molecular_virial()` instead.

`get_bond_virial`()
Returns the bond-stretching contribution to the atomic virial.

`get_coulomb_virial`()
Returns the electrostatic (Coulomb) contribution to the atomic virial.

`get_dispersion_virial`()
Returns the dispersion (van der Waals) contribution to the atomic virial.

`get_molecular_pressure`(*forces*)
Returns the molecular pressure of a system:

$$P = \frac{2K_{\rm mol} + W_{\rm mol}}{3V},$$

where $W_{\rm mol}$ is the molecular virial of the system (see `get_molecular_virial()`), $K_{\rm mol}$ is the center-of-mass kinetic energy summed for all molecules, and V is the box volume. If keyword *temperature* is was employed in the `PressureComputer` creation, then the moleculer kinetic energy is replaced by its equipartition-theorem average $\langle K_{\rm mol} \rangle = 3N_{\rm mols} k_B T/2$, where T is the heat-bath temperature.

forces [vector<openmm.Vec3>] A vector whose length equals the number of particles in the System. The i-th element contains the force on the i-th particle.

`get_molecular_virial`(*forces*)
Returns the molecular virial of a system.

To compute the molecular virial, only the center-of-mass coordinates of the molecules are considered to scale in a box volume change, while the internal molecular structure keeps rigid. The molecular virial is computed from the nonbonded part of the atomic virial by using the formulation of Ref. [1]:

$$W_{\rm mol} = W - \sum_i (\mathbf{r}_i - \mathbf{r}_i^{\rm cm}) \cdot \mathbf{F}_i,$$

where $\mathbf{r}_i$ is the coordinate of atom i, $\mathbf{F}_i$ is the resultant pairwise force acting on it, and $\mathbf{r}_i^{\rm cm}$ is the center-of-mass coordinate of the molecule to which it belongs.

forces [vector<openmm.Vec3>] A vector whose length equals the number of particles in the System. The i-th element contains the force on the i-th particle.

4.2 forces

class `atomsmm.forces.`**`NonbondedExceptionsForce`**
Bases: `atomsmm.forces._AtomsMM_CustomBondForce`

A special class designed to compute only the exceptions of an OpenMM NonbondedForce object.

class `atomsmm.forces.`**`DampedSmoothedForce`**(*alpha, cutoff_distance, switch_distance, degree=1*)
Bases: `atomsmm.forces._AtomsMM_CustomNonbondedForce`

A damped-smoothed version of the Lennard-Jones/Coulomb potential.

$$
\begin{aligned}
&V(r) = \left\{4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] + \frac{q_1 q_2}{4\pi\epsilon_0}\frac{\mathrm{erfc}(r)}{r}\right\} S(r) \\
&\sigma = \frac{\sigma_1 + \sigma_2}{2} \\
&\epsilon = \sqrt{\epsilon_1 \epsilon_2} \\
&S(r) = [1 + \theta(r - r_\mathrm{switch}) u^3 (15u - 6u^2 - 10)] \\
&u = \frac{r^n - r_\mathrm{switch}^n}{r_\mathrm{cut}^n - r_\mathrm{switch}^n}
\end{aligned}
$$

Warning: Long-range dispersion correction is not employed.

In the equations above, $\theta(x)$ is the Heaviside step function. Note that the switching function employed here, with *u* being a quadratic function of *r*, is slightly different from the one normally used in OpenMM, in which *u* is a linear function of *r*.

Parameters

- **alpha** (*Number or unit.Quantity*) – The Coulomb damping parameter (in inverse distance unit).
- **cutoff_distance** (*Number or unit.Quantity*) – The distance at which the nonbonded interaction vanishes.
- **switch_distance** (*Number or unit.Quantity*) – The distance at which the switching function begins to smooth the approach of the nonbonded interaction towards zero.
- **degree** (*int, optional, default=1*) – The degree *n* in the definition of the switching variable *u* (see above).

class `atomsmm.forces.`**`NearNonbondedForce`**(*cutoff_distance*, *switch_distance*, *adjustment=None*, *subtract=False*, *actual_cutoff=None*)

Bases: `atomsmm.forces._AtomsMM_CustomNonbondedForce`, `atomsmm.forces.NearForce`

This is a smoothed version of the Lennard-Jones + Coulomb potential

$$V_\mathrm{LJC}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] + \frac{1}{4\pi\epsilon_0}\frac{q_1 q_2}{r}.$$

The smoothing is accomplished by application of a 5th-order spline function $S(u(r))$, which varies softly from 1 down to 0 along the range $r_\mathrm{switch} \leq r \leq r_\mathrm{cut}$. Such function is

$$S(u) = 1 + u^3(15u - 6u^2 - 10),$$

where

$$u(r) = \begin{cases} 0 & r < r_\mathrm{switch} \\ \dfrac{r - r_\mathrm{switch}}{r_\mathrm{cut} - r_\mathrm{switch}} & r_\mathrm{switch} \leq r \leq r_\mathrm{cut} \\ 1 & r > r_\mathrm{cut} \end{cases}.$$

Such type of smoothing is essential for application in multiple time-scale integration using the RESPA-2 scheme described in Refs. [2], [3], and [4].

Three distinc versions are available:

1. Applying the switch directly to the potential:

$$V(r) = S(u(r))V_{\rm LJC}(r).$$

2. Applying the switch to a shifted version of the potential:

$$V(r) = S(u(r))\left[V_{\rm LJC}(r) - V_{\rm LJC}(r_{\rm cut})\right]$$

3. Applying the switch to the force that results from the potential:

$$\begin{aligned}V(r) &= V^*_{\rm LJC}(r) - V^*_{\rm LJC}(r_{\rm cut})\\ V^*_{\rm LJC}(r) &= \left\{4\epsilon\left[f_{12}(u(r))\left(\frac{\sigma}{r}\right)^{12} - f_6(u(r))\left(\frac{\sigma}{r}\right)^{6}\right] + \frac{f_1(u(r))}{4\pi\epsilon_0}\frac{q_1q_2}{r}\right\}\end{aligned}$$

where $f_n(u)$ is the solution of the 1st order differential equation

$$\begin{aligned}&f_n - \frac{u+b}{n}\frac{df_n}{du} = S(u)\\ &f_n(0) = 1\\ &b = \frac{r_{\rm switch}}{r_{\rm cut} - r_{\rm switch}}\end{aligned}$$

As a consequence of this modification, $V'(r) = S(u(r))V'_{\rm LJC}(r)$.

Note: In all cases, the Lorentz-Berthelot mixing rule is applied for unlike-atom interactions.

Parameters

- **cutoff_distance** (*unit.Quantity*) – The distance at which the nonbonded interaction vanishes.
- **switch_distance** (*unit.Quantity*) – The distance at which the switching function begins to smooth the approach of the nonbonded interaction towards zero.
- **adjustment** (*str, optional, default=None*) – A keyword for modifying the potential energy function. If it is *None*, then the switching function is applied directly to the original potential. Other options are *'shift'* and *'force-switch'*. If it is *'shift'*, then the switching function is applied to a potential that is already null at the cutoff due to a previous shift. If it is *'force-switch'*, then the potential is modified so that the switching function is applied to the forces rather than the potential energy.
- **subtract** (*bool, optional, default=False*) – Whether to substract (rather than add) the force.
- **actual_cutoff** (*unit.Quantity, optional, default=None*) – The cutoff that will actually be used by OpenMM. This is often required for compatibility with other forces in the same force group. If it is *None*, then the passed *cutoff_distance* (see above) will be used.

class `atomsmm.forces.`**`NearExceptionForce`**(*cutoff_distance*, *switch_distance*, *adjustment=None*, *subtract=False*)

Bases: `atomsmm.forces._AtomsMM_CustomBondForce`, `atomsmm.forces.NearForce`

class `atomsmm.forces.`**`FarNonbondedForce`**(*preceding*, *cutoff_distance*, *switch_distance=None*)

Bases: `atomsmm.forces._AtomsMM_CompoundForce`

The complement of NearNonbondedForce and NonbondedExceptionsForce classes in order to form a complete OpenMM NonbondedForce.

Note: Except for the shifting, this model is the 'far' part of the RESPA2 scheme of Refs. [2] and [3], with the switching function applied to the potential rather than to the force.

Parameters

- **preceding** (*NearNonbondedForce*) – The NearNonbondedForce object with which this Force is supposed to match.
- **cutoff_distance** (*Number or unit.Quantity*) – The distance at which the nonbonded interaction vanishes.
- **switch_distance** (*Number or unit.Quantity, optional, default=None*) – The distance at which the switching function begins to smooth the approach of the nonbonded interaction towards zero. If this is None, then no switching will be done prior to the potential cutoff.
- **nonbondedMethod** (*openmm.NonbondedForce.Method, optional, default=PME*) – The method to use for nonbonded interactions. Allowed values are NoCutoff, CutoffNonPeriodic, CutoffPeriodic, Ewald, PME, or LJPME.
- **ewaldErrorTolerance** (*Number, optional, default=1E-5*) – The error tolerance for Ewald summation.

class `atomsmm.forces.`**`SoftcoreLennardJonesForce`**(*cutoff_distance=None*, *use_switching_function=None*, *switch_distance=None*, *use_dispersion_correction=None*, *parameter='lambda'*)

Bases: `atomsmm.forces._AtomsMM_CustomNonbondedForce`

A softened version of the Lennard-Jones potential.

$$
\begin{aligned}
V(r) &= 4\lambda\epsilon\left(\frac{1}{s^2} - \frac{1}{s}\right) \\
s &= \left(\frac{r}{\sigma}\right)^6 + \frac{1}{2}(1-\lambda) \\
\sigma &= \frac{\sigma_1 + \sigma_2}{2} \\
\epsilon &= \sqrt{\epsilon_1\epsilon_2}
\end{aligned}
$$

Parameters

- **cutoff_distance** (*Number or unit.Quantity*) – The distance at which the nonbonded interaction vanishes.
- **switch_distance** (*Number or unit.Quantity*) – The distance at which the switching function begins to smooth the approach of the nonbonded interaction towards zero.

class `atomsmm.forces.`**`SoftcoreForce`**(*cutoff_distance*, *switch_distance=None*)

Bases: `atomsmm.forces._AtomsMM_CustomNonbondedForce`

A softened version of the Lennard-Jones+Coulomb potential.

$$
\begin{aligned}
V(r) &= V_{\mathrm{vdw}}(r) + V_{\mathrm{coul}}(r) \\
V_{\mathrm{vdw}}(r) &= 4\lambda_{\mathrm{vdw}}\epsilon\left(\frac{1}{s^2} - \frac{1}{s}\right) \\
s &= \left(\frac{r}{\sigma}\right)^6 + \frac{1}{2}(1-\lambda_{\mathrm{vdw}}) \\
\sigma &= \frac{\sigma_1 + \sigma_2}{2} \\
\epsilon &= \sqrt{\epsilon_1\epsilon_2} \\
V_{\mathrm{coul}}(r) &= \lambda_{\mathrm{coul}}\frac{q_1 q_2}{4\pi\epsilon_0}\frac{1}{r}
\end{aligned}
$$

Parameters

- **cutoff_distance** (*Number or unit.Quantity*) – The distance at which the nonbonded interaction vanishes.
- **switch_distance** (*Number or unit.Quantity*) – The distance at which the switching function begins to smooth the approach of the nonbonded interaction towards zero.

4.3 integrators

class `atomsmm.integrators.`**`GlobalThermostatIntegrator`**(*stepSize*, *nveIntegrator*, *thermostat=None*)

This class extends OpenMM's CustomIntegrator class in order to facilitate the construction of NVT integrators which include a global thermostat, that is, one that acts equally and simultaneously on all degrees of freedom of the system. In this case, a complete NVT step is split as:

$$e^{\delta t\, iL_\mathrm{NVT}} = e^{\frac{1}{2}\delta t\, iL_\mathrm{T}} e^{\delta t\, iL_\mathrm{NVE}} e^{\frac{1}{2}\delta t\, iL_\mathrm{T}}$$

The propagator $e^{\delta t\, iL_\mathrm{NVE}}$ is a Hamiltonian

corresponds to a Hamiltonian iL_T

Parameters

- **stepSize** (*unit.Quantity*) – The step size with which to integrate the system (in time unit).
- **nveIntegrator** (`HamiltonianPropagator`) – The Hamiltonian propagator.
- **thermostat** (`ThermostatPropagator`, optional, default=None) – The thermostat propagator.
- **randomSeed** (*int, optional, default=None*) – A seed for random numbers.

class `atomsmm.integrators.`**`MultipleTimeScaleIntegrator`**(*stepSize*, *loops*, *move=None*, *boost=None*, *bath=None*, ***kwargs*)

This class implements a Multiple Time-Scale (MTS) integrator using the RESPA method.

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – A list of *N* integers. Assuming that force group *0* is composed of the fastest forces, while group *N-1* is composed of the slowest ones, *loops[k]* determines how many steps involving forces of group *k* are internally executed for every step involving those of group *k+1*.
- **move** (`Propagator`, optional, default = None) – A move propagator.
- **boost** (`Propagator`, optional, default = None) – A boost propagator.
- **bath** (`Propagator`, optional, default = None) – A bath propagator.

Keyword Arguments

- **`scheme`** (str, optional, default = *middle*) – The splitting scheme used to solve the equations of motion. Available options are *middle*, *xi-respa*, *xo-respa*, *side*, and *blitz*. If it is *middle* (default), then the bath propagator will be inserted between half-step coordinate moves during the fastest-force loops. If it is *xi-respa*, *xo-respa*, or *side*, then the bath propagator will be integrated in both extremities of each loop concerning one of the *N* time scales, with *xi-respa* referring to the time scale of fastest forces (force group *0*), *xo-respa* referring to the time scale of the slowest forces (force group *N-1*), and *side* requiring the user to select the time scale in which to locate the bath propagator via keyword argument *location* (see below). If it is *blitz*, then the force-related propagators will be fully integrated at the outset of each loop in all time scales and the bath propagator will be integrated between half-step coordinate moves during the fastest-force loops.

- **`location`** (*`int, optional, default = None`*) – The index of the force group (from *0* to *N-1*) that defines the time scale in which the bath propagator will be located. This is only meaningful if keyword *scheme* is set to *side* (see above).
- **`nsy`** (*`int, optional, default = 1`*) – The number of Suzuki-Yoshida terms to factorize the bath propagator. Valid options are 1, 3, 7, and 15.
- **`nres`** (*`int, optional, default = 1`*) – The number of RESPA-like subdivisions to factorize the bath propagator.

Warning: The *xo-respa* and *xi-respa* schemes implemented here are slightly different from the ones described in the paper by Leimkuhler, Margul, and Tuckerman [4].

class `atomsmm.integrators.`**`NHL_R_Integrator`**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

This class is an implementation of the massive Nosé-Hoover-Langevin (RESPA) integrator. The method consists in solving the following equations for each degree of freedom (DOF) in the system:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = \frac{f}{m} - v_2 v$$
$$dv_2 = \frac{mv^2 - kT}{Q_2} dt - \gamma v_2 dt + \sqrt{\frac{2\gamma kT}{Q_2}} dW$$

The equations are integrated by a reversible, multiple timescale numerical scheme.

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in *`MultipleTimeScaleIntegrator`*.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which the inertial parameters are computed as $Q_2 = kT\tau^2$.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class *`MultipleTimeScaleIntegrator`* apply here.

`initialize`()

Perform initialization of atomic velocities and other random per-dof variables.

class `atomsmm.integrators.`**`Langevin_R_Integrator`**(*stepSize*, *loops*, *temperature*, *frictionConstant*, ***kwargs*)

This class is an implementation of the multiple time scale Langevin (RESPA) integrator. The method consists in solving the following equations for each degree of freedom (DOF) in the system:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = \frac{f}{m} - \gamma v dt + \sqrt{\frac{2\gamma kT}{m}} dW$$

The equations are integrated by a reversible, multiple timescale numerical scheme.

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

class `atomsmm.integrators.`**`SIN_R_Integrator`**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

This class is an implementation of the Stochastic-Iso-NH-RESPA or SIN(R) method of Leimkuhler, Margul, and Tuckerman [4]. The method consists in solving the following equations for each degree of freedom (DOF) in the system:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = \frac{f}{m} - \lambda v$$
$$\frac{dv_1}{dt} = -\lambda v_1 - v_2 v_1$$
$$dv_2 = \frac{Q_1 v_1^2 - kT}{Q_2} dt - \gamma v_2 dt + \sqrt{\frac{2\gamma kT}{Q_2}} dW$$

where:

$$\lambda = \frac{fv - \frac{1}{2} Q_1 v_2 v_1^2}{mv^2 + \frac{1}{2} Q_1 v_1^2}.$$

A consequence of these equations is that

$$mv^2 + \frac{1}{2} Q_1 v_1^2 = kT.$$

The equations are integrated by a reversible, multiple timescale numerical scheme.

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.

- **timeScale** (*unit.Quantity*) – A time scale τ from which the inertial parameters are computed as $Q_1 = Q_2 = kT\tau^2$.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

`initialize`()

Perform initialization of atomic velocities and other random per-dof variables.

class `atomsmm.integrators.`**`NewMethodIntegrator`**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

This class is an implementation of the Stochastic-Iso-NH-RESPA or SIN(R) method of Leimkuhler, Margul, and Tuckerman [4]. The method consists in solving the following equations for each degree of freedom (DOF) in the system:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = \frac{f}{m} - \lambda v$$
$$\frac{dv_1}{dt} = -\lambda v_1 - v_2 v_1$$
$$dv_2 = \frac{Q_1 v_1^2 - kT}{Q_2} dt - \gamma v_2 dt + \sqrt{\frac{2\gamma kT}{Q_2}} dW$$

where:

$$\lambda = \frac{fv - \frac{1}{2} Q_1 v_2 v_1^2}{mv^2 + \frac{1}{2} Q_1 v_1^2}.$$

A consequence of these equations is that

$$mv^2 + \frac{1}{2} Q_1 v_1^2 = kT.$$

The equations are integrated by a reversible, multiple timescale numerical scheme.

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which the inertial parameters are computed as $Q_1 = Q_2 = kT\tau^2$.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

`initialize`()

Perform initialization of atomic velocities and other random per-dof variables.

class `atomsmm.integrators.`**`LimitedSpeedBAOABIntegrator`**(*stepSize*, *loops*, *temperature*, *frictionConstant*, ***kwargs*)

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

initialize()

Perform initialization of atomic velocities and other random per-dof variables.

class atomsmm.integrators.**LimitedSpeedNHLIntegrator**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

initialize()

Perform initialization of atomic velocities and other random per-dof variables.

class atomsmm.integrators.**LimitedSpeedStochasticIntegrator**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

initialize()

Perform initialization of atomic velocities and other random per-dof variables.

class atomsmm.integrators.**LimitedSpeedStochasticVelocityIntegrator**(*stepSize*, *loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

Parameters

- **stepSize** (*unit.Quantity*) – The largest time step for numerically integrating the system of equations.
- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class `MultipleTimeScaleIntegrator` apply here.

initialize()

Perform initialization of atomic velocities and other random per-dof variables.

class atomsmm.integrators.**ExtendedSystemVariable**(*name*, *mass*, *kT*, *time_scale*, *lower_limit=0*, *upper_limit=1*, *periodic=False*, *thermostat='Nose-Hoover'*, *friction_constant=Quantity(value=0.1, unit=/femtosecond)*)

An extended-system variable used for Adiabatic Free Energy Dynamics (AFED).

Parameters

- **name** (*str*) – The name of the extended-space variable.
- **mass** (*Number or unit.Quantity*) – The mass of the extended-space variable.
- **kT** (*Number of unit.Quantity*) – The temperature of the extended-space variable.
- **time_scale** (*Number of unit.Quantity*) – The time scale of the thermostat that controls the temperature of this variable.
- **lower_limit** (*Number, optional, default=0*) – The lower limit of the variable.
- **upper_limit** (*Number, optional, default=1*) – The upper limit of the variable.
- **periodic** (*Bool, optional, default=False*) – Whether this variable is subject to periodic boundary conditions. If this is *False*, then hard, ellastic walls will be considered instead.

class atomsmm.integrators.**AdiabaticDynamicsIntegrator**(*custom_integrator*, *nsteps*, *variables*)

This class implements the Adiabatic Free Energy Dynamics (AFED) method.

The equations of motion go as follows:

$$\begin{aligned}
e^{2n_{\rm nsteps}\delta t\mathcal{L}} =& [e^{\frac{1}{2}\delta t\mathbf{F}_\lambda^t\nabla_{\mathbf{p}_\lambda}} e^{\delta t\mathcal{L}_{\rm atoms}} e^{\frac{1}{2}\delta t\mathbf{F}_\lambda^t\nabla_{\mathbf{p}_\lambda}}]^{n_{\rm nsteps}} \times \\
& e^{n_{\rm nsteps}\delta t\mathbf{p}_\lambda^t\mathbf{M}_\lambda^{-1}\nabla_\lambda} e^{2n_{\rm nsteps}\delta t\mathcal{L}_{\rm bath}} e^{n_{\rm nsteps}\delta t\mathbf{p}_\lambda^t\mathbf{M}_\lambda^{-1}\nabla_\lambda} \times \\
& [e^{\frac{1}{2}\delta t\mathbf{F}_\lambda^t\nabla_{\mathbf{p}_\lambda}} e^{\delta t\mathcal{L}_{\rm atoms}} e^{\frac{1}{2}\delta t\mathbf{F}_\lambda^t\nabla_{\mathbf{p}_\lambda}}]^{n_{\rm nsteps}}
\end{aligned}$$

Parameters

- **custom_integrator** (*openmm.CustomIntegrator*) – A CustomIntegrator employed to solve the equations of motion of the physical particles, that is, to enact the propagator $e^{\delta t \mathcal{L}_\text{atoms}}$. The size of an overall AFED time step will be given by $\Delta t = 2 n_\text{nsteps} \delta t$, where δt is the time step size previously specified for the *custom_integrator*.
- **nsteps** (*int*) – The number of consecutive *custom_integrator* steps executed in the begining of an overall AFED step, and then again in the end.
- **variables** (list(`ExtendedSystemVariable`)) – A list of extended-system variables whose adiabatic dynamics must be taken into account.

initialize()

Perform initialization of atomic velocities and other random per-dof variables.

4.4 propagators

class `atomsmm.propagators.`**Propagator**

Bases: `object`

This is the base class for propagators, which are building blocks for constructing CustomIntegrator objects in OpenMM. Shortly, a propagator translates the effect of an exponential operator like $e^{\delta t \, iL}$. This effect can be either the exact solution of a system of deterministic or stochastic differential equations or an approximate solution obtained by a splitting scheme such as, for instance, $e^{\delta t \, (iL_A + iL_B)} \approx e^{\delta t \, iL_A} e^{\delta t \, iL_B}$.

integrator(*stepSize*)

This method generates an `Integrator` object which implements the effect of the propagator.

Parameters **stepSize** (*unit.Quantity*) – The step size for integrating the equations of motion.

Returns `Integrator`

class `atomsmm.propagators.`**ChainedPropagator**(*propagators*)

Bases: *`atomsmm.propagators.Propagator`*

This class combines a list of propagators $A1 = e^{\delta t \, iL_{A1}}$ and $B = e^{\delta t \, iL_B}$ by making $C = AB$, that is,

$$e^{\delta t \, iL_C} = e^{\delta t \, iL_A} e^{\delta t \, iL_B}.$$

Warning: Propagators are applied to the system in the right-to-left direction. In general, the effect of the chained propagator is non-commutative. Thus, *ChainedPropagator(A, B)* results in a time-asymmetric propagator unless *A* and *B* commute.

Note: It is possible to create nested chained propagators. If, for instance, B is a chained propagator given by $B = DE$, then an object instantiated by *ChainedPropagator(A, B)* will be a propagator corresponding to $C = ADE$.

Parameters

- **A** (`Propagator`) – The secondly applied propagator in the chain.
- **B** (`Propagator`) – The firstly applied propagator in the chain.

class `atomsmm.propagators.`**`SplitPropagator`**(*A*, *n*)

Bases: *`atomsmm.propagators.Propagator`*

This class splits a propagators $A = e^{\delta t \, iL_A}$ into a sequence of *n* propagators $a = e^{\frac{\delta t}{n} \, iL_A}$, that is,

$$e^{\delta t \, iL_A} = \left(e^{\frac{\delta t}{n} \, iL_A}\right)^n.$$

Parameters

- **A** (*`Propagator`*) – The propagator to be split.
- **n** (*int*) – The number of parts.

class `atomsmm.propagators.`**`TrotterSuzukiPropagator`**(*A*, *B*)

Bases: *`atomsmm.propagators.Propagator`*

This class combines two propagators $A = e^{\delta t \, iL_A}$ and $B = e^{\delta t \, iL_B}$ by using the time-symmetric Trotter-Suzuki splitting scheme [5] $C = B^{1/2} A B^{1/2}$, that is,

$$e^{\delta t \, iL_C} = e^{1/2 \delta t \, iL_B} e^{\delta t \, iL_A} e^{1/2 \delta t \, iL_B}.$$

Note: It is possible to create nested Trotter-Suzuki propagators. If, for instance, B is a Trotter-Suzuki propagator given by $B = E^{1/2} D E^{1/2}$, then an object instantiated by *TrotterSuzukiPropagator(A, B)* will be a propagator corresponding to $C = E^{1/4} D^{1/2} E^{1/4} A E^{1/4} D^{1/2} E^{1/4}$.

Parameters

- **A** (*`Propagator`*) – The middle propagator of a Trotter-Suzuki splitting scheme.
- **B** (*`Propagator`*) – The side propagator of a Trotter-Suzuki splitting scheme.

class `atomsmm.propagators.`**`SuzukiYoshidaPropagator`**(*A*, *nsy=3*)

Bases: *`atomsmm.propagators.Propagator`*

This class splits a propagator $A = e^{\delta t \, iL_A}$ by using a high-order, time-symmetric Suzuki-Yoshida scheme [6][7][8] given by

$$e^{\delta t \, iL_A} = \prod_{i=1}^{n_{sy}} e^{w_i \delta t \, iL_A},$$

where n_{sy} is the number of employed Suzuki-Yoshida weights.

Parameters

- **A** (*`Propagator`*) – The propagator to be splitted by the high-order Suzuki-Yoshida scheme.
- **nsy** (*int, optional, default=3*) – The number of Suzuki-Yoshida weights to be employed. This must be 3, 7, or 15.

class `atomsmm.propagators.`**`TranslationPropagator`**(*constrained=True*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements a coordinate translation propagator $e^{\delta t \mathbf{p}^T \mathbf{M}^{-1} \nabla_\mathbf{r}}$.

Parameters **constrained** (*bool, optional, default=True*) – If *True*, distance constraints are taken into account.

class atomsmm.propagators.**VelocityBoostPropagator**(*constrained=True*)

Bases: *atomsmm.propagators.Propagator*

This class implements a velocity boost propagator $e^{\frac{1}{2}\delta t \mathbf{F}^T \nabla_\mathbf{P}}$.

Parameters **constrained** (*bool, optional, default=True*) – If *True*, distance constraints are taken into account.

class atomsmm.propagators.**MassiveIsokineticPropagator**(*temperature*, *timeScale*, *L*, *forceDependent*)

Bases: *atomsmm.propagators.Propagator*

This class implements an unconstrained, massive isokinetic propagator. It provides, for every degree of freedom in the system, a solution for one of *ODE* systems below.

1. Force-dependent equations:

$$\frac{dv}{dt} = \frac{F}{m} - \lambda_F v$$
$$\frac{dv_1}{dt} = -\lambda_F v_1$$
$$\lambda_F = \frac{Fv}{mv^2 + \frac{1}{2}Q_1 v_1^2}$$

where F is a constant force. The exact solution for these equations is:

$$v = H\hat{v}$$
$$v_1 = Hv_{1,0}$$

where:

$$\hat{v} = v_0 \cosh\left(\frac{Ft}{\sqrt{mkT}}\right) + \sqrt{\frac{kT}{m}} \sinh\left(\frac{Ft}{\sqrt{mkT}}\right)$$
$$H = \sqrt{\frac{kT}{m\hat{v}^2 + \frac{1}{2}Q_1 v_{1,0}^2}}$$

2. Force-indepependent equations:

$$\frac{dv}{dt} = -\lambda_N v$$
$$\frac{dv_1}{dt} = -(\lambda_N + v_2)v_1$$
$$\lambda_N = \frac{-\frac{1}{2}Q_1 v_2 v_1^2}{mv^2 + \frac{1}{2}Q_1 v_1^2}$$

where v_2 is a constant thermostat 'velocity'. In this case, the exact solution is:

$$v = Hv_0$$
$$v_1 = H\hat{v}_1$$

where:

$$\hat{v}_1 = v_{1,0} \exp(-v_2 t)$$
$$H = \sqrt{\frac{kT}{mv_0^2 + \frac{1}{2}Q_1 \hat{v}_1^2}}$$

Both *ODE* systems above satisfy the massive isokinetic constraint $mv^2 + \frac{1}{2}Q_1 v_1^2 = kT$.

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.

- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **forceDependent** (*bool*) – If *True*, the propagator will solve System 1. If *False*, then System 2 will be solved.

class `atomsmm.propagators.`**`NewMethodPropagator`**(*temperature*, *timeScale*, *L*, *forceDependent*)

Bases: *`atomsmm.propagators.Propagator`*

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **forceDependent** (*bool*) – If *True*, the propagator will solve System 1. If *False*, then System 2 will be solved.

class `atomsmm.propagators.`**`RestrainedLangevinPropagator`**(*temperature*, *frictionConstant*, *L*, *kind*)

Bases: *`atomsmm.propagators.Propagator`*

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **forceDependent** (*bool*) – If *True*, the propagator will solve System 1. If *False*, then System 2 will be solved.

class `atomsmm.propagators.`**`LimitedSpeedLangevinPropagator`**(*temperature*, *frictionConstant*, *L*, *kind*)

Bases: *`atomsmm.propagators.Propagator`*

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **kind** (*str*) – Options are *move*, *boost*, and *bath*.

class `atomsmm.propagators.`**`LimitedSpeedNHLPropagator`**(*temperature*, *timeScale*, *frictionConstant*, *L*, *kind*)

Bases: *`atomsmm.propagators.Propagator`*

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.

- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **kind** (*str*) – Options are *move*, *boost*, and *bath*.

class `atomsmm.propagators.`**`LimitedSpeedStochasticPropagator`**(*temperature*, *timeScale*, *frictionConstant*, *L*, *kind*)

Bases: `atomsmm.propagators.Propagator`

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **kind** (*str*) – Options are *move*, *boost*, and *bath*.

class `atomsmm.propagators.`**`LimitedSpeedStochasticVelocityPropagator`**(*temperature*, *timeScale*, *frictionConstant*, *L*, *kind*)

Bases: `atomsmm.propagators.Propagator`

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which to compute the inertial parameter $Q_1 = kT\tau^2$.
- **L** (*int*) – The parameter L.
- **kind** (*str*) – Options are *move*, *boost*, and *bath*.

class `atomsmm.propagators.`**`OrnsteinUhlenbeckPropagator`**(*temperature*, *frictionConstant*, *velocity='v'*, *mass='m'*, *force=None*, *overall=False*, ***globals*)

Bases: `atomsmm.propagators.Propagator`

This class implements an unconstrained, Ornstein-Uhlenbeck (OU) propagator, which provides a solution for the following stochastic differential equation for every degree of freedom in the system:

$$dV = \frac{F}{M}dt - \gamma V dt + \sqrt{\frac{2\gamma kT}{M}}dW.$$

In this equation, *V*, *M*, and *F* are generic forms of velocity, mass, and force. By default, the propagator acts on the atomic velocities (*v*) and masses (*m*), while the forces are considered as null.

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.

- **frictionConstant** (*unit.Quantity, optional, default=None*) – The friction constant γ present in the stochastic equation.
- **velocity** (*str, optional, default='v'*) – The name of a per-dof variable considered as the velocity of each degree of freedom.
- **mass** (*str, optional, default='m'*) – The name of a per-dof or global variable considered as the mass associated to each degree of freedom.
- **force** (*str, optional, default=None*) – The name of a per-dof variable considered as the force acting on each degree of freedom. If it is *None*, then this force is considered as null.

class `atomsmm.propagators.`**`GenericBoostPropagator`**(*velocity='v'*, *mass='m'*, *force='f'*, *perDof=True*, ***globals*)

Bases: `atomsmm.propagators.Propagator`

This class implements a linear boost by providing a solution for the following *ODE* for every degree of freedom in the system:

$$\frac{dV}{dt} = \frac{F}{M}.$$

Parameters

- **velocity** (*str, optional, default='v'*) – The name of a per-dof variable considered as the velocity of each degree of freedom.
- **mass** (*str, optional, default='m'*) – The name of a per-dof or global variable considered as the mass associated to each degree of freedom.
- **force** (*str, optional, default='f'*) – The name of a per-dof variable considered as the force acting on each degree of freedom.

Keyword Arguments **`perDof`** (`bool, default=True`) – This must be *True* if the propagated velocity is a per-dof variable or *False* if it is a global variable.

class `atomsmm.propagators.`**`GenericScalingPropagator`**(*velocity*, *damping*, *perDof=True*, ***globals*)

Bases: `atomsmm.propagators.Propagator`

This class implements scaling by providing a solution for the following *ODE* for every degree of freedom in the system:

$$\frac{dV}{dt} = -\lambda_{\rm damping} * V.$$

Parameters

- **velocity** (*str*) – The name of a per-dof variable considered as the velocity of each degree of freedom.
- **damping** (*str*) – The name of a per-dof or global variable considered as the damping parameter associated to each degree of freedom.

class `atomsmm.propagators.`**`RespaPropagator`**(*loops*, *move=None*, *boost=None*, *core=None*, *shell=None*, ***kwargs*)

Bases: `atomsmm.propagators.Propagator`

This class implements a multiple timescale (MTS) rRESPA propagator [9] with N force groups, where group 0 goes in the innermost loop (shortest time step) and group $N-1$ goes in the outermost loop (largest time step). The complete Liouville-like operator corresponding to the equations of motion is split as

$$iL = iL_{\rm move} + \sum_{k=0}^{N-1} \left(iL_{{\rm boost},k} \right) + iL_{\rm core} + \sum_{k=0}^{N-1} \left(iL_{{\rm shell},k} \right)$$

In this scheme, iL_move is the only component that entails changes in the atomic coordinates, while $iL_{\text{boost},k}$ is the only component that depends on the forces of group k. Therefore, operator iL_core and each operator $iL_{\text{shell},k}$ are reserved to changes in atomic velocities due to the action of thermostats, as well as to changes in the thermostat variables themselves.

The rRESPA split can be represented recursively as

$$e^{\Delta t iL} = e^{\Delta t iL_{N-1}}$$

where

$$e^{\delta t iL_k} = \begin{cases} \left(e^{\frac{\delta t}{2n_k} iL_{\text{shell},k}} e^{\frac{\delta t}{2n_k} iL_{\text{boost},k}} e^{\frac{\delta t}{n_k} iL_{k-1}} e^{\frac{\delta t}{2n_k} iL_{\text{boost},k}} e^{\frac{\delta t}{2n_k} iL_{\text{shell},k}}\right)^{n_k} & k \geq 0 \\ e^{\frac{\delta t}{2} iL_\text{move}} e^{\delta t iL_\text{core}} e^{\frac{\delta t}{2} iL_\text{move}} & k = -1 \end{cases}$$

Parameters

- **loops** (*list(int)*) – A list of *N* integers, where *loops[k]* determines how many iterations of force group *k* are internally executed for every iteration of force group *k+1*.
- **move** (`Propagator`, optional, default=None) – A propagator used to update the coordinate of every atom based on its current velocity. If it is *None*, then an unconstrained, linear translation is applied.
- **boost** (`Propagator`, optional, default=None) – A propagator used to update the velocity of every atom based on the resultant force acting on it. If it is *None*, then an unconstrained, linear boosting is applied.
- **core** (`Propagator`, optional, default=None) – An internal propagator to be used for controlling the configurational probability distribution sampled by the rRESPA scheme. This propagator will be integrated in the innermost loop (shortest time step). If it is *None* (default), then no core propagator will be applied.
- **shell** (dict(int : `Propagator`), optional, default=None) – A dictionary of propagators to be used for controlling the configurational probability distribution sampled by the rRESPA scheme. Propagator *shell[k]* will be excecuted in both extremities of the loop involving forces of group *k*. If it is *None* (default), then no shell propagators will be applied. Dictionary keys must be integers from *0* to *N-1* and omitted keys mean that no shell propagators will be considered at those particular loop levels.
- **has_memory** (*bool, optional, default=True*) – If *True*, integration in the fastest time scale remembers the lattest forces computed in all other time scales. To compensate, each remembered force is substracted during the integration in its respective time scale. **Warning**: this integration scheme is not time-reversal symmetric.

class `atomsmm.propagators.`**`MultipleTimeScalePropagator`**(*loops*, *move=None*, *boost=None*, *bath=None*, ***kwargs*)

Bases: `atomsmm.propagators.RespaPropagator`

This class implements a Multiple Time-Scale (MTS) propagator using the RESPA method.

Parameters

- **loops** (*list(int)*) – A list of *N* integers. Assuming that force group *0* is composed of the fastest forces, while group *N-1* is composed of the slowest ones, *loops[k]* determines how many steps involving forces of group *k* are internally executed for every step involving those of group *k+1*.
- **move** (`Propagator`, optional, default = None) – A move propagator.
- **boost** (`Propagator`, optional, default = None) – A boost propagator.

- **bath** (`Propagator`, optional, default = None) – A bath propagator.

Keyword Arguments

- **`scheme`** (str, optional, default = *middle*) – The splitting scheme used to solve the equations of motion. Available options are *middle*, *xi-respa*, *xo-respa*, *side*, and *blitz*. If it is *middle* (default), then the bath propagator will be inserted between half-step coordinate moves during the fastest-force loops. If it is *xi-respa*, *xo-respa*, or *side*, then the bath propagator will be integrated in both extremities of each loop concerning one of the *N* time scales, with *xi-respa* referring to the time scale of fastest forces (force group *0*), *xo-respa* referring to the time scale of the slowest forces (force group *N-1*), and *side* requiring the user to select the time scale in which to locate the bath propagator via keyword argument *location* (see below). If it is *blitz*, then the force-related propagators will be fully integrated at the outset of each loop in all time scales and the bath propagator will be integrated between half-step coordinate moves during the fastest-force loops.
- **`location`** (*`int, optional, default = None`*) – The index of the force group (from *0* to *N-1*) that defines the time scale in which the bath propagator will be located. This is only meaningful if keyword *scheme* is set to *side* (see above).
- **`nsy`** (*`int, optional, default = 1`*) – The number of Suzuki-Yoshida terms to factorize the bath propagator. Valid options are 1, 3, 7, and 15.
- **`nres`** (*`int, optional, default = 1`*) – The number of RESPA-like subdivisions to factorize the bath propagator.

Warning: The *xo-respa* and *xi-respa* schemes implemented here are slightly different from the ones described in the paper by Leimkuhler, Margul, and Tuckerman [4].

class `atomsmm.propagators.`**`SIN_R_Propagator`**(*loops*, *temperature*, *timeScale*, *frictionConstant*, ***kwargs*)

Bases: `atomsmm.propagators.MultipleTimeScalePropagator`

This class is an implementation of the Stochastic-Iso-NH-RESPA or SIN(R) method of Leimkuhler, Margul, and Tuckerman [4]. The method consists in solving the following equations for each degree of freedom (DOF) in the system:

$$\frac{dx}{dt} = v$$
$$\frac{dv}{dt} = \frac{f}{m} - \lambda v$$
$$\frac{dv_1}{dt} = -\lambda v_1 - v_2 v_1$$
$$dv_2 = \frac{Q_1 v_1^2 - kT}{Q_2} dt - \gamma v_2 dt + \sqrt{\frac{2\gamma kT}{Q_2}} dW$$

where:

$$\lambda = \frac{fv - \frac{1}{2} Q_1 v_2 v_1^2}{mv^2 + \frac{1}{2} Q_1 v_1^2}.$$

A consequence of these equations is that

$$mv^2 + \frac{1}{2} Q_1 v_1^2 = kT.$$

The equations are integrated by a reversible, multiple timescale numerical scheme.

Parameters

- **loops** (*list(int)*) – See description in `MultipleTimeScaleIntegrator`.
- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **timeScale** (*unit.Quantity*) – A time scale τ from which the inertial parameters are computed as $Q_1 = Q_2 = kT\tau^2$.
- **frictionConstant** (*unit.Quantity*) – The friction constant γ present in the stochastic equation of motion for per-DOF thermostat variable v_2.
- ****kwargs** (*keyword arguments*) – The same keyword arguments of class *`MultipleTimeScalePropagator`* apply here.

class `atomsmm.propagators.`**`VelocityVerletPropagator`**

Bases: *`atomsmm.propagators.Propagator`*

This class implements a Velocity Verlet propagator with constraints.

$$e^{\delta t \, iL_\mathrm{NVE}} = e^{\frac{1}{2}\delta t \mathbf{F}^T \nabla_\mathbf{p}} e^{\delta t \mathbf{p}^T \mathbf{M}^{-1} \nabla_\mathbf{r}} e^{\frac{1}{2}\delta t \mathbf{F}^T \nabla_\mathbf{p}}$$

Note: In the original OpenMM VerletIntegrator class, the implemented propagator is a leap-frog version of the Verlet method.

class `atomsmm.propagators.`**`UnconstrainedVelocityVerletPropagator`**

Bases: *`atomsmm.propagators.Propagator`*

This class implements a Velocity Verlet propagator with constraints.

$$e^{\delta t \, iL_\mathrm{NVE}} = e^{\frac{1}{2}\delta t \mathbf{F}^T \nabla_\mathbf{p}} e^{\delta t \mathbf{p}^T \mathbf{M}^{-1} \nabla_\mathbf{r}} e^{\frac{1}{2}\delta t \mathbf{F}^T \nabla_\mathbf{p}}$$

Note: In the original OpenMM VerletIntegrator class, the implemented propagator is a leap-frog version of the Verlet method.

class `atomsmm.propagators.`**`VelocityRescalingPropagator`**(*temperature*, *degreesOfFreedom*, *timeScale*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements the Stochastic Velocity Rescaling propagator of Bussi, Donadio, and Parrinello [10], which is a global version of the Langevin thermostat [11].

This propagator provides a solution for the following *SDE* [11]:

$$d\mathbf{p} = \frac{1}{2\tau}\left[\frac{(N_f - 1)k_B T}{2K} - 1\right]\mathbf{p}dt + \sqrt{\frac{k_B T}{2K\tau}}\mathbf{p}dW$$

The gamma-distributed random numbers required for the solution are generated by using the algorithm of Marsaglia and Tsang [12].

Warning: An integrator that uses this propagator will fail if no initial velocities are provided to the system particles.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **degreesOfFreedom** (*int*) – The number of degrees of freedom in the system, which can be retrieved via function `countDegreesOfFreedom()`.
- **timeScale** (*unit.Quantity*) – The relaxation time of the thermostat.

class `atomsmm.propagators.`**`NoseHooverPropagator`**(*temperature*, *degreesOfFreedom*, *timeScale*, *nloops=1*)

Bases: `atomsmm.propagators.Propagator`

This class implements a Nose-Hoover propagator.

As usual, the inertial parameter Q is defined as $Q = N_f k_B T \tau^2$, with τ being a relaxation time [9].

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **degreesOfFreedom** (*int*) – The number of degrees of freedom in the system, which can be retrieved via function `countDegreesOfFreedom()`.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **nloops** (*int, optional, default=1*) – Number of RESPA-like subdivisions.

class `atomsmm.propagators.`**`MassiveNoseHooverPropagator`**(*temperature*, *timeScale*, *nloops=1*)

Bases: `atomsmm.propagators.Propagator`

This class implements a massive Nose-Hoover propagator.

As usual, the inertial parameter Q is defined as $Q = N_f k_B T \tau^2$, with τ being a relaxation time [9].

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **nloops** (*int, optional, default=1*) – Number of RESPA-like subdivisions.

class `atomsmm.propagators.`**`MassiveGeneralizedGaussianMomentPropagator`**(*temperature*, *timeScale*, *nloops=1*)

Bases: `atomsmm.propagators.Propagator`

This class implements a massive Generalized Gaussian Moment propagator.

As usual, the inertial parameter Q is defined as $Q = N_f k_B T \tau^2$, with τ being a relaxation time [9].

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **nloops** (*int, optional, default=1*) – Number of RESPA-like subdivisions.

class `atomsmm.propagators.`**`NoseHooverChainPropagator`**(*temperature*, *degreesOfFreedom*, *timeScale*, *frictionConstant=None*)

Bases: `atomsmm.propagators.Propagator`

This class implements a Nose-Hoover chain [9] with two global thermostats.

This propagator provides a solution for the following *ODE* system:

$$\begin{aligned}
\frac{d\mathbf{p}}{dt} &= -\frac{p_{\eta,1}}{Q_1}\mathbf{p} \\
\frac{dp_{\eta,1}}{dt} &= \mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} - N_f k_B T - \frac{p_{\eta,2}}{Q_2}p_{\eta,1} \\
\frac{dp_{\eta,2}}{dt} &= \frac{p_{\eta,1}^2}{Q_1} - k_B T
\end{aligned}$$

As usual, the inertial parameter Q is defined as $Q = N_f k_B T\tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{(\delta t/2)\mathcal{L}_{B2}}e^{(\delta t/2)\mathcal{L}_{S1}}e^{(\delta t/2)\mathcal{L}_{B1}}e^{(\delta t)\mathcal{L}_{S}}e^{(\delta t/2)\mathcal{L}_{B1}}e^{(\delta t/2)\mathcal{L}_{S1}}e^{(\delta t/2)\mathcal{L}_{B2}}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B2' is a boost of thermostat 2, whose solution is:

$$p_{\eta,2}(t) = p_{\eta,2}^0 + \left(\frac{p_{\eta,1}^2}{Q_1} - k_B T\right)t$$

Equation 'S1' is a scaling of thermostat 1, whose solution is:

$$p_{\eta,1}(t) = p_{\eta,1}^0 e^{-\frac{p_{\eta,2}}{Q_2}t}$$

Equation 'B1' is a boost of thermostat 1, whose solution is:

$$p_{\eta,1}(t) = p_{\eta,1}^0 + \left(\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} - N_f k_B T\right)t$$

Equation 'S' is a scaling of particle momenta, whose solution is:

$$\mathbf{p}(t) = \mathbf{p}_0 e^{-\frac{p_{\eta,1}}{Q_1}t}$$

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **degreesOfFreedom** (*int*) – The number of degrees of freedom in the system, which can be retrieved via function `countDegreesOfFreedom()`.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

class atomsmm.propagators.**NoseHooverLangevinPropagator**(*temperature*, *degreesOfFreedom*, *timeScale*, *frictionConstant=None*)

Bases: *atomsmm.propagators.Propagator*

This class implements a Nose-Hoover-Langevin propagator [13][14], which is similar to a Nose-Hoover chain [9] of two thermostats, but with the second one being a stochastic (Langevin-type) rather than a deterministic thermostat.

This propagator provides a solution for the following *SDE* system:

$$d\mathbf{p} = -\frac{p_\eta}{Q}\mathbf{p}dt \qquad \text{(S)}$$

$$dp_\eta = (\mathbf{p}^T\mathbf{M}^{-1}\mathbf{p} - N_f k_B T)dt - \gamma p_\eta dt + \sqrt{2\gamma Q k_B T}dW \qquad \text{(O)}$$

As usual, the inertial parameter Q is defined as $Q = N_f k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$p_\eta(t) = p_{\eta 0} + (\mathbf{p}^T \mathbf{M}^{-1} \mathbf{p} - N_f k_B T) t$$

Equation 'S' is a scaling, whose solution is:

$$\mathbf{p}(t) = \mathbf{p}_0 e^{-\frac{p_\eta}{Q} t}$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$p_\eta(t) = p_{\eta 0} e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_N$$

where R_N is a normally distributed random number.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **degreesOfFreedom** (*int*) – The number of degrees of freedom in the system, which can be retrieved via function `countDegreesOfFreedom()`.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

class atomsmm.propagators.**RegulatedTranslationPropagator**(*temperature*, *n*, *alpha_n=1*)

Bases: *atomsmm.propagators.Propagator*

An unconstrained, regulated translation propagator which provides, for every degree of freedom in the system, a solution for the following *ODE*:

$$\frac{dr_i}{dt} = c_i \tanh\left(\frac{\alpha_n p_i}{m_i c_i}\right)$$

where $c_i = \sqrt{\frac{\alpha_n n k_B T}{m_i}}$ is the speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. The exact solution for this equation is:

$$r_i(t) = r_i^0 + c_i \mathrm{tanh}\left(\frac{\alpha_n p}{m c_i}\right) t$$

where r_i^0 is the initial coordinate.

Parameters

- **temperature** (*unit.Quantity*) – The temperature to which the configurational sampling should correspond.
- **n** (*int or float*) – The regulating parameter.

Keyword Arguments **alpha_n** (*int or float, default=1*) – Another regulating parameter.

class atomsmm.propagators.**RegulatedBoostPropagator**

Bases: *atomsmm.propagators.Propagator*

An unconstrained, regulated boost propagator which provides, for every degree of freedom in the system, a solution for the following *ODE*:

$$\frac{dp_i}{dt} = F_i$$

where F_i is a constant force. The exact solution for this equation is:

$$p_i(t) = p_i^0 + F_i t$$

where p_i^0 is the initial momentum.

class atomsmm.propagators.**RegulatedMassiveNoseHooverLangevinPropagator**(*temperature*, *n*, *timeScale*, *frictionConstant*, *alpha_n=1*, *split=False*, *adiabatic=False*)

Bases: *atomsmm.propagators.Propagator*

This class implements a regulated version of the massive Nose-Hoover-Langevin propagator [13][14]. It provides, for every degree of freedom in the system, a solution for the following *SDE* system:

$$\begin{aligned} dp_i &= -v_{\eta_i} p_i dt \\ dv_{\eta_i} &= \frac{p_i v_i - k_B T}{Q} dt - \gamma v_{\eta_i} dt + \sqrt{\frac{2\gamma k_B T}{Q}} dW_i, \end{aligned}$$

where:

$$v_i = c_i \tanh\left(\frac{\alpha_n p_i}{m_i c_i}\right).$$

Here, $c_i = \sqrt{\frac{\alpha_n n k_B T}{m_i}}$ is the speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. As usual, the inertial parameter Q is defined as $Q = k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{\delta t \mathcal{L}} = e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 + \frac{p_i v_i - k_B T}{Q} t$$

Equation 'S' is a scaling, whose solution is:

$$p_i(t) = p_i^0 e^{-v_{\eta_i} t}$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_{N,i}$$

where $R_{N,i}$ is a normally distributed random number.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **n** (*int or float*) – The regulating parameter.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

Keyword Arguments **`alpha_n`** (*`int or float, default=1`*) – Another regulating parameter.

class `atomsmm.propagators.`**`TwiceRegulatedMassiveNoseHooverLangevinPropagator`**(*temperature, n, timeScale, frictionConstant, alpha_n=1, split=False, adiabatic=False*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements a doubly-regulated version of the massive Nose-Hoover-Langevin propagator [13][14]. It provides, for every degree of freedom in the system, a solution for the following *SDE* system:

$$\begin{aligned} dp_i &= -v_{\eta_i} m_i v_i dt \\ dv_{\eta_i} &= \frac{1}{Q}\left(\frac{n+1}{n\alpha_n} m_i v_i^2 - k_B T\right) dt - \gamma v_{\eta_i} dt + \sqrt{\frac{2\gamma k_B T}{Q}} dW_i, \end{aligned}$$

where:

$$v_i = c_i \tanh\left(\frac{\alpha_n p_i}{m_i c_i}\right).$$

Here, $c_i = \sqrt{\alpha_n n m_i k T}$ is speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. As usual, the inertial parameter Q is defined as $Q = k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{\delta t \mathcal{L}} = e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 + \frac{1}{Q}\left(\frac{n+1}{\alpha_n n} m_i v_i^2 - k_B T\right) t$$

Equation 'S' is a scaling, whose solution is:

$$p_i(t) = \frac{m_i c_i}{\alpha_n} \text{arcsinh} \left[\sinh\left(\frac{\alpha_n p_i}{m_i c_i}\right) e^{-\alpha_n v_{\eta_i} t} \right]$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_N$$

where R_N is a normally distributed random number.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **n** (*int or float*) – The regulating parameter.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

Keyword Arguments **`alpha_n`** (*`int or float, default=1`*) – Another regulating parameter.

`class` `atomsmm.propagators.`**`RegulatedAtomicNoseHooverLangevinPropagator`**(*temperature*, *n*, *timeScale*, *frictionConstant*, *alpha_n=1*, *split=False*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements a regulated version of the massive Nose-Hoover-Langevin propagator [13][14]. It provides, for every degree of freedom in the system, a solution for the following *SDE* system:

$$\begin{aligned} dp_i &= -v_{\eta_i} p_i dt \\ dv_{\eta_i} &= \frac{p_i v_i - k_B T}{Q} dt - \gamma v_{\eta_i} dt + \sqrt{\frac{2\gamma k_B T}{Q}} dW_i, \end{aligned}$$

where:

$$v_i = c_i \tanh\left(\frac{\alpha_n p_i}{m_i c_i}\right).$$

Here, $c_i = \sqrt{\frac{\alpha_n n k_B T}{m_i}}$ is the speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. As usual, the inertial parameter Q is defined as $Q = k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{\delta t \mathcal{L}} = e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 + \frac{p_i v_i - k_B T}{Q} t$$

Equation 'S' is a scaling, whose solution is:

$$p_i(t) = p_i^0 e^{-v_{\eta_i} t}$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_{N,i}$$

where $R_{N,i}$ is a normally distributed random number.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **n** (*int or float*) – The regulating parameter.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

Keyword Arguments **`alpha_n`** (*`int or float, default=1`*) – Another regulating parameter.

`class` `atomsmm.propagators.`**`TwiceRegulatedAtomicNoseHooverLangevinPropagator`**(*temperature*, *n*, *timeScale*, *frictionConstant*, *alpha_n=1*, *split=False*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements a doubly-regulated version of the atomic Nose-Hoover-Langevin propagator [13][14]. It provides, for every atom in the system, a solution for the following *SDE* system:

$$dv_i = -v_{\eta,\text{atom}_i} v_i \left[1 - \left(\frac{v_i}{c_i}\right)^2\right] dt$$

$$dv_{\eta,j} = \frac{1}{Q}\left(\frac{n+1}{\alpha_n n} m_j \mathbf{v}_j^T \mathbf{v}_j - 3k_B T\right) dt - \gamma v_{\eta,j} dt + \sqrt{\frac{2\gamma k_B T}{Q}} dW_j,$$

where $c_i = \sqrt{\alpha_n n m_i k T}$ is speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. As usual, the inertial parameter Q is defined as $Q = 3k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{\delta t \mathcal{L}} = e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$v_{\eta,j}(t) = v_{\eta,j}^0 + \frac{1}{Q}\left(\frac{n+1}{\alpha_n n} m_j \mathbf{v}_j^T \mathbf{v}_j - 3k_B T\right) t$$

Equation 'S' is a scaling, whose solution is:

$$v_{s,i}(t) = v_i^0 e^{-\alpha_n v_{\eta_i} t}$$
$$v_i(t) = \frac{v_{s,i}(t)}{\sqrt{1 - \left(\frac{v_i^0}{c_i}\right)^2 + \left(\frac{v_{s,i}(t)}{c_i}\right)^2}}$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$v_{\eta_i}(t) = v_{\eta_i}^0 e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_N$$

where R_N is a normally distributed random number.

Parameters

- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **n** (*int or float*) – The regulating parameter.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

Keyword Arguments **`alpha_n`** (*`int or float, default=1`*) – Another regulating parameter.

class `atomsmm.propagators.`**`TwiceRegulatedGlobalNoseHooverLangevinPropagator`**(*degreesOfFreedom, temperature, n, timeScale, frictionConstant, alpha_n=1, split=False*)

Bases: *`atomsmm.propagators.Propagator`*

This class implements a doubly-regulated version of the global Nose-Hoover-Langevin propagator [13][14]. It provides, for every degree of freedom in the system, a solution for the following *SDE* system:

$$dv_i = -\alpha_n v_\eta v_i \left[1 - \left(\frac{v_i}{c_i}\right)^2\right] dt$$

$$dv_\eta = \frac{1}{Q}\left(\frac{n+1}{n\alpha_n}\mathbf{v}^T\mathbf{M}\mathbf{v} - N_f k_B T\right) dt - \gamma v_\eta dt + \sqrt{\frac{2\gamma k_B T}{Q}} dW,$$

where $c_i = \sqrt{\alpha_n n m_i kT}$ is speed limit for such degree of freedom and, by default, $\alpha_n = \frac{n+1}{n}$. As usual, the inertial parameter Q is defined as $Q = N_f k_B T \tau^2$, with τ being a relaxation time [9]. An approximate solution is obtained by applying the Trotter-Suzuki splitting formula:

$$e^{\delta t \mathcal{L}} = e^{(\delta t/2)\mathcal{L}_B} e^{(\delta t/2)\mathcal{L}_S} e^{\delta t \mathcal{L}_O} e^{(\delta t/2)\mathcal{L}_S} e^{(\delta t/2)\mathcal{L}_B}$$

Each exponential operator above is the solution of a differential equation.

Equation 'B' is a boost, whose solution is:

$$v_\eta(t) = v_\eta^0 + \frac{1}{Q}\left(\frac{n+1}{n\alpha_n}\mathbf{v}^T\mathbf{M}\mathbf{v} - N_f k_B T\right)t$$

Equation 'S' is a scaling, whose solution is:

$$v_{s,i}(t) = v_i^0 e^{-\alpha_n v_\eta t}$$
$$v_i(t) = \frac{v_{s,i}(t)}{\sqrt{1 - \left(\frac{v_i^0}{c_i}\right)^2 + \left(\frac{v_{s,i}(t)}{c_i}\right)^2}}$$

Equation 'O' is an Ornstein–Uhlenbeck process, whose solution is:

$$v_\eta(t) = v_\eta^0 e^{-\gamma t} + \sqrt{\frac{k_B T}{Q}(1 - e^{-2\gamma t})} R_N$$

where R_N is a normally distributed random number.

Parameters

- **degreesOfFreedom** (*int*) – The number of degrees of freedom in the system
- **temperature** (*unit.Quantity*) – The temperature of the heat bath.
- **n** (*int or float*) – The regulating parameter.
- **timeScale** (*unit.Quantity (time)*) – The relaxation time of the Nose-Hoover thermostat.
- **frictionConstant** (*unit.Quantity (1/time)*) – The friction coefficient of the Langevin thermostat.

Keyword Arguments **`alpha_n`** (*`int or float, default=1`*) – Another regulating parameter.

class `atomsmm.propagators.`**`ExtendedSystemPropagator`**(*parameter*, *mass*, *period*, *propagator*, *group=None*)

Bases: *`atomsmm.propagators.Propagator`*

4.5 reporters

class `atomsmm.reporters.`**`ExtendedStateDataReporter`**(*file*, *reportInterval*, ***kwargs*)

An extension of OpenMM's StateDataReporter class, which outputs information about a simulation, such as energy and temperature, to a file.

All original functionalities of StateDataReporter are preserved and the following ones are included:

1. Report the Coulomb contribution of the potential energy (keyword: *coulombEnergy*):

   This contribution includes both real- and reciprocal-space terms.

2. Report the atomic virial of a fully-flexible system (keyword: *atomicVirial*):

Considering full scaling of atomic coordinates in a box volume change (i.e. without any distance constraints), the internal virial of the system is given by

$$W = -\sum_{i,j} r_{ij} E'(r_{ij}),$$

where $E'(r)$ is the derivative of the pairwise interaction potential as a function of the distance between to atoms. Such interaction includes van der Waals, Coulomb, and bond-stretching contributions. Bond-bending and dihedral angles are not considered because they are invariant to full volume-scaling of atomic coordinates.

3. Report the nonbonded contribution of the atomic virial (keyword: *nonbondedVirial*):

   The nonbonded virial is given by

$$W_{\text{nb}} = -\sum_{i,j} r_{ij} E'_{\text{nb}}(r_{ij}),$$

   where $E'_{\text{nb}}(r)$ is the derivative of the nonbonded pairwise potential, which comprises van der Waals and Coulomb interactions only.

4. Report the atomic pressure of a fully-flexible system (keyword: *atomicPressure*):

$$P = \frac{2K + W}{3V},$$

   where K is the kinetic energy sum for all atoms in the system. If keyword *bathTemperature* is employed (see below), the instantaneous kinetic energy is substituted by its equipartition-theorem average $\langle K \rangle = 3N_{\text{atoms}} k_B T/2$, where T is the heat-bath temperature.

5. Report the molecular virial of a system (keyword: *molecularVirial*):

   To compute the molecular virial, only the center-of-mass coordinates of the molecules are considered to scale in a box volume change, while the internal molecular structure is kept unaltered. The molecular virial is computed from the nonbonded part of the atomic virial by using the formulation of Ref. [1]:

$$W_{\text{mol}} = W - \sum_i (\mathbf{r}_i - \mathbf{r}_i^{\text{cm}}) \cdot \mathbf{F}_i,$$

   where $\mathbf{r}_i$ is the coordinate of atom i, $\mathbf{F}_i$ is the resultant pairwise force acting on it (excluding bond-bending and dihedral angles), and $\mathbf{r}_i^{\text{cm}}$ is the center-of-mass coordinate of its containing molecule.

6. Report the molecular pressure of a system (keyword: *molecularPressure*):

$$P = \frac{2K_{\text{mol}} + W_{\text{mol}}}{3V},$$

   where K_{mol} is the center-of-mass kinetic energy summed for all molecules in the system. If keyword *bathTemperature* is employed (see below), the instantaneous kinetic energy is substituted by its equipartition-theorem average $\langle K_{\text{mol}} \rangle = 3N_{\text{mols}} k_B T/2$, where T is the heat-bath temperature.

7. Report the center-of-mass kinetic energy (keyword: *molecularKineticEnergy*):

$$K_{\text{mol}} = \frac{1}{2} \sum_{i=1}^{N_{\text{mol}}} M_i v_{\text{cm},i}^2,$$

where N_mol is the number of molecules in the system, M_i is the total mass of molecule *i*, and $v_{\mathrm{cm},i}$ is the center-of-mass velocity of molecule *i*.

8. Report potential energies at multiple global parameter states (keyword: *globalParameterStates*):

   Computes and reports the potential energy of the system at a number of provided global parameter states.

9. Report global parameter values (keyword: *globalParameters*):

   Reports the values of specified global parameters.

10. Report derivatives of energy with respect to global parameters (keyword: *energyDerivatives*):

    Computes and reports derivatives of the potential energy of the system at the current state with respect to specified global parameters.

11. Report values of collective variables (keyword: *collectiveVariables*)

    Report the values of a set of collective variables.

12. Allow specification of an extra file for reporting (keyword: *extraFile*).

    This can be used for replicating a report simultaneously to *sys.stdout* and to a file using a unique reporter.

Keyword Arguments

- **`coulombEnergy`** (*`bool, optional, default=False`*) – Whether to write the Coulomb contribution of the potential energy to the file.
- **`atomicVirial`** (*`bool, optional, default=False`*) – Whether to write the total atomic virial to the file.
- **`nonbondedVirial`** (*`bool, optional, default=False`*) – Whether to write the nonbonded contribution to the atomic virial to the file.
- **`atomicPressure`** (*`bool, optional, default=False`*) – Whether to write the internal atomic pressure to the file.
- **`molecularVirial`** (*`bool, optional, default=False`*) – Whether to write the molecular virial to the file.
- **`molecularPressure`** (*`bool, optional, default=False`*) – Whether to write the internal molecular pressure to the file.
- **`molecularKineticEnergy`** (*`bool, optional, default=False`*) – Whether to write the molecular center-of-mass kinetic energy to the file.
- **`globalParameterStates`** (pandas.DataFrame, optional, default=None) – A DataFrame containing context global parameters (column names) and sets of values thereof. If it is provided, then the potential energy will be reported for every state these parameters define.
- **`globalParameters`** (*`list(str), optional, default=None`*) – A list of global parameter names. If it is provided, then the values of these parameters will be reported.
- **`energyDerivatives`** (*`list(str), optional, default=None`*) – A list of global parameter names. If it is provided, then the derivatives of the total potential energy with respect to these parameters will be reported. It is necessary that the calculation of these derivatives has been activated beforehand (see, for instance, CustomIntegrator).

- **collectiveVariables** (*list(openmm.CustomCVForce), optional, default=None*) – A list of CustomCVForce objects. If it is provided, then the values of all collective variables associated with these objects will be reported.
- **pressureComputer** (*PressureComputer*, optional, default=None) – A computer designed to determine pressures and virials. This is mandatory if any keyword related to virial or pressure is set as *True*.
- **extraFile** (*str or file, optional, default=None*) – Extra file to write to, specified as a file name or a file object.

class atomsmm.reporters.**XYZReporter**(*file*, *reportInterval*, ***kwargs*)

Outputs to an XYZ-format file a series of frames containing the coordinates, velocities, momenta, or forces on all atoms in a Simulation.

Note: Coordinates are expressed in nanometers, velocities in nanometer/picosecond, momenta in dalton*nanometer/picosecond, and forces in dalton*nanometer/picosecond^2.

To use this reporter, create an XYZReporter object and append it to the Simulation's list of reporters.

Keyword Arguments

- **output** (*str, default='positions'*) – Which kind of info to report. Valid options are 'positions', 'velocities', 'momenta' and 'forces'.
- **groups** (*set(int), default=None*) – Which force groups to consider in the force calculations. If this is *None*, then all force groups will be evaluated.

class atomsmm.reporters.**CenterOfMassReporter**(*file*, *reportInterval*, ***kwargs*)

Outputs to an XYZ-format file a series of frames containing the center-of-mass coordinates, center-of-mass velocities, total momenta, or resultant forces on all molecules in a Simulation.

Note: Coordinates are expressed in nanometers, velocities in nanometer/picosecond, momenta in dalton*nanometer/picosecond, and forces in dalton*nanometer/picosecond^2.

To use this reporter, create an CenterOfMassReporter object and append it to the Simulation's list of reporters.

Keyword Arguments

- **output** (*str, default='positions'*) – Which kind of info to report. Valid options are 'positions', 'velocities', 'momenta' and 'forces'.
- **groups** (*set(int), default=None*) – Which force groups to consider in the force calculations. If this is *None*, then all force groups will be evaluated.

class atomsmm.reporters.**CustomIntegratorReporter**(*file*, *reportInterval*, ***kwargs*)

Outputs global and per-DoF variables of a CustomIntegrator instance.

Keyword Arguments **describeOnly** (*bool, optional, default=True*) – Whether to output only descriptive statistics that summarize the activated per-Dof variables.

class atomsmm.reporters.**ExpandedEnsembleReporter**(*file*, *reportInterval*, *states*, *temperature*, ***kwargs*)

Performs an Expanded Ensemble simulation and reports the energies of multiple states.

Parameters

- **states** (*pandas.DataFrame_*) – A DataFrame containing context global parameters (column names) and sets of values thereof. The potential energy will be reported for every state these parameters define. If one of the variables is named as *weight*, then its set of values will be assigned to every state as an importance sampling weight. Otherwise, all states will have identical weights. States which are supposed to only have their energies reported, with no actual visits, can have their weights set up to *-inf*.

- **temperature** (*unit.Quantity*) – The system temperature.

Keyword Arguments **`reportsPerExchange`** (*`int, optional, default=1`*) – The number of reports between attempts to exchange the global parameter state, that is, the exchange interval measured in units of report intervals.

`state_sampling_analysis`(*staging_variable=None*, *to_file=True*, *isochronal_n=2*)
Build histograms of states visited during the overall process as well as during downhill walks.

Returns *pandas.DataFrame_*

4.6 systems

`class` `atomsmm.systems.`**`RESPASystem`**(*system*, *rcutIn*, *rswitchIn*, ***kwargs*)
Bases: `simtk.openmm.openmm.System`

An OpenMM System prepared for Multiple Time-Scale Integration with RESPA.

Parameters

- **system** (*openmm.System*) – The original system from which to generate the RESPASystem.
- **rcutIn** (*unit.Quantity*) – The distance at which the short-range nonbonded interactions will completely vanish.
- **rswitchIn** (*unit.Quantity*) – The distance at which the short-range nonbonded interactions will start vanishing by application of a switching function.

Keyword Arguments

- **`adjustment`** (*`str, optional, default='force-switch'`*) – A keyword for modifying the near nonbonded potential energy function. If it is *None*, then the switching function is applied directly to the original potential. Other options are *'shift'* and *'force-switch'*. If it is *'shift'*, then the switching function is applied to a potential that is already null at the cutoff due to a previous shift. If it is *'force-switch'*, then the potential is modified so that the switching function is applied to the forces rather than the potential energy.
- **`fastExceptions`** (*`bool, optional, default=True`*) – Whether nonbonded exceptions must be considered to belong to the group of fastest forces. If *False*, then they will be split into intermediate and slowest forces.

`redefine_bond`(*topology*, *residue*, *atom1*, *atom2*, *length*, *K=None*, *group=1*)
Changes the equilibrium length of a specified bond for integration within its original time scale. The difference between the original and the redefined bond potentials is evaluated at another time scale.

Parameters

- **topology** (*openmm.Topology*) – The topology corresponding to the original system.
- **residue** (*str*) – A name or regular expression to identify the residue which contains the redefined bond.
- **atom1** (*str*) – A name or regular expression to identify the first atom that makes the bond.
- **atom2** (*str*) – A name or regular expression to identify the second atom that makes the bond.

- **length** (*unit.Quantity*) – The redifined equilibrium length for integration at the shortest time scale.
- **K** (*unit.Quantity, optional, default=None*) – The harmonic force constant for the bond. If this is *None*, then the original value will be maintained.
- **group** (*int, optional, default=1*) – The force group with which the difference between the original and the redefined bond potentials must be evaluated.

redefine_angle(*topology*, *residue*, *atom1*, *atom2*, *atom3*, *angle*, *K=None*, *group=1*)

Changes the equilibrium value of a specified angle for integration within its original time scale. The difference between the original and the redefined angle potentials is evaluated at another time scale.

Parameters

- **topology** (*openmm.Topology*) – The topology corresponding to the original system.
- **residue** (*str*) – A name or regular expression to identify the residue which contains the redefined angle.
- **atom1** (*str*) – A name or regular expression to identify the first atom that makes the angle.
- **atom2** (*str*) – A name or regular expression to identify the second atom that makes the angle.
- **atom3** (*str*) – A name or regular expression to identify the third atom that makes the angle.
- **angle** (*unit.Quantity*) – The redifined equilibrium angle value for integration at the shortest time scale.
- **K** (*unit.Quantity, optional, default=None*) – The harmonic force constant for the angle. If this is *None*, then the original value will be maintained.
- **group** (*int, optional, default=1*) – The force group with which the difference between the original and the redefined angle potentials must be evaluated.

class atomsmm.systems.**SolvationSystem**(*system*, *solute_atoms*, *use_softcore=True*, *softcore_group=0*, *split_exceptions=False*)

Bases: `simtk.openmm.openmm.System`

An OpenMM System prepared for solvation free-energy calculations.

Parameters

- **system** (*openmm.System*) – The original system from which to generate the SolvationSystem.
- **solute_atoms** (*set(int)*) – A set containing the indexes of all solute atoms.
- **use_softcore** (*bool, optional, default=True*) – Whether to define a softcore potential for the coupling/decoupling of solute-solvent Lennard-Jones interactions. If this is *False*, then a linear scaling of both *sigma* and *epsilon* will be applied instead.
- **softcore_group** (*int, optional, default=0*) – The force group to be assigned to the solute-solvent softcore interactions, if any.
- **split_exceptions** (*bool, optional, default=False*) – Whether preexisting exceptions should be separated from the nonbonded force before new exceptions are created.

class atomsmm.systems.**AlchemicalSystem**(*system*, *atoms*, *coupling='softcore'*, *group=0*, *use_lrc=False*)

Bases: `simtk.openmm.openmm.System`

An OpenMM System prepared for solvation free-energy calculations.

Parameters

- **system** (*openmm.System*) – The original system from which to generate the SolvationSystem.
- **atoms** (*set(int)*) – A set containing the indexes of all solute atoms.
- **coupling** (*str, optional, default='softcore'*) – The model used for coupling the alchemical atoms to the system. The options are *softcore*, *linear*, *art*, *spline*, or some function of *lambda_vdw*. Use *softcore* for the model of Beutler et al. (1994), *linear* for a simple linear coupling, *art* for the sine-based coupling model of Abrams, Rosso, and Tuckerman (2006), and *spline* for multiplying the solute-solvent interactions by $\lambda_\text{vdw}^3(10 - 15\lambda_\text{vdw} + 6\lambda_\text{vdw}^2)$. Alternatively, you can enter any other valid function of *lambda_vdw*.
- **group** (*int, optional, default=0*) – The force group to be assigned to the solute-solvent softcore interactions, if any.
- **use_lrc** (*bool, optional, defaul=False*) – Whether to use long-range (dispersion) correction in solute-solvent interactions.

class atomsmm.systems.**AlchemicalRespaSystem**(*system, rcutIn, rswitchIn, alchemical_atoms=[], coupling_parameter='lambda', coupling_function='lambda', middle_scale=True, coulomb_scaling=False, lambda_coul=0, use_softcore=False, split_alchemical=True*)

Bases: `simtk.openmm.openmm.System`

An OpenMM System prepared for Multiple Time-Scale Integration with RESPA and for alchemical coupling/decoupling of specified atoms.

Short-range forces for integration at intermediate time scale are generated by applying a switching function to the force that results from the following potential:

$$V(r) = V^*_\text{LJC}(r) - V^*_\text{LJC}(r_\text{cut,in})$$
$$V^*_\text{LJC}(r) = \left\{4\epsilon\left[f_{12}(u(r))\left(\frac{\sigma}{r}\right)^{12} - f_6(u(r))\left(\frac{\sigma}{r}\right)^6\right] + \frac{f_1(u(r))}{4\pi\epsilon_0}\frac{q_1 q_2}{r}\right\}$$

where $f_n(u)$ is the solution of the 1st order differential equation

$$f_n - \frac{u+b}{n}\frac{df_n}{du} = S(u)$$
$$f_n(0) = 1$$
$$b = \frac{r_\text{switch,in}}{r_\text{cut,in} - r_\text{switch,in}}$$

As a consequence of this modification, $V'(r) = S(u(r))V'_\text{LJC}(r)$.

Examples of coupling function are:

1. Linear coupling (default):

$$f(\lambda) = \lambda$$

2. A 5-th order polinomial whose 1st- and 2nd-order derivatives are null at both extremes.

$$f(\lambda) = \lambda^3(10 - 15\lambda + 6\lambda^2)$$

3. A 5-th order polinomial whose 1st-order derivative is null at both extremes and whose 2nd- and 3rd-order derivatives are also null at $\lambda = 0$:

$$f(\lambda) = \lambda^4(5 - 4\lambda)$$

4. The sine-based coupling model of Abrams, Rosso, and Tuckerman [15]:

$$f(\lambda) = \lambda - \frac{\sin{(2\pi\lambda)}}{2\pi}$$

Parameters

- **system** (*openmm.System*) – The original system from which to generate the SolvationSystem.
- **rcutIn** (*unit.Quantity*) – The distance at which the short-range nonbonded interactions will completely vanish.
- **rswitchIn** (*unit.Quantity*) – The distance at which the short-range nonbonded interactions will start vanishing by application of a switching function.
- **alchemical_atoms** (*list(int), optional, default=[]*) – A set containing the indexes of all alchemical atoms.
- **coupling_parameter** (*str, optional, defaul='lambda'*) – The name of the coupling parameter.
- **coupling_function** (*str, optional, default='lambda'*) – A function $f(\lambda)$ used for coupling the alchemical atoms to the system, where λ is the coupling parameter. This must be a function of a single variable named as in argument *coupling_parameter* (see above). It is expected that $f(0) = 0$ and $f(1) = 1$.
- **middle_scale** (*bool, optional, default=True*) – Whether to use an intermediate time scale in the RESPA integration.
- **coulomb_scaling** (*bool, optional, default=False*) – Whether to consider scaling of electrostatic interactions between alchemical and non-alchemical atoms. Otherwise, these interactions will not exist.
- **lambda_coul** (*float, optional, default=0*) – A scaling factor to be applied to all electrostatic interactions between alchemical and non-alchemical atoms.

`reset_coulomb_scaling_factor`(*lambda_coul*, *context=None*)

Resets the scaling factor of the solute-solvent electrostatic interactions.

Parameters

- **lambda_coul** (*float*) – The scaling factor value.
- **context** (*Context_, optional, default=None*) – A context in which the particle parameters should be updated.

`class` `atomsmm.systems.`**`ComputingSystem`**(*system*)

Bases: `atomsmm.systems._AtomsMM_System`

An OpenMM System prepared for computing the Coulomb contribution to the potential energy, as well as the total internal virial of an atomic system.

..warning: Currently, virial computation is only supported for fully flexible systems (i.e. without distance constraints).

Parameters **system** (*openmm.System*) – The original system from which to generate the ComputingSystem.

4.7 utils

exception `atomsmm.utils.`**`InputError`**(*msg*)

`atomsmm.utils.`**`countDegreesOfFreedom`**(*system*)

Counts the number of degrees of freedom in a system, given by:

$$N_{\text{DOF}} = 3N_{\text{movingparticles}} - 3 - N_{\text{constraints}}$$

Parameters **system** (*openmm.System*) – The system whose degrees of freedom will be summed up.

`atomsmm.utils.`**`findNonbondedForce`**(*system*, *position=0*)

Searches for a NonbondedForce object in an OpenMM system.

Parameters

- **system** (*openmm.System*) – The system to which the wanted NonbondedForce object is attached.
- **position** (*int, optional, default=0*) – The position index of the wanted force among the NonbondedForce objects attached to the system.

Returns *int* – The index of the wanted NonbondedForce object.

`atomsmm.utils.`**`hijackForce`**(*system*, *index*)

Extracts a Force object from an OpenMM system.

Warning: Side-effect: the passed system object will no longer have the hijacked Force object in its force list.

Parameters **index** (*int*) – The index of the Force object to be hijacked.

Returns *openmm.Force* – The hijacked Force object.

`atomsmm.utils.`**`splitPotentialEnergy`**(*system*, *topology*, *positions*, ***globals*)

Computes the potential energy of a system, with possible splitting into contributions of all Force objects attached to the system.

Parameters

- **system** (*openmm.System*) – The system whose energy is to be computed.
- **topology** (*openmm.app.topology.Topology*) – The topological information about a system.
- **positions** (*list(tuple)*) – A list of 3D vectors containing the positions of all atoms.

Keyword Arguments **`for global context variables.`** (`Values`) –

Returns *unit.Quantity or dict(str, unit.Quantity)* – The total potential energy or a dict containing all potential energy terms.

`atomsmm.utils.`**`evaluateForce`**(*force*, *positions*, *boxVectors=None*)

Computes the value of a Force object for a given set of particle coordinates and box vectors. Whether periodic boundary conditions will be used or not depends on the corresponding attribute of the Force object specified as the collective variable.

Parameters

- **positions** (*list(openmm.Vec3)*) – A list whose length equals the number of particles in the system and which contains the coordinates of these particles.

- **boxVectors** (*list(openmm.Vec3), optional, default=None*) – A list with three vectors which describe the edges of the simulation box.

Returns *unit.Quantity*

Example

```
>>> import afed
>>> from simtk import unit
>>> model = afed.AlanineDipeptideModel()
>>> psi_angle, _ = model.getDihedralAngles()
>>> psi = afed.DrivenCollectiveVariable('psi', psi_angle, unit.radians,␣
↪period=360*unit.degrees)
>>> psi.evaluate(model.getPositions())
Quantity(value=3.141592653589793, unit=radian)
```

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

5.1 Bug reports

When reporting a bug please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.2 Documentation improvements

AtomsMM could always use more documentation, whether as part of the official AtomsMM docs, in docstrings, or even on the web in blog posts, articles, and such.

5.3 Feature requests and feedback

The best way to send feedback is to file an issue at https://github.com/atoms-ufrj/atomsmm/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

5.4 Development

Prerequisites:

1. Anaconda or Miniconda
2. Conda virtualenv
3. Tox

To set up *atomsmm* for local development:

1. Fork atomsmm (look for the "Fork" button).
2. Clone your fork locally:

```
git clone git@github.com:your_name_here/atomsmm.git
```

3. Create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. When you're done making changes, run all the checks, doc builder and spell checker with tox one command:

```
tox
```

5. Commit your changes and push your branch to GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

6. Submit a pull request through the GitHub website.

5.4.1 Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make the pull request.

For merging, you should:

1. Include passing tests (run `tox`)[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.

It will be slower though …

5.4.2 Tips

To run a subset of tests:

```
tox -e envname -- py.test -k test_myfeature
```

To run all the test environments in *parallel* (you need to `pip install detox`):

```
detox
```

CHAPTER SIX

AUTHORS

- Charlles R. A. Abreu - http://atoms.peq.coppe.ufrj.br

CHAPTER

SEVEN

CHANGELOG

CHAPTER

EIGHT

GLOSSARY

ODE	Ordinary Differential Equation
SDE	Stochastic Differential Equation

CHAPTER

NINE

BIBLIOGRAPHY

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[1] Philippe H. Hünenberger. Calculation of the group-based pressure in molecular simulations. i. a general formulation including ewald and particle-particle–particle-mesh electrostatics. *The Journal of Chemical Physics*, 116(16):6880–6897, apr 2002. doi:10.1063/1.1463057.

[2] Ruhong Zhou, Edward Harder, Huafeng Xu, and B. J. Berne. Efficient multiple time step method for use with ewald and particle mesh ewald for large biomolecular systems. *The Journal of Chemical Physics*, 115(5):2348–2358, aug 2001. doi:10.1063/1.1385159.

[3] Joseph A. Morrone, Ruhong Zhou, and B. J. Berne. Molecular dynamics with multiple time scales: how to avoid pitfalls. *Journal of Chemical Theory and Computation*, 6(6):1798–1804, jun 2010. doi:10.1021/ct100054k.

[4] Ben Leimkuhler, Daniel T. Margul, and Mark E. Tuckerman. Stochastic, resonance-free multiple time-step algorithm for molecular dynamics with very large time steps. *Molecular Physics*, 111(22-23):3579–3594, dec 2013. doi:10.1080/00268976.2013.844369.

[5] Masuo Suzuki. Generalized trotter's formula and systematic approximants of exponential operators and inner derivations with applications to many-body problems. *Communications in Mathematical Physics*, 51(2):183–190, jun 1976. doi:10.1007/bf01609348.

[6] Masuo Suzuki. Decomposition formulas of exponential operators and lie exponentials with some applications to quantum mechanics and statistical physics. *Journal of Mathematical Physics*, 26(4):601–612, apr 1985. doi:10.1063/1.526596.

[7] Haruo Yoshida. Construction of higher order symplectic integrators. *Physics Letters A*, 150(5-7):262–268, nov 1990. doi:10.1016/0375-9601(90)90092-3.

[8] Masuo Suzuki. General theory of fractal path integrals with applications to many-body theories and statistical physics. *Journal of Mathematical Physics*, 32(2):400–407, feb 1991. doi:10.1063/1.529425.

[9] M. Tuckerman, B. J. Berne, and G. J. Martyna. Reversible multiple time scale molecular dynamics. *The Journal of Chemical Physics*, 97(3):1990–2001, aug 1992. doi:10.1063/1.463137.

[10] Giovanni Bussi, Davide Donadio, and Michele Parrinello. Canonical sampling through velocity rescaling. *The Journal of Chemical Physics*, 126(1):014101, jan 2007. doi:10.1063/1.2408420.

[11] Giovanni Bussi and Michele Parrinello. Stochastic thermostats: comparison of local and global schemes. *Computer Physics Communications*, 179(1-3):26–29, jul 2008. doi:10.1016/j.cpc.2008.01.006.

[12] George Marsaglia and Wai Wan Tsang. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software*, 26(3):363–372, sep 2000. doi:10.1145/358407.358414.

[13] Alex A. Samoletov, Carl P. Dettmann, and Mark A. J. Chaplain. Thermostats for \textquotedblleft slow\textquotedblright configurational modes. *Journal of Statistical Physics*, 128(6):1321–1336, jul 2007. doi:10.1007/s10955-007-9365-2.

[14] Ben Leimkuhler, Emad Noorizadeh, and Florian Theil. A gentle stochastic thermostat for molecular dynamics. *Journal of Statistical Physics*, 135(2):261–277, apr 2009. doi:10.1007/s10955-009-9734-0.

[15] Jerry B. Abrams, Lula Rosso, and Mark E. Tuckerman. Efficient and precise solvation free energies via alchemical adiabatic molecular dynamics. *The Journal of Chemical Physics*, 125(7):074115, August 2006. doi:10.1063/1.2232082.

PYTHON MODULE INDEX

a

atomsmm.computers, 7
atomsmm.forces, 8
atomsmm.integrators, 12
atomsmm.propagators, 18
atomsmm.reporters, 35
atomsmm.systems, 39
atomsmm.utils, 43

c

computers (*Unix, Windows*), 7

f

forces (*Unix, Windows*), 8

i

integrators (*Unix, Windows*), 12

p

propagators (*Unix, Windows*), 18

r

reporters (*Unix, Windows*), 35

s

system (*Unix, Windows*), 39

u

utils (*Unix, Windows*), 43

INDEX

A

AdiabaticDynamicsIntegrator (*class in atomsmm.integrators*), 17
AlchemicalRespaSystem (*class in atomsmm.systems*), 41
AlchemicalSystem (*class in atomsmm.systems*), 40
atomsmm.computers (*module*), 7
atomsmm.forces (*module*), 8
atomsmm.integrators (*module*), 12
atomsmm.propagators (*module*), 18
atomsmm.reporters (*module*), 35
atomsmm.systems (*module*), 39
atomsmm.utils (*module*), 43

C

CenterOfMassReporter (*class in atomsmm.reporters*), 38
ChainedPropagator (*class in atomsmm.propagators*), 18
computers (*module*), 7
ComputingSystem (*class in atomsmm.systems*), 42
countDegreesOfFreedom() (*in module atomsmm.utils*), 43
CustomIntegratorReporter (*class in atomsmm.reporters*), 38

D

DampedSmoothedForce (*class in atomsmm.forces*), 8

E

evaluateForce() (*in module atomsmm.utils*), 43
ExpandedEnsembleReporter (*class in atomsmm.reporters*), 38
ExtendedStateDataReporter (*class in atomsmm.reporters*), 35
ExtendedSystemPropagator (*class in atomsmm.propagators*), 35
ExtendedSystemVariable (*class in atomsmm.integrators*), 17

F

FarNonbondedForce (*class in atomsmm.forces*), 10
findNonbondedForce() (*in module atomsmm.utils*), 43
forces (*module*), 8

G

GenericBoostPropagator (*class in atomsmm.propagators*), 23
GenericScalingPropagator (*class in atomsmm.propagators*), 23
get_atomic_pressure() (*atomsmm.computers.PressureComputer method*), 7
get_atomic_virial() (*atomsmm.computers.PressureComputer method*), 7
get_bond_virial() (*atomsmm.computers.PressureComputer method*), 8
get_coulomb_virial() (*atomsmm.computers.PressureComputer method*), 8
get_dispersion_virial() (*atomsmm.computers.PressureComputer method*), 8
get_molecular_pressure() (*atomsmm.computers.PressureComputer method*), 8
get_molecular_virial() (*atomsmm.computers.PressureComputer method*), 8
GlobalThermostatIntegrator (*class in atomsmm.integrators*), 12

H

hijackForce() (*in module atomsmm.utils*), 43

I

initialize() (*atomsmm.integrators.AdiabaticDynamicsIntegrator method*), 18
initialize() (*atomsmm.integrators.LimitedSpeedBAOABIntegrator*

method), 16
initialize() (*atomsmm.integrators.LimitedSpeedNHLIntegrator method*), 16
initialize() (*atomsmm.integrators.LimitedSpeedStochasticIntegrator method*), 16
initialize() (*atomsmm.integrators.LimitedSpeedStochasticVelocityIntegrator method*), 17
initialize() (*atomsmm.integrators.NewMethodIntegrator method*), 15
initialize() (*atomsmm.integrators.NHL_R_Integrator method*), 13
initialize() (*atomsmm.integrators.SIN_R_Integrator method*), 15
InputError, 43
integrator() (*atomsmm.propagators.Propagator method*), 18
integrators (*module*), 12

L

Langevin_R_Integrator (*class in atomsmm.integrators*), 13
LimitedSpeedBAOABIntegrator (*class in atomsmm.integrators*), 15
LimitedSpeedLangevinPropagator (*class in atomsmm.propagators*), 21
LimitedSpeedNHLIntegrator (*class in atomsmm.integrators*), 16
LimitedSpeedNHLPropagator (*class in atomsmm.propagators*), 21
LimitedSpeedStochasticIntegrator (*class in atomsmm.integrators*), 16
LimitedSpeedStochasticPropagator (*class in atomsmm.propagators*), 22
LimitedSpeedStochasticVelocityIntegrator (*class in atomsmm.integrators*), 16
LimitedSpeedStochasticVelocityPropagator (*class in atomsmm.propagators*), 22

M

MassiveGeneralizedGaussianMomentPropagator (*class in atomsmm.propagators*), 27
MassiveIsokineticPropagator (*class in atomsmm.propagators*), 20
MassiveNoseHooverPropagator (*class in atomsmm.propagators*), 27
MultipleTimeScaleIntegrator (*class in atomsmm.integrators*), 12
MultipleTimeScalePropagator (*class in atomsmm.propagators*), 24

N

NearExceptionForce (*class in atomsmm.forces*), 10
NearNonbondedForce (*class in atomsmm.forces*), 9
NewMethodIntegrator (*class in atomsmm.integrators*), 15
NewMethodPropagator (*class in atomsmm.propagators*), 21
NHL_R_Integrator (*class in atomsmm.integrators*), 13
NonbondedExceptionsForce (*class in atomsmm.forces*), 8
NoseHooverChainPropagator (*class in atomsmm.propagators*), 27
NoseHooverLangevinPropagator (*class in atomsmm.propagators*), 28
NoseHooverPropagator (*class in atomsmm.propagators*), 27

O

ODE, **53**
OrnsteinUhlenbeckPropagator (*class in atomsmm.propagators*), 22

P

PressureComputer (*class in atomsmm.computers*), 7
Propagator (*class in atomsmm.propagators*), 18
propagators (*module*), 18

R

redefine_angle() (*atomsmm.systems.RESPASystem method*), 40
redefine_bond() (*atomsmm.systems.RESPASystem method*), 39
RegulatedAtomicNoseHooverLangevinPropagator (*class in atomsmm.propagators*), 32
RegulatedBoostPropagator (*class in atomsmm.propagators*), 29
RegulatedMassiveNoseHooverLangevinPropagator (*class in atomsmm.propagators*), 30
RegulatedTranslationPropagator (*class in atomsmm.propagators*), 29
reporters (*module*), 35
reset_coulomb_scaling_factor() (*atomsmm.systems.AlchemicalRespaSystem method*), 42
RespaPropagator (*class in atomsmm.propagators*), 23
RESPASystem (*class in atomsmm.systems*), 39
RestrainedLangevinPropagator (*class in atomsmm.propagators*), 21

S

SDE, **53**
SIN_R_Integrator (*class in atomsmm.integrators*), 14
SIN_R_Propagator (*class in atomsmm.propagators*), 25
SoftcoreForce (*class in atomsmm.forces*), 11
SoftcoreLennardJonesForce (*class in atomsmm.forces*), 11
SolvationSystem (*class in atomsmm.systems*), 40
splitPotentialEnergy() (*in module atomsmm.utils*), 43
SplitPropagator (*class in atomsmm.propagators*), 18
state_sampling_analysis() (*atomsmm.reporters.ExpandedEnsembleReporter method*), 39
SuzukiYoshidaPropagator (*class in atomsmm.propagators*), 19
system (*module*), 39

T

TranslationPropagator (*class in atomsmm.propagators*), 19
TrotterSuzukiPropagator (*class in atomsmm.propagators*), 19
TwiceRegulatedAtomicNoseHooverLangevinPropagator (*class in atomsmm.propagators*), 33
TwiceRegulatedGlobalNoseHooverLangevinPropagator (*class in atomsmm.propagators*), 34
TwiceRegulatedMassiveNoseHooverLangevinPropagator (*class in atomsmm.propagators*), 31

U

UnconstrainedVelocityVerletPropagator (*class in atomsmm.propagators*), 26
utils (*module*), 43

V

VelocityBoostPropagator (*class in atomsmm.propagators*), 19
VelocityRescalingPropagator (*class in atomsmm.propagators*), 26
VelocityVerletPropagator (*class in atomsmm.propagators*), 26

X

XYZReporter (*class in atomsmm.reporters*), 38