pygamd Documentation

Release v1

You-Liang Zhu

Apr 17, 2024

GENERAL INTRODUCTION

1 Introduction **1**

2 Installation **3**
2.1 Source code . . . 3

3 Usage **5**

4 Units **7**
4.1 Fundamental Units . . . 7
4.2 Temperature units (thermal energy) . . . 7
4.3 Charge units . . . 7
4.4 Common derived units . . . 7
4.5 Example physical units . . . 8

5 System info **9**
5.1 Data format . . . 9
5.1.1 MST format . . . 9
5.2 Data input . . . 14
5.2.1 MST reader . . . 14
5.3 Data output . . . 15
5.3.1 Collective information . . . 15
5.3.2 MST dump . . . 15
5.3.3 XML dump . . . 15
5.4 Group . . . 16

6 Application **17**
6.1 Modules management . . . 17
6.2 Multi-stage simulation . . . 17
6.3 Two-dimensional simulation . . . 18

7 Force field **21**
7.1 Non-bonded interactions . . . 21
7.1.1 Non-bonded functions . . . 21
7.1.2 Self-defined functions . . . 22
7.1.3 Non-bonded functions for charged beads . . . 23
7.1.4 Self-defined functions for charged beads . . . 24
7.2 Bonded interactions . . . 26
7.2.1 Bond interactions . . . 26
7.2.2 Angle bending . . . 28
7.2.3 Dihedral torsion . . . 30

8 Integration **33**
8.1 NVT ensemble . . . 33
8.1.1 NVT . . . 33
8.1.2 GWVV . . . 34
8.1.3 BD . . . 34

9 Modules **35**
9.1 Dissipative particle dynamics . . . 35
9.1.1 DPD force . . . 35
9.1.2 GWVV integration . . . 36

10 Data **37**
10.1 Data format . . . 37
10.1.1 XML format . . . 37
10.2 Data input . . . 41
10.2.1 XML reader . . . 41
10.2.2 Binary reader . . . 41
10.3 Data output . . . 42
10.3.1 Common functions for Data output . . . 42
10.3.2 Collective information . . . 42
10.3.3 MOL2 dump . . . 43
10.3.4 XML dump . . . 43
10.3.5 DCD trajectory dump . . . 46
10.3.6 Binary dump . . . 46

11 System **49**
11.1 Import libraries . . . 49
11.1.1 import cuda lib (Linux) . . . 49
11.1.2 import hip lib (Linux) . . . 49
11.1.3 import cuda lib (Windows) . . . 49
11.2 Information . . . 49
11.2.1 Perform configuration . . . 49
11.2.2 All information . . . 50
11.3 Particle list . . . 50
11.3.1 Neighbor list . . . 50
11.3.2 Cell list . . . 51
11.4 Particle set . . . 52
11.5 Dynamic Particle Set . . . 53
11.6 Information computation . . . 54
11.7 Memory sorting . . . 54

12 Application **55**
12.1 Modules management . . . 55
12.2 Multi-stage simulation . . . 56
12.3 Two-dimensional simulation . . . 56

13 Force field **59**
13.1 Short range non-bonded interactions . . . 59
13.1.1 Lennard-Jones (LJ) interaction . . . 59
13.1.2 Shift Lennard-Jones (LJ) interaction . . . 60
13.1.3 Linear molecule π-π interaction . . . 61
13.1.4 Generalized exponential model . . . 62
13.1.5 Lennard-Jones and Ewald (short range) interaction . . . 63
13.1.6 Pair interaction . . . 64
13.2 Bonded interactions . . . 66

13.2.1 Bond stretching . . . 66
13.2.2 Angle bending . . . 70
13.2.3 Dihedral torsion . . . 73
13.3 Numerical interaction . . . 77
13.3.1 Theory description . . . 77
13.3.2 Non-bonded interaction . . . 78
13.3.3 Bond interaction . . . 78
13.3.4 Angle interaction . . . 79
13.3.5 Dihedral interaction . . . 79
13.3.6 Self-defined functions . . . 80
13.4 Coulomb interaction . . . 81
13.4.1 Ewald summation theory . . . 81
13.4.2 Ewald (short-range) . . . 82
13.4.3 Ewald for DPD (short-range) . . . 83
13.4.4 PPPM (long-range) . . . 84
13.4.5 ENUF (long-range) . . . 85
13.5 Read force parameters . . . 85
13.5.1 Force field format . . . 85
13.5.2 Use force fields . . . 87
13.5.3 Use GROMACS force fields . . . 90
13.5.4 Convert GROMACS files . . . 90

14 Integration **91**
14.1 NVE ensemble . . . 91
14.1.1 NVE thermostat . . . 91
14.1.2 NVE for rigid body . . . 91
14.1.3 NVE for rigid body with tunable freedoms . . . 92
14.2 NVT ensemble . . . 92
14.2.1 Nose Hoover thermostat . . . 92
14.2.2 Berendsen thermostat . . . 93
14.2.3 Andersen thermostat . . . 93
14.2.4 Langevin dynamic thermostat . . . 94
14.2.5 NVT for rigid body . . . 95
14.2.6 Langevin dynamic for rigid body . . . 95
14.3 NPT ensemble . . . 96
14.3.1 Andersen barostat . . . 96
14.3.2 NPT for rigid body . . . 97
14.3.3 Martyna-Tobias-Klein barostat . . . 97
14.3.4 Martyna-Tobias-Klein barostat for rigid body . . . 98

15 Constraint **101**
15.1 Variant . . . 101
15.1.1 Variant Const . . . 101
15.1.2 Variant Linear . . . 101
15.1.3 Variant Sin . . . 102
15.1.4 Variant Well . . . 102
15.2 Space constraint . . . 102
15.2.1 Bounce back condition . . . 102
15.2.2 LJ surface force . . . 103
15.3 Remove CM momentum . . . 104
15.4 Constant chemical potential . . . 104
15.5 Bond constraint . . . 105
15.6 Virtual site . . . 106

16 External field **107**
16.1 External force . . . 107
16.2 Axial stretching . . . 108
16.3 RNEMD . . . 108

17 Modules **111**
17.1 MD-SCF . . . 111
17.1.1 MDSCF force . . . 111
17.1.2 MDSCF electrostatic force . . . 112
17.2 Polymerization . . . 112
17.2.1 Polymerization model . . . 112
17.2.2 Depolymerization model . . . 114
17.3 Anisotropic particle . . . 115
17.3.1 Gay-Berne model . . . 115
17.3.2 Soft anisotropic model . . . 117
17.4 Dissipative particle dynamics . . . 120
17.4.1 DPD force . . . 120
17.4.2 GWVV integration . . . 121
17.4.3 Coulomb interaction in DPD . . . 122
17.5 Particle type change . . . 123
17.5.1 ChangeType . . . 123

18 molgen **125**
18.1 Description of molgen . . . 125
18.2 Molecule definition . . . 126
18.3 Objects definition . . . 129
18.4 Generator definition . . . 130

19 dataTackle **133**
19.1 Load dataTackle . . . 133
19.2 Usage . . . 133
19.3 Functions . . . 134
19.3.1 1 Rg^2: . . . 134
19.3.2 2 Ed^2: . . . 134
19.3.3 3 RDF: . . . 135
19.3.4 4 bond_distri: . . . 135
19.3.5 5 angle_distri: . . . 135
19.3.6 6 dihedral_distri: . . . 135
19.3.7 7 stress tensor: . . . 136
19.3.8 8 density: . . . 136
19.3.9 9 unwrapping: . . . 136
19.3.10 10 MSD: . . . 136
19.3.11 11 RDF-CM: . . . 137
19.3.12 12 MSD-CM: . . . 137
19.3.13 13 ents: . . . 137
19.3.14 14 strfac: . . . 137
19.3.15 15 domain size: . . . 137
19.3.16 16 dynamic strfac: . . . 137
19.3.17 17 config check: . . . 138
19.3.18 18 RDF between types: . . . 138
19.3.19 19 MST conversion: . . . 138

20 License **139**

21 Indices and tables **141**

Index 143

CHAPTER

ONE

INTRODUCTION

Molecular dynamics (MD) simulations are exceptionally important in the research field of polymers, soft matters, biomolecules, and etc. In general, all-atom or coarse-grained force fields are not easily ported between MD packages. It is quit difficult to realize a new method in a well-developed MD package by users. Thereby, we provide a MD simulation package named PYGAMD (Python GPU-Accelerated Molecular Dynamics Software) to solve these problems.

PYGAMD is a platform where users could build up their customized force fields including potential forms and parameters. This is achieved by that PYGAMD is programmed based on Numba, a Just-In-Time Python Compiler. The potential forms and methods could be conveyed from user interface to the underlying computation. More important, PYGAMD provide a high performance on GPU computation up to traditional packages programmed by CUDA and C.

This package is the version 1 of PYGAMD which includes MD engine **pygamd**, molecular configuration generator **molgen**, and data tackler **dataTackle** etc. The **pygamd** is purely written by Python language based on Python3 Numba compiler. The plugins **molgen** and **dataTackle** that are written by C++ and CUDA C, need to be compiled.

CHAPTER

TWO

INSTALLATION

2.1 Source code

The entire PYGAMD package is a Free Software under the GNU General Public License. The **pygamd** engine, **molgen** plugin, and **dataTackle** plugin could be separately installed. The installation instructions for **pygamd**, **molgen**, and **dataTackle** are as follows.

1. Installation for **pygamd**:

```
Requrements:

1. Python3 including numba, numpy, cupy, and pybind11 packages
2. NVIDIA CUDA Toolkit >= 7.0

Installation:

python3 setup.py install
```

2. Installation for PIP:

```
pip install pygamd
python3 setup.py install
```

CHAPTER

THREE

USAGE

With a prepared script, you could run pygamd MD engine for obtaining trajectory.

Examples:

```
python3 yourscript.py --gpu=0 >a.log&
```

Where you could specify the GPU id (default value is 0) with the `--gpu=` option and output the screen information into `a.log` file.

Here is an example of script for DPD simulation.

Firstly, importing the pygamd module installed as a package of python3 and reading system information by *snapshot.read* from a mst file

Examples:

```
import pygamd
mst = pygamd.snapshot.read("AB.mst")
```

After that, we need to build up an application by *application.dynamics* which will call defined and added objects.

Examples:

```
app = pygamd.application.dynamics(info=mst, dt=0.04)
```

Further, we should define objects by the classes of pygamd and pass them to the application, such as the following example: DPD force *force.dpd* , NVT thermostat with GWVV algorithm *integration.gwvv*, and th dump of system collective information *dump.data*.

Examples:

```
fn = pygamd.force.dpd(info=mst, rcut=1.0)
fn.setParams(type_i="A", type_j="A", alpha=25.0, sigma=3.0)
fn.setParams(type_i="A", type_j="B", alpha=40.0, sigma=3.0)
fn.setParams(type_i="B", type_j="B", alpha=25.0, sigma=3.0)
app.add(fn)


gw = pygamd.integration.gwvv(info=mst, group='all')
app.add(gw)

di = pygamd.dump.data(info=mst, group='all', file='data.log', period=500)
app.add(di)
```

Finally, running the simulation with the number of time steps.

Examples:

```
app.run(10000)
```

CHAPTER

FOUR

UNITS

PYGAMD stores and computes all values in reduced units. The quantities in real units can be converted into the ones in reduced units by defining a set of fundamental units by user himself.

4.1 Fundamental Units

The three fundamental units are:

- distance - σ
- energy - ε
- mass - $\Updownarrow$

4.2 Temperature units (thermal energy)

PYGAMD accepts all temperature inputs and provides all temperature output values in units of energy: $k_B T$, where k_B is Boltzmann's constant. In reduced units, one usually reports the value $T^* = k_B T/\varepsilon$.

4.3 Charge units

The charge used in PYGAMD is also reduced. The units of charge are: $(4\pi\epsilon_0\epsilon_r\sigma\varepsilon)^{1/2}$, where ϵ_0 is vacuum permittivity and ϵ_r is relative permittivity.

With $f = 1/4\pi\epsilon_0 = 138.935\ kJ\ mol^{-1}\ nm\ e^{-2}$, the units of charge are: $(\epsilon_r\sigma\varepsilon/f)^{1/2}$. Divide a given charge by this quantity to convert it into an input value for PYGAMD.

4.4 Common derived units

Here are some commonly used derived units:

- time - $\tau = \sqrt{\Updownarrow\sigma^2/\varepsilon}$
- volume - σ^3
- velocity - σ/τ
- momentum - $\Updownarrow\sigma/\tau$
- acceleration - σ/τ^2

- force - ε/σ
- pressure - ε/σ^3

4.5 Example physical units

There are many possible choices of physical units that one can assign. One common choice is:

- distance - $\sigma = \mathrm{nm}$
- energy - $\varepsilon = \mathrm{kJ/mol}$
- mass - $\Updownarrow = \mathrm{amu}$

Derived units / values in this system:

- time - picoseconds
- velocity - nm/picosecond
- pressure - 16.3882449645417 atm
- force - 1.66053892103218 pN
- k_B = 0.00831445986144858 kJ/mol/Kelvin

CHAPTER

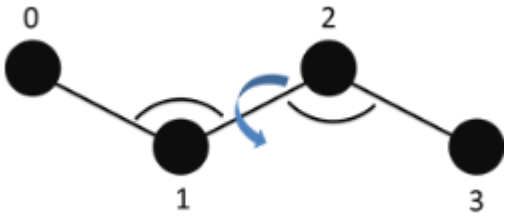
FIVE

SYSTEM INFO

5.1 Data format

5.1.1 MST format

We take MST format files as the standard input and output configuration files. The MST files can contain coordinates, types, masses, velocities, bond connections, angles, dihedrals and so on. Here is an example of the MST file of a single molecule system. The molecule consisting of four particles is depicted in following picture.



The data in a line of MST file corresponds to a particle and all particles are given in sequence. For example, the coordinate of a particle in x, y, and z directions is written in a line and three columns in MST files. However, this rule does not include topological relevant information, including bonds, angles and dihedrals.

Snapshot file

An example of MST snapshot file with particles coordinates, velocities, types, masses ...

```
mst_version 1.0
        num_particles
                4
        timestep
                0
        dimension
                3
        box
                10.0    10.0    10.0
        position
                -1  2 -1
                -2  3  0
                -1  4  1
                -1  5  2
```

(continues on next page)

(continued from previous page)

```
        velocity
                1  2  3
                1  0  0
                3 -2  1
                0  1  1
        type
                A
                B
                B
                A
        mass
                1.0
                2.1
                1.0
                1.0
```

The file could include more information, such as bond, angle, dihedral ...

```
# bond with 'bond type (string), particle index i (int), j (int)'.
            bond
                    polymer 0 1
                    polymer 1 2
                    polymer 2 3

# angle with 'angle type (string), particle index i (int), j (int), k (int)'.
            angle
                    theta 0 1 2
                    theta 1 2 3

# dihedral with 'dihedral type (string), particle index i (int), j (int), k␣
↪(int), l (int)'.
            dihedral
                    phi 0 1 2 3

# virial site with 'vsite type (string), particle index i (int), j (int), k␣
↪(int), l (int)'.
            vsite
                    v 3 0 1 2

# the diameter of particles with float type.
            diameter
                    1.0
                    1.0
                    1.0
                    1.0

# the charge of particles with float type.
            charge
                     1.333
                     1.333
                    -1.333
                    -1.333
```

(continues on next page)

(continued from previous page)

```
# the body index of particles with int type, -1 for non-body particles.
           body
                   -1
                   -1
                    0
                    0

# the image in x, y, and z directions of particles with int type.
           image
                   0 0 0
                   0 0 0
                   0 0 0
                   0 0 0

# the velocity in x, y, and z directions of particles with float type.
           velocity
                    3.768     -2.595    -1.874
                   -3.988     -1.148     2.800
                    1.570      1.015    -3.167
                    2.441     -1.859    -1.039


# the orientation vector (x, y, z) of particles with float type.
           orientation
                   -0.922     0.085     0.376
                   -0.411    -0.637    -0.651
                    0.293     0.892    -0.342
                   -0.223     0.084     0.970

# the quaternion vector (x, y, z, w) of particles with float type.
           quaternion
                    0.369   0.817  -0.143   0.418
                   -0.516  -0.552   0.653   0.024
                   -0.521  -0.002   0.131   0.843
                   -0.640   0.159  -0.048  -0.749

# the angular velocity of rotation in x, y, and z directions of particles␣
↪with float type.
           rotation
                   -0.640    0.571   -0.512
                   -0.744    0.346    0.569
                    0.620   -0.086    0.779
                   -0.542    0.319   -0.776

# the moment of inertia in x, y, and z directions of particles with float type.
           inert
                   1.0 1.0 3.0
                   1.0 1.0 3.0
                   1.0 1.0 3.0
                   1.0 1.0 3.0
```

(continues on next page)

(continued from previous page)

```
# the rotated angles of in x, y, and z directions of particles with float type.
            rotangle
                    9.478    -1.677    8.239
                    8.908    -1.214    8.086
                    9.011    -0.653    7.600
                    8.993    -0.488    8.331

# the initiator indication of particles with int type, 1 for initiator.
            init
                    0
                    1
                    0
                    1

# the crosslinking number of particles with int type, 0 for reactable monomer.
            cris
                    0
                    0
                    0
                    0

# the molecule index of particles with int type, -1 for free particles.
            molecule
                    0
                    0
                    1
                    1
```

The attribute of anisotropic particles …

```
# the particle patch attribute with 'particle type (string), patch number (int)'
# followd by 'patch type(string), patch size (float),
# patch position vector in x, y, z directions (float)'.
            patch
                    B 2
                    p1 60  0    0    1
                    p1 60  0    0   -1

# the patch-patch interaction parameter with 'patch type (string), patch type␣
↪(string),
# gamma_epsilon (float), alpha (float)'.
            patch_param
                    p1 p1 88.0 0.5

# the particle shape attribute with 'particle type(string), diameter a,␣
↪diameter b, diameter c,
# epsion a, epsion b, epsion c (float)'. The a, b, c are along x, y, z␣
↪directions in body frame,
# respectively.
            asphere
                    A 1.0 1.0 1.0 3.0 3.0 3.0
                    B 1.0 1.0 3.0 1.0 1.0 0.2
```

(continues on next page)

(continued from previous page)

```
# the end of file.
    mst_end
```

Trajectory file

A MST trajectory file could contain multiple frames. The properties in trajectory file are divied into two classes, i.e. invariant data and variant data. The invarant data is only output once, whereas the variant data is output every frame.

An example of MST trajectory file:

```
mst_version 1.0
invariant_data
        num_particles
                4
        dimension
                3
        box
                10.0    10.00   10.0
        bond
                polymer 0 1
                polymer 1 2
                polymer 2 3
        angle
                theta 0 1 2
                theta 1 2 3
        dihedral
                phi 0 1 2 3
        type
                A
                B
                B
                A
variant_data
frame   0
        timestep
                0
        position
                0       0       0
                1       0       0
                2       0       0
                3       0       0
        image
                0       0       0
                0       0       0
                0       0       0
                0       0       0
frame_end
frame   1
        timestep
```

(continues on next page)

(continued from previous page)

```
                10000
        position
                0       1       0
                1       1       0
                2       1       0
                3       1       0
        image
                0       0       0
                0       0       0
                0       0       0
                0       0       0
frame_end
frame   2
        timestep
                20000
        position
                0       2       0
                1       2       0
                2       2       0
                3       2       0
        image
                0       0       0
                0       0       0
                0       0       0
                0       0       0
frame_end
```

5.2 Data input

class snapshot

The module of *snapshot.read*.

5.2.1 MST reader

class snapshot.**read**(*filename*)

The constructor of MST file parser object.

Parameters

filename (*str*) – The file name of MST file.

Example:

```
mst = pygamd.snapshot.read("lj.mst")
# builds up a parser object for the input MST file.
```

5.3 Data output

5.3.1 Collective information

class dump.**data**(*info*, *group*, *file*, *period*)

Constructor of an information dump object for a group of particles.

Parameters

- **info** – system information.
- **group** – a group of particles.
- **file** – the name of output file.
- **period** – the period of data output.

Example:

```
dd = pygamd.dump.data(info=mst, group=['a'], file='data.log', period=100)
app.add(dd)
```

5.3.2 MST dump

class dump.**mst**(*info*, *group*, *file*, *period*, *properties=None*, *split=False*)

Constructor of an object to dump MST files.

Parameters

- **info** – system information.
- **group** – a group of particles.
- **file** – the name of output file.
- **period** – the period of data output.
- **properties** – the properties for output, candidates are 'position', 'type', 'velocity', 'mass', 'image', 'force', 'potential', 'virial', 'bond', 'angle', 'dihedral'
- **split** – if the trajectory file is splited into separated snapshot files

Example:

```
dm = pygamd.dump.mst(info=mst, group='all', file='p.mst', period=100000)
app.add(dm)
```

5.3.3 XML dump

class dump.**xml**(*info*, *group*, *file*, *period*, *properties=None*, *split=False*)

Constructor of an object to dump XML files.

Parameters

- **info** – system information.
- **group** – a group of particles.
- **file** – the name of output file.

- **period** – the period of data output.
- **properties** – the properties for output, candidates are 'position', 'type', 'velocity', 'mass', 'image', 'force', 'potential', 'virial', 'bond', 'angle', 'dihedral'
- **split** – if the trajectory file is splited into separated snapshot files

Example:

```
dx = pygamd.dump.xml(info=mst, group='all', file='p', period=100000)
app.add(dx)
```

5.4 Group

A group specifies the particles for certain functions, such as integration and data output. Group objects will not be defined in script. Instead, they will be defined in other objects. And then, ususally only keywords or a list of particle types and particle indexes are needed to indicate the particles.

class chare.**particle_set**(*info*, *group*)

Parameters

- **info** – system information.
- **group** – either a string or a python list. The string is a keyword with candidates "all", "body", "charge", and "nonbody". The list could contain particle types and particle indexes.

Example:

```
dm = pygamd.dump.mst(info=mst, group='all', file='p.mst', period=100000)
app.add(dm)

dm = pygamd.dump.mst(info=mst, group=['A', 'B'], file='p.mst', period=100000)
app.add(dm)


dm = pygamd.dump.mst(info=mst, group=['A', 'B', 0, 1, 2], file='p.mst',␣
↪period=100000)
app.add(dm)
```

CHAPTER

SIX

APPLICATION

6.1 Modules management

PYGAMD is organized by being composed of modules. Application manages and calls modules, and thereby run simulations. Usually, we only define an application object in the context of script. The modules can be added into by `add()` or removed from by `remove()` the application before runing `run()` the simulation.

class application.**dynamics**(*info*, *dt*, *sort=True*)

Constructor of application object.

Parameters

- **info** – system information
- **dt** – integration time step
- **sort** – if device memory is sorted by Hilbert curve, the default is True.

add(*object*)

adds an object to the application.

remove(*object*)

removes an added object.

run(*N*)

runs the simulation for N time steps.

Example:

```
app = pygamd.application.dynamics(info=mst, dt=0.001)
# builds up an application.
app.run(10000)
# runs the simulation for 10000 time steps.
```

6.2 Multi-stage simulation

An application can have single or multiple stage simulations. The commands in the context of script are executed sequentially. Every stage simulation is achieved with `run()`. Before a stage of simulation, the modules and parameters can be adjusted. New modules can be added into the applicaitons by `add()`. The added modules at last stage can be removed from the application, otherwise they will be kept. For example:

- First stage simulation:

```
app = pygamd.application.dynamics(info=mst, dt=0.001)
app.add(lj)
app.add(nvt)
app.run(1000)
```

- Second state simulation:

```
app.remove(lj)
app.remove(nvt)
app.add(harmonic)
app.add(npt)
app.run(1000)
```

6.3 Two-dimensional simulation

1. Controlling script, i.e. 'file.py' script is same for two- and three-dimensional simulations.

2. However, configuration file i.e. MST file should indicate two-dimensional system by:

   1. pointing out dimensions with dimensions="2"
   2. setting the length of box in Z diretion to zero with lz="0"
   3. specifying the position of particles in Z direction as 0.0

An example is given:

```
mst_version 1.0
        num_particles
                8
        timestep
                0
        dimension
                2
        box
                200.0    200.0    0.0
        position
                 28.5678528848   -37.9327360252     0.0000000000
                 28.0019705849   -37.1082499897     0.0000000000
                 29.5648198865   -37.8549105956     0.0000000000
                 28.1367681830   -38.8350474902     0.0000000000
                -37.5589154370   -72.8549398355     0.0000000000
                -38.4958248509   -72.5053675968     0.0000000000
                -36.7877222908   -72.2183386015     0.0000000000
                -37.3931991693   -73.8411133084     0.0000000000
mst_end
```

3. For molgen script to generate a two-dimensional configuration file, a specification of two dimensions and box size in Z direction as 0.0 is necessary. Such as:

```
import molgen

mol=molgen.Molecule(4)
mol.setParticleTypes("A,B,B,B")
```

(continues on next page)

(continued from previous page)

```
mol.setTopology("0-1,0-2,0-3")
mol.setBondLength("A","B", 1.0)
mol.setAngleDegree("B", "A", "B", 120)
mol.setInit("B", 1)
mol.setCris("A", 1)


gen=molgen.Generators(200, 200, 0.0) # box size in X, Y, and Z directions
gen.addMolecule(mol, 2000)
gen.setDimension(2)
gen.setMinimumDistance(1.0)
gen.outPutMST("pn2d")
```

CHAPTER

SEVEN

FORCE FIELD

7.1 Non-bonded interactions

Overview

The pygamd MD engine provides a few of functions for non-bonded interactions. However, it supports well self-defined analytical functions via writting codes of device function in script.

Non-bonded functions	`force.nonbonded`
Self-defined functions	`force.nonbonded`
Non-bonded functions for charged beads	`force.nonbonded_c`
Self-defined functions for charged beads	`force.nonbonded_c`

7.1.1 Non-bonded functions

Description:

The function describing non-bonded interactions could be either the one called from non-bonded interaction function libary, or the one defined by user himself in script. Non-bonded interaction function libary contains Lennard-Jones function named as 'lj' and harmonic function named as 'harmonic'.

Lennard-Jones function ('lj')

$$\begin{aligned} V_{\mathrm{LJ}}(r) = & \quad 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \alpha \left(\frac{\sigma}{r}\right)^{6} \right] & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - the depth of the potential well (in energy units)
- σ - the collision diameter (in distance units)
- α - the factor of attraction (unitless)
- r_{cut} - cutoff radius (in distance units) - *note*: equal to or smaller than the global rcut specified in `force.nonbonded`

Harmonic function ('harmonic')

$$V_{\mathrm{H}}(r) = \frac{1}{2}\alpha \left(r - r_{\mathrm{cut}}\right)^2, r < r_{\mathrm{cut}}$$

Coefficients:

- α - spring constant (in units of energy/distance^2)
- r_{cut} - cutoff radius (in distance units) - *note*: equal to or smaller than the global rcut specified in `force.nonbonded`

class force.**nonbonded**(*info*, *rcut*, *func*, *exclusion=None*)

Constructor of non-bonded interaction calculation object.

Parameters

- **info** – system information.
- **rcut** – cut-off radius of interactions.
- **func** – function name.
- **exclusion** – a python list of exclusions, the candidates are 'bond', 'angle', 'dihedral', the default is None.

setParams(*type_i*, *type_j*, *param*)

specifies interaction parameters with type_i, type_j, a list of parameters.

Example:

```
fn = pygamd.force.nonbonded(info=mst, rcut=3.0, func='lj')
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 3.0])
app.add(fn)


fn = pygamd.force.nonbonded(info=mst, rcut=3.0, func='lj', exclusion=['bond'])
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 3.0])
app.add(fn)
```

7.1.2 Self-defined functions

Description:

The device function for non-bonded interactions could be written in script and conveyed to kernel funciton for calculation. The function has three parameters where rsq, param, and fp are square of distance, interaction parameters, and force and potential, respectively.

With the potential form of non-bonded interactions $p(r)$, the expression of parameters in script are:

- p = $p(r)$
- f = $-(\partial p(r)/\partial r)(1/r)$

Function code template:

```
@cuda.jit(device=True)
def func(rsq, param, fp):
        rcut = param[0]
        p1 = param[1]
        p2 = param[2]
        p3 = param[3]
        ...
        if rsq<rcut*rcut:
                calculation codes
                ...
```

(continues on next page)

(continued from previous page)

```
                fp[0]=f
                fp[1]=p

fn = pygamd.force.nonbonded(info, rcut, func)
fn.setParams(type_i, type_j, param=[rcut, p1, p2, p3, ...])
....
app.add(fn)
```

Example:

```
from numba import cuda
import numba as nb

@cuda.jit(device=True)
def lj(rsq, param, fp):
        epsilon = param[0]
        sigma = param[1]
        alpha = param[2]
        rcut = param[3]
        if rsq<rcut*rcut:
                sigma2 = sigma*sigma
                r2inv = sigma2/rsq;
                r6inv = r2inv * r2inv * r2inv;
                f = nb.float32(4.0) * epsilon * r2inv * r6inv * (nb.float32(12.
↪0)
                        * r6inv - nb.float32(6.0) * alpha)/sigma2
                p = nb.float32(4.0) * epsilon * r6inv * ( r6inv - nb.float32(1.
↪0))
                fp[0]=f
                fp[1]=p

fn = pygamd.force.nonbonded(info=mst, rcut=3.0, func=lj)
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 3.0])
app.add(fn)
```

7.1.3 Non-bonded functions for charged beads

Description:

The function describing non-bonded interactions including electrostatic part could be either the one called from non-bonded interaction function libary, or the one defined by user himself in script.

Lennard-Jones-coulomb function ('lj_coulomb')

$$\begin{aligned} V_{\mathrm{LJ}}(r) = & \quad 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \alpha\left(\frac{\sigma}{r}\right)^{6}\right] + f\frac{q_i q_j}{\epsilon_r r} & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

Following coefficients need being set per pair of particle types:

- ϵ - the depth of the potential well (in energy units)
- σ - the collision diameter (in distance units)

- α - the factor of attraction (unitless)
- ϵ_r - dielectric coefficient : $f = 1/4\pi\epsilon_0 = 138.935\ kJ\ mol^{-1}\ nm\ e^{-2}$
- r_{cut} - cutoff radius (in distance units) - *note*: equal to or smaller than the global rcut specified in `force.nonbonded_c`

class force.**nonbonded_c**(*info*, *rcut*, *func*, *exclusion=None*)

Constructor of non-bonded interaction calculation object.

Parameters

- **info** – system information.
- **rcut** – cut-off radius of interactions.
- **func** – function name.
- **exclusion** – a python list of exclusions, the candidates are 'bond', 'angle', 'dihedral', the default is None.

setParams(*type_i*, *type_j*, *param*)

specifies interaction parameters with type_i, type_j, a list of parameters.

Example:

```
fn = pygamd.force.nonbonded_c(info=mst, rcut=3.0, func='lj_coulomb')
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 15.0, 3.0])
app.add(fn)


fn = pygamd.force.nonbonded_c(info=mst, rcut=3.0, func='lj_coulomb', exclusion=[
↪'bond'])
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 15.0, 3.0])
app.add(fn)
```

7.1.4 Self-defined functions for charged beads

Description:

The device function for non-bonded interactions could be written in script and conveyed to kernel funciton for calculation. The function has three parameters where rsq, qi, qj, param, and fp are the square of distance, qi and qj, interaction parameters, and force and potential, respectively.

With the potential form of non-bonded interactions $p(r)$, the expression of parameters in script are:

- p = $p(r)$
- f = $-(\triangle p(r)/\triangle r)(1/r)$

Function code template:

```
@cuda.jit(device=True)
def func(rsq, qi, qj, param, fp):
        rcut = param[0]
        p1 = param[1]
        p2 = param[2]
        p3 = param[3]
        ...
```

(continues on next page)

(continued from previous page)

```
        if rsq<rcut*rcut:
                calculation codes
                ...
                fp[0]=f
                fp[1]=p

fn = pygamd.force.nonbonded_c(info, rcut, func)
fn.setParams(type_i, type_j, param=[rcut, p1, p2, p3, ...])
....
app.add(fn)
```

Example:

```
from numba import cuda
import numba as nb

@cuda.jit(device=True)
def lj_coulomb(rsq, qi, qj, param, fp):
        epsilon = param[0]
        sigma = param[1]
        alpha = param[2]
        epsilonr = param[3]
        rcut = param[4]
        coulomb_eff = 138.935/epsilonr
        if rsq<rcut*rcut:
                sigma2 = sigma*sigma
                r2inv = sigma2/rsq;
                r6inv = r2inv * r2inv * r2inv;
                f = nb.float32(4.0) * epsilon * r2inv * r6inv * (nb.float32(12.
↪0)
                        * r6inv - nb.float32(6.0) * alpha)/sigma2 + coulomb_
↪eff*qi*qj*r2inv*rinv
                p = nb.float32(4.0) * epsilon * r6inv * ( r6inv - nb.float32(1.
↪0))
                        + coulomb_eff*qi*qj*rinv
                fp[0]=f
                fp[1]=p

fn = pygamd.force.nonbonded_c(info=mst, rcut=3.0, func=lj_coulomb)
fn.setParams(type_i="a", type_j="a", param=[1.0, 1.0, 1.0, 15.0, 3.0])
app.add(fn)
```

7.2 Bonded interactions

7.2.1 Bond interactions

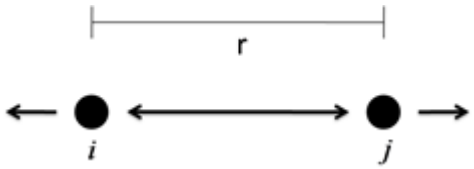
Overview

Bonds impose connected forces on specific pairs of particles to model chemical bonds. The bonds are specified in *MST format* configuration file with the format:

```
bond
bond_type(str)  particle_i(int)  particle_j(int)
...
```

By themselves, bonds do nothing. Only when a bond force object is instantiated in script(i.e. *force.bond*), are bond forces actually calculated.

Bond functions *force.bond*



Bond functions

Description:

Function of bond interactions could be either the one called from bond interaction function libary, or the one defined by user himself. Bond interaction function libary contains harmonic function named as 'harmonic'.

Harmonic function (harmonic)

$$V_{\mathrm{bond}}(r) = \frac{1}{2}k\left(r - r_0\right)^2$$

Coefficients:

- k - spring constant `k` (in units of energy/distance^2)
- r_0 - equilibrium length `r0` (in distance units)

class force.**bond**(*info*, *func*)

Constructor of a bond interaction object.

Parameters

- **info** – system information.
- **func** – function that is either a string or a device function.

setParams(*bond_type*, *param*)

specifies the bond interaction parameters with bond type and a list of parameters.

Example:

```
fb = pygamd.force.bond(info=mst, func='harmonic')
fb.setParams(bond_type = 'A-A', param=[4.0, 0.0])#(param=[k, r0])
fb.setParams(bond_type = 'A-B', param=[4.0, 0.0])#(param=[k, r0])
fb.setParams(bond_type = 'B-B', param=[4.0, 0.0])#(param=[k, r0])
app.add(fb)
```

Self-defined bond functions

Description:

The device function for bond interactions could be written in script and conveyed to kernel funciton for calculation.

With the potential form of bond interactions $p(r)$, the expression of parameters in script are:

- p = $p(r)$
- f = $-(\partial p(r)/\partial r)(1/r)$

Function code template:

```
@cuda.jit(device=True)
def func(rsq, param, fp):
        p0 = param[0]
        p1 = param[1]
        ...
        calculation codes
        ...
        fp[0]=f
        fp[1]=p

fb = pygamd.force.bond(info, func)
fb.setParams(bond_type, param=[p0, p1, ...])
app.add(fb)
```

Example:

```
from numba import cuda
import numba as nb

@cuda.jit(device=True)
def harmonic(rsq, param, fp):
        k = param[0]
        r0 = param[1]
        r = math.sqrt(rsq)
        f = k * (r0/r - nb.float32(1.0))
        p = nb.float32(0.5) * k * (r0 - r) * (r0 - r)
        fp[0]=f
        fp[1]=p

fb = pygamd.force.bond(info=mst, func=harmonic)
fb.setParams(bond_type='a-a', param=[100.0, 1.0])
app.add(fb)
```

7.2.2 Angle bending

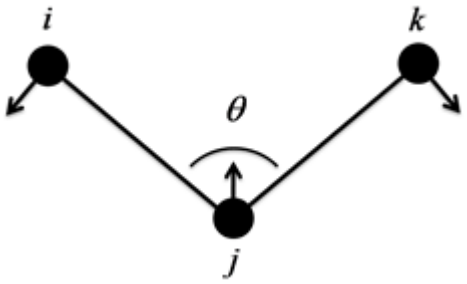
Overview

Angles impose forces on specific triplets of particles to model chemical angles between two bonds. The angles are specified in *MST format* configuration file with the format:

```
angle
angle_type(str)  particle_i(int)  particle_j(int)  particle_k(int)
...
```

By themselves, angles do nothing. Only when an angle force object is instantiated(i.e. `force.angle`), are angle forces actually calculated.

Angle functions `force.angle`



Angle functions

Description:

Function of angle interactions could be either the one called from angle interaction function libary, or the one defined by user himself. Angle interaction function libary contains harmonic function named as 'harmonic' and harmonic cosine function named as 'harmonic_cos'.

Harmonic function (harmonic)

$$V_{\text{angle}}(\theta) = \frac{1}{2} k \left(\theta - \theta_0\right)^2$$

Coefficients:

- k - potential constant `k` (in units of energy/radians^2)
- θ_0 - equilibrium angle `theta0` (in radians)

Note: Angles for the functions in library should be given in script in the unit of degree, and the program will convert them into radian automatically.

Harmonic cosine function (harmonic_cos)

$$V_{\text{angle}}(\theta) = k \left[1 - \cos\left(\theta - \theta_0\right)\right]$$

Coefficients:

- k - potential constant `k` (in units of energy)

- θ_0 - equilibrium angle `theta0` (in radians)

Note: Angles for the functions in library should be given in script in the unit of degree, and the program will convert them into radian automatically.

---

class force.**angle**(*info*, *func*)

Constructor of an angle interaction object.

Parameters

- **info** – system information.
- **func** – function that is either a string or a device function.

setParams(*angle_type*, *param*)

specifies the angle interaction parameters with angle type and a list of parameters.

Example:

```
fa = pygamd.force.angle(info=mst, func='harmonic')
fa.setParams(angle_type='a-a-a', param=[100.0, 90.0])
app.add(fa)
```

Self-defined bond functions

Description:

The device function for angle interactions could be written in script and conveyed to kernel funciton for calculation.

With the potential form of angle interactions $p(\theta)$, the expression of parameters in script are:

- p = $p(\theta)$
- f = $\partial p(\theta)/\partial\theta$

Function code template:

```
@cuda.jit(device=True)
def func(cos_abc, sin_abc, param, fp):
        p0 = param[0]
        p1 = param[1]
        ...
        calculation codes
        ...
        fp[0]=f
        fp[1]=p

fa = pygamd.force.angle(info, func)
fa.setParams(bond_type, param=[p0, p1, ...])
app.add(fa)
```

Example:

```
from numba import cuda
import numba as nb

@cuda.jit(device=True)
def harmonic(cos_abc, sin_abc, param, fp):
        k = param[0]
        t0 = param[1]
        dth = math.acos(cos_abc) - math.pi*t0/180.0
        f = k * dth
        p = nb.float32(0.5) * f * dth
        fp[0]=f
        fp[1]=p

fa = pygamd.force.angle(info=mst, func=harmonic)
fa.setParams(angle_type='a-a-a', param=[400.0, 90.0])#param=[k, t0]
app.add(fa)
```

7.2.3 Dihedral torsion

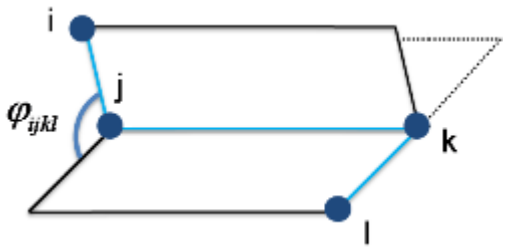
Overview

Dihedrals impose forces on specific quadruplets of particles to model the rotation about chemical bonds. The dihedrals are specified in *MST format* configuration file with the format:

```
dihedral
dihedral_type(str)  particle_i(int)  particle_j(int)  particle_k(int)  particle_l(int)
...
```

By themselves, dihedrals do nothing. Only when a dihedral force object is instantiated(i.e. *force.dihedral*), are dihedral forces actually calculated.

Dihedral functions	*force.dihedral*



Dihedral functions

Description:

Function of angle interactions could be either the one called from angle interaction function libary, or the one defined by user himself. Angle interaction function libary contains harmonic function named as 'harmonic' and harmonic cosine function named as 'harmonic_cos'.

Harmonic function for proper dihedrals(harmonic)

$$V_{\mathrm{dihedral}}(\varphi) = k\left[1 + f\cos\left(\varphi - \delta\right)\right]$$

Coefficients:

- k - multiplicative constant `k` (in units of energy)
- δ - phase shift angle `delta` (in radians)
- f - factor `f` (unitless) - *optional*: defaults to *-1.0*

Note: Dihedral angles for the functions in library should be given in script in the unit of degree, and the program will convert them into radian automatically.

Harmonic function for improper dihedrals (harmonic)

$$V_{\mathrm{dihedral}}(\varphi) = k\left(\varphi - \delta\right)^2$$

Coefficients:

- k - potential constant `k` (in units of energy/radians^2)
- δ - phase shift angle `delta` (in radians)

Note: Dihedral angles for the functions in library should be given in script in the unit of degree, and the program will convert them into radian automatically.

`class force.dihedral(`*info*, *func*`)`

Constructor of a dihedral interaction object.

Parameters

- **info** – system information.
- **func** – function that is either a string or a device function.

`setParams(`*dihedral_type*, *param*, *term='proper'*`)`

specifies the dihedral interaction parameters with dihedral type, a list of parameters and the term of dihedral. The term candidates of dihedral are 'proper' and 'improper' with the default 'proper'.

`setCosFactor(`*factor*`)`

specifies the factor of harmonic function for proper dihedral.

Example:

```
fd = pygamd.force.dihedral(info=mst, func='harmonic')
fd.setParams(dihedral_type='a-a-a-a', param=[100.0, 90.0])
app.add(fd)
```

Self-defined bond functions

Description:

The device function for dihedral interactions could be written in script and conveyed to kernel funciton for calculation.

With the potential form of dihedral interactions $p(\varphi)$, the expression of parameters in script are:

- p = $p(\varphi)$
- f = $\partial p(\varphi)/\partial\varphi$

Function code template:

```
@cuda.jit(device=True)
def func(cos_abcd, sin_abcd, param, fp):
        p0 = param[0]
        p1 = param[1]
        ...
        calculation codes
        ...
        fp[0]=f
        fp[1]=p

fd = pygamd.force.dihedral(info, func)
fd.setParams(dihedral_type, param=[p0, p1, ...])
app.add(fd)
```

Example:

```
from numba import cuda
import numba as nb

@cuda.jit(device=True)
def harmonic(cos_abcd, sin_abcd, param, fp):
        k = param[0]
        cos_phi0 = param[1]
        sin_phi0 = param[2]
        cos_factor = param[3]
        f = cos_factor * (-sin_abcd*cos_phi0 + cos_abcd*sin_phi0)
        p = nb.float32(1.0) + cos_factor * (cos_abcd*cos_phi0 + sin_abcd*sin_
↪phi0)
        fp[0]=-k*f
        fp[1]=k*p

fd = pygamd.force.dihedral(info=mst, func=harmonic)
fd.setParams(dihedral_type='a-a-a-a', param=[100.0, math.cos(math.pi), math.
↪sin(math.pi), -1.0])
app.add(fd)
```

CHAPTER

EIGHT

INTEGRATION

8.1 NVT ensemble

Overview

NVT	*integration.nvt*
GWVV	*integration.gwvv*
BD	*integration.bd*

8.1.1 NVT

class integration.**nvt**(*info*, *group*, *method*, *tau*, *temperature*)

Constructor of a NVT thermostat object for a group of particles.

Parameters

- **info** – system information.
- **group** – a group of particles.
- **method** – thermostat method, the candidates are "nh" for nose hoover.
- **tau** – thermostat coupling parameter.
- **temperature** – temperature.

setT(*float T*)

specifies the temperature as a constant value.

setT(*Variant vT*)

specifies the temperature as a function of time steps.

Example:

```
inn = pygamd.integration.nvt(info=mst, group=['a'], method="nh", tau=1.0,␣
↪temperature=1.0)
app.add(inn)
```

8.1.2 GWVV

class integration.**gwvv**(*info*, *group*)

Constructor of a GWVV thermostat object for a group of particles.

Parameters

- **info** – system information.
- **group** – a group of particles.

Example:

```
inn = pygamd.integration.gwvv(info=mst, group='all')
app.add(inn)
```

8.1.3 BD

class integration.**bd**(*info*, *group*, *temperature*)

Constructor of a Brownian Dynamics thermostat object for a group of particles.

Parameters

- **info** – system information.
- **group** – a group of particles.
- **temperature** – temperature.

setParams(*string typ*, *float gamma*)

specifies the gamma parameter for particle type.

Example:

```
inn = pygamd.integration.bd(info=mst, group="all", temperature=1.0)
app.add(inn)
```

CHAPTER

NINE

MODULES

9.1 Dissipative particle dynamics

9.1.1 DPD force

Description:

The DPD force consists of pair-wise conservative, dissipative and random terms.

$$\begin{aligned} \vec{F}_{ij}^{C} &= \alpha \left(1 - \frac{r_{ij}}{r_{cut}}\right) \vec{e}_{ij} \\ \vec{F}_{ij}^{D} &= -\gamma \omega^{D}(r_{ij})(\vec{e}_{ij} \cdot \vec{v}_{ij})\vec{e}_{ij} \\ \vec{F}_{ij}^{R} &= T\sigma\omega^{R}(r_{ij})\xi_{ij}\vec{e}_{ij} \end{aligned}$$

- $\gamma = \sigma^2/2k_BT$
- $\omega^D(r_{ij}) = [\omega^R(r_{ij})]^2 = (1 - r_{ij}/r_{cut})^2$
- ξ_{ij} - a random number with zero mean and unit variance
- T - *temperature* - *optional*: defaults to 1.0
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to 1.0

The following coefficients must be set per unique pair of particle types:

- α - *alpha* (in energy units)
- σ - *sigma* (unitless)

class force.**dpd**(*info*, *rcut=1.0*)

Constructor of a DPD interaction object.

Parameters

- **info** – system information.
- **rcut** – the cut-off radius of interactions.

setParams(*type_i*, *type_j*, *alpha*, *sigma*)

specifies the DPD interaction parameters between two types of particles.

Example:

```
fn = pygamd.force.dpd(info=mst, rcut=1.0)
fn.setParams(type_i="A", type_j="A", alpha=25.0, sigma=3.0)
app.add(fn)
```

9.1.2 GWVV integration

Description:

Integration algorithm.

$$
\begin{aligned}
v_i^0 &\leftarrow v_i + \lambda \frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t}) \\
v_i &\leftarrow v_i + \frac{1}{2}\frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t}) \\
r_i &\leftarrow r_i + v_i \Delta t \\
&Calculate \quad F_i^c\{r_j\}, F_i^d\{r_j, v_j^0\}, F_i^r\{r_j\} \\
v_i &\leftarrow v_i + \frac{1}{2}\frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t})
\end{aligned}
$$

- λ - *lambda* (unitless) - *optional*: defaults to 0.65

class integration.**gwvv**(*info*, *group*)

Constructor of a GWVV NVT thermostat for a group of DPD particles.

Parameters

- **info** – system information.
- **group** – a group of particles.

Example:

```
gw = pygamd.integration.gwvv(info=mst, group='all')
app.add(gw)
```

CHAPTER

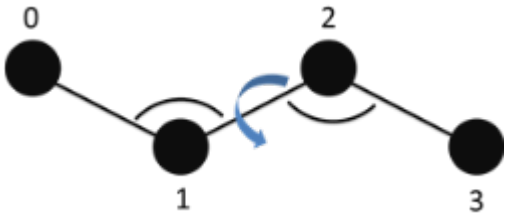
TEN

DATA

10.1 Data format

10.1.1 XML format

We take XML format files as the standard input and output configuration files. The XML files can contain coordinates, types, masses, velocities, bond connections, angles, dihedrals and so on. Here is an example of the XML file of a single molecule system. The molecule consisting of four particles is depicted in following picture.



The data in a line of XML file corresponds to a particle and all particles are given in sequence. For example, the coordinate of a particle in x, y, and z directions is written in a line and three columns in XML files. However, this rule does not include topological relevant information, including bonds, angles and dihedrals.

An example XML file with particles coordinates, velocities, types, masses ...

```
<?xml version="1.0" encoding="UTF-8"?>
<galamost_xml version="1.3">
<configuration time_step="0" dimensions="3" natoms="4" >
<box lx="10" ly="10" lz="10"/>
<position num="4">
-1 2 -1
-2 3 0
-1 4 1
-1 5 2
</position>
<velocity num="4">
1 2 3
1 0 0
3 -2 1
0 1 1
</velocity>
<type num="4">
A
```

(continues on next page)

(continued from previous page)

```
B
B
A
</type>
<mass num="4">
1.0
2.1
1.0
1.0
</mass>
</configuration>
</galamost_xml>
```

The file could include the nodes of bond, angle, dihedral ...

```
# bond with 'bond type, the index of particle i, j'.
 <bond num="3">
 polymer 0 1
 polymer 1 2
 polymer 2 3
 </bond>

# angle with 'angle type, the index of particle i, j, k'.
 <angle num="2">
 theta 0 1 2
 theta 1 2 3
 </angle>

# dihedral with 'dihedral type, the index of particle i, j, k, l'.
 <dihedral num="1">
 phi 0 1 2 3
 </dihedral>
```

The other nodes of XML ...

```
 # the diameter of particles in float type.
  <diameter num="4">
  1.0
  1.0
  1.0
  1.0
  </diameter>

 # the charge of particles in float type.
  <charge num="4">
   1.333
   1.333
  -1.333
  -1.333
  </charge>

 # the body index of particles in int type, -1 for non-body particles.
```

(continues on next page)

(continued from previous page)

```
 <body num="4">
 -1
 -1
 0
 0
 </body>

# the image in x, y, and z directions of particles in int3 type.
 <image num="4">
 0 0 0
 0 0 0
 0 0 0
 0 0 0
 </image>

# the velocity in x, y, and z directions of particles in float3 type.
 <velocity num="4">
  3.768     -2.595    -1.874
 -3.988     -1.148     2.800
  1.570      1.015    -3.167
  2.441     -1.859    -1.039
 </velocity>


# the orientation vector (x, y, z) of particles in float3 type.
 <orientation num="4">
  -0.922     0.085     0.376
  -0.411    -0.637    -0.651
   0.293     0.892    -0.342
  -0.223     0.084     0.970
 </orientation>

# the quaternion vector (x, y, z, w) of particles in float4 type.
 <quaternion num="4">
  0.369   0.817  -0.143   0.418
 -0.516  -0.552   0.653   0.024
 -0.521  -0.002   0.131   0.843
 -0.640   0.159  -0.048  -0.749
 </quaternion>

# the angular velocity of rotation in x, y, and z directions of particles in␣
↪float3 type.
 <rotation num="4">
  -0.640    0.571   -0.512
  -0.744    0.346    0.569
   0.620   -0.086    0.779
  -0.542    0.319   -0.776
 </rotation>

# the moment of inertia in x, y, and z directions of particles in float3 type.
 <inert num="4">
 1.0 1.0 3.0
```

(continues on next page)

(continued from previous page)

```
  1.0 1.0 3.0
  1.0 1.0 3.0
  1.0 1.0 3.0
  </inert>

# the initiator indication of particles in int type, 1 for initiator.
  <h_init num="4">
  0
  1
  0
  1
  </h_init>

# the crosslinking number of particles in int type, 0 for reactable monomer.
  <h_cris num="4">
  0
  0
  0
  0
  </h_cris>

# the molecule index of particles in int type.
  <molecule num="4">
  0
  0
  1
  1
  </molecule>
```

The nodes of anisotropic particle attribute …

```
# the particle patch attribute with 'particle type, patch number'
# followd by 'patch type, patch size, patch position vector in x, y, z␣
↪directions'.
  <Patches>
  B 2
  p1 60  0    0    1
  p1 60  0    0   -1
  </Patches>

# the patch-patch interaction parameter with 'patch type, patch type, gamma_
↪epsilon, alpha'.
  <PatchParams>
  p1 p1 88.0 0.5
  </PatchParams>

# the particle shape attribute with 'particle type, diameter a, diameter b,␣
↪diameter c,
# epsion a, epsion b, epsion c'. The a, b, c are along x, y, z directions in␣
↪body frame,
# respectively.
  <Aspheres>
```

(continues on next page)

(continued from previous page)

```
A 1.0 1.0 1.0 3.0 3.0 3.0
B 1.0 1.0 3.0 1.0 1.0 0.2
</Aspheres>
```

10.2 Data input

class Reader

The basic class of XmlReader and *BinaryReader*.

10.2.1 XML reader

class XMLReader(*filename*)

The constructor of XML file parser object.

Parameters
filename (*str*) – The input XML file name.

Example:

```
filename = 'dppc.xml'
# sets the name of input XML file.
build_method = gala.XMLReader(filename)
# builds up a parser object for the input XML file.
```

10.2.2 Binary reader

class BinaryReader(*filename*)

The constructor of binary file parser object.

Parameters
filename (*str*) – The input binary file name.

Example:

```
filename = 'initial.bin'
# sets the name of binary file.

build_method = gala.BinaryReader(filename)
# builds up a reading object for input binary file.
```

10.3 Data output

10.3.1 Common functions for Data output

setPrecision(*int npre*)

Set the number of places after decimal point.

setHead(*int nhead*)

Set the number of places before decimal point.

10.3.2 Collective information

class DumpInfo(*all_info*, *comp_info*, *filename*)

Constructor of an information dump object for a group of particles.

Parameters

- **all_info** (AllInfo) – System information.
- **comp_info** (ComputeInfo) – Object for calculating collective information.
- **filename** (*str*) – Output file name.

dumpAnisotropy()

Outputs information related to anisotropic particles.

dumpVirial(*Force object*)

Outputs virials of the Force object.

dumpPotential(*Force object*)

Outputs potentials of the Force object.

dumpVirialMatrix(*Force object*)

Outputs virial matrixes including 'virial_xx', 'virial_xy', 'virial_xz', 'virial_yy', 'virial_yz', and 'virial_zz' of the Force object.

dumpPressTensor()

Outputs press tensors including 'press_xx', 'press_xy', 'press_xz', 'press_yy', 'press_yz', and 'press_zz' of the system.

dumpTypeTemp(*string type*)

Outputs temperatures of a type of particles.

dumpParticleForce(*int tag_i*)

Outputs the forces of a particle indicated by tag_i in order to trace it's force condition.

dumpParticlePosition(*int tag_i*)

Outputs the positions of a particle indicated by tag_i in order to trace it's position.

dumpBoxSize()

Outputs box sizes including the box lengths in 'X', 'Y', and 'Z' directions and volume.

setPeriod(*int period*)

Period to output data.

Example:

```
dinfo = gala.DumpInfo(all_info, comp_info, 'data.log')
dinfo.setPeriod(200)
app.add(dinfo)
```

10.3.3 MOL2 dump

class MOL2Dump(*all_info*, *filename*)

Constructor of an object to dump mol2 files.

Parameters

- **all_info** (AllInfo) – System information.
- **filename** (*str*) – Output file base name.

setChangeFreeType(*string type*)

specifies the type of free particles which will be changed to be 'F' in output file.

deleteBoundaryBond(*bool switch*)

switches on the function of screening the bonds across the box with 'True'.

Example:

```
mol2 = gala.MOL2Dump(all_info, 'particles')
mol2.setPeriod(100000)
mol2.deleteBoundaryBond(True)
app.add(mol2)
```

10.3.4 XML dump

class XMLDump(*all_info*, *filename*)

Constructor of an object to dump XML files.

Parameters

- **all_info** (AllInfo) – System information.
- **filename** (*str*) – Output file base name.

class XMLDump(*all_info*, *group*, *filename*)

Constructor of an object to dump XML files for a group of particles.

Parameters

- **all_info** (AllInfo) – System information.
- **group** (ParticleSet) – A group of particles.
- **filename** (*str*) – Output file base name.

setOutput(*PyObject* out_put_list*)

indicates the output data type with the candidates:

```
        ['position', 'type', 'velocity', 'mass', 'image', 'force',
        'potential', 'virial', 'virial_matrix', 'charge', 'diameter',
        'body', 'orientation', 'quaternion', 'rotation', 'rotangle',
        'torque', 'inert', 'init', 'cris', 'molecule', 'bond', 'angle',
        'dihedral', 'constraint', 'vsite']

Each data type also could be outputed by a single function as following.
```

setOutputPosition(*bool switch*)

Outputs particle position (default value is true).

setOutputType(*bool switch*)

Outputs particle type (default value is true).

setOutputImage(*bool switch*)

Outputs particle image.

setOutputVelocity(*bool switch*)

Outputs particle velocity.

setOutputMass(*bool switch*)

Outputs particle mass.

setOutputCharge(*bool switch*)

Outputs particle charge.

setOutputDiameter(*bool switch*)

Outputs particle diameter.

setOutputBody(*bool switch*)

Outputs particle body.

setOutputVirial(*bool switch*)

Outputs particle virial.

setOutputForce(*bool switch*)

Outputs particle force (x, y, z).

setOutputPotential(*bool switch*)

Outputs particle potential.

setOutputOrientation(*bool switch*)

Outputs particle orientation.

setOutputQuaternion(*bool switch*)

Outputs particle quaternion.

setOutputRotation(*bool switch*)

Outputs particle rotation velocity.

setOutputRotangle(*bool switch*)

Outputs particle accumulated rotated angle at the direction of (0, 0, 1) in 3D or (0, 1, 0) in 2D.

setOutputTorque(*bool switch*)

Outputs particle torque.

setOutputInert(*bool switch*)

Outputs particle inert tensor.

setOutputInit(*bool switch*)

Outputs particle initiator indicator.

setOutputCris(*bool switch*)

Outputs particle cross-linking indicator.

setOutputBond(*bool switch*)

Outputs bonds.

setOutputAngle(*bool switch*)

Outputs angles.

setOutputDihedral(*bool switch*)

Outputs dihedrals.

setOutputConstraint(*bool switch*)

Outputs bond constraints.

setOutputVsite(*bool switch*)

Outputs virual sites.

setOutputLocalForce(*Force object*)

Outputs particle force(x, y, z and w where w is potential) of a Force object.

setOutputLocalVirial(*Force object*)

Outputs particle virial of a Force object.

setOutputLocalVirialMatrix(*Force object*)

Outputs particle virial matrix of a Force object.

setOutputPatch(*AniForce object*)

outputs patch information for display in OVITO.

setOutputEllipsoid(*BondForceHarmonicEllipsoid object*)

outputs ellipsoid bond information for display in OVITO.

setOutputEllipsoid(*PBGBForce object*)

outputs ellipsoid information for display in OVITO.

setOutputEllipsoid(*GBForce object*)

outputs ellipsoid information for display in OVITO.

Example:

```
xml = gala.XMLDump(all_info, 'particles')
xml.setOutput(['image', 'bond'])
xml.setPeriod(100000)
app.add(xml)
```

10.3.5 DCD trajectory dump

class DCDDump(*all_info*, *filename*, *overwrite*)

The constructor of a dump object of DCD file.

Parameters

- **all_info** (AllInfo) – The system information.
- **filename** (*str*) – The output file name.
- **overwrite** (*bool*) – If overwrite the existed DCD file.

class DCDDump(*all_info*, *group*, *filename*, *overwrite*)

The constructor of a dump object of DCD file for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **filename** (*str*) – The output file name.
- **overwrite** (*bool*) – If overwrite the existed DCD file.

unpbc(*bool switch*)

Outputs particle positions without the application of periodic boundary condition (PBC). Default value is False, i.e. with PBC condition.

unwrap(*bool switch*)

Unwraps the molecules across box boundary due to PBC condition. Default value is False, i.e. wrapping molecules.

Example:

```
dcd = gala.DCDDump(all_info, 'particles', True)
dcd.unwrap(True)
dcd.setPeriod(100000)
app.add(dcd)
```

10.3.6 Binary dump

class BinaryDump(*all_info*, *filename*)

Constructor of an object to dump binary files.

Parameters

- **all_info** (AllInfo) – System information.
- **filename** (*str*) – Output file base name.

setOutput(*PyObject* out_put_list*)

indicates the output data type with the candidates:

```
['position', 'type', 'velocity', 'mass', 'image', 'force',
'potential', 'virial', 'virial_matrix', 'charge', 'diameter',
'body', 'orientation', 'quaternion', 'rotation', 'rotangle',
'torque', 'inert', 'init', 'cris', 'molecule', 'bond', 'angle',
'dihedral', 'constraint', 'vsite']
```

setOutputAll()

Outputs all data.

setOutputForRestart()

Outputs data needed for restarting.

enableCompression(*bool switch*)

Compresses output file.

Example:

```
binary = gala.BinaryDump(all_info, 'particle')
binary.setOutput(['image', 'bond'])
binary.setPeriod(10000)
app.add(binary)
```

CHAPTER

ELEVEN

SYSTEM

11.1 Import libraries

11.1.1 import cuda lib (Linux)

Import the pygamd module from cuda libray for the simulations on NVIDIA GPU.

Examples:

```
from poetry import cu_gala as gala
```

11.1.2 import hip lib (Linux)

Import the pygamd module from hip libray for the simulations on DCU.

Examples:

```
from poetry import hip_gala as gala
```

11.1.3 import cuda lib (Windows)

Import the pygamd module from cuda libray for the simulations on NVIDIA GPU.

Examples:

```
from poetry import win_gala as gala
```

11.2 Information

11.2.1 Perform configuration

class PerformConfig(*gpu_list*)

The constructor of perform configuration object.

Parameters

gpu_list (*PyObject**) – The gpu list.

Example:

```
global _options
parser = OptionParser()
parser.add_option('--gpu', dest='gpu',help='GPU on which to execute')
(_options, args) = parser.parse_args()

perform_config = gala.PerformConfig(_options.gpu)
```

11.2.2 All information

class AllInfo(*reader*, *perf_conf*)

The constructor of all information object.

Parameters

- **reader** (Reader) – The file parser
- **perf_conf** (PerformConfig) – The perform configuration

setNDimensions(*unsigned int nds*)

set the number of dimensions

addParticleType(*string ptype*)

add a particle type to system

addBondType(*string bondtype*)

add a bond type to system

addAngleType(*string angletype*)

add a angle type to system

addBondTypeByPairs()

add bond types to system according to particle types

addAngleTypeByPairs()

add angle types to system according to particle types

Example:

```
filename = 'PE-1000.xml'
build_method = gala.XMLReader(filename)
perform_config = gala.PerformConfig(_options.gpu)
all_info = gala.AllInfo(build_method, perform_config)
```

11.3 Particle list

11.3.1 Neighbor list

class NeighborList(*all_info*, *r_cut*, *r_buffer*)

Constructor of a neighbor list object

Parameters

- **all_info** (AllInfo) – System information

- **r_cut** (*float*) – Cut-off radius
- **r_buffer** (*float*) – Buffer distance

setRCut(*float r_cut*, *float r_buffer*)

specifies the cut-off and buffer distance.

setRCutPair(*string typi*, *string typj*, *float r_cut*)

specifies the cut-off per unique pair of particle types.

setNsq()

switches on the method of searching all particle to build up list.

setDataReproducibility()

switches on the data reproducibility.

addExclusionsFromBonds()

adds 1-2 exclusion into exclusion list.

addExclusionsFromAngles()

adds 1-3 exclusion into exclusion list.

addExclusionsFromDihedrals()

adds 1-4 exclusion into exclusion list.

addExclusionsFromBodies()

adds body exclusion into exclusion list.

setFilterDiameters()

considers the radius of particle in neighbor list which includes the particles within r_cut + (diameter_i + diameter_j)/2.

Example:

```
neighbor_list = gala.NeighborList(all_info, 3.0 ,0.4)
```

11.3.2 Cell list

class CellList(*all_info*)

Constructor of a cell list object

Parameters
all_info (AllInfo) – System information

setNominalWidth(*float width*)

specifies the length of cell.

setNominalDim(*unsigned int x*, *unsigned int y*, *unsigned int z*)

specifies the dimensions of grid in 'X', 'Y', and 'Z' directions.

setDataReproducibility()

switches on data reproducibility function.

Example:

```
cell_list = gala.CellList(all_info)
```

11.4 Particle set

class ParticleSet(*all_info*, *keyword*)

The constructor of particle set object with keyword.

Parameters

- **all_info** (AllInfo) – The system information.
- **keyword** (*str*) – Keywords or the python list of keywords. The candidates of keyword are "all", "body", "non_body", "charge", and particle type(string).

class ParticleSet(*all_info*, *min*, *max*)

The constructor of particle set object with the range of particle index.

Parameters

- **all_info** (AllInfo) – The system information.
- **min** (*int*) – The particle index range from min to max.
- **max** (*int*) – The particle index range from min to max.

class ParticleSet(*all_info*, *members_list*)

The constructor of particle set object with python object.

Parameters

- **all_info** (AllInfo) – The system information.
- **members_list** (*PyObject**) – The Python object. Note: multiple keywords and particle index can be included in Python object

class ParticleSet(*all_info*, *member_tags*)

The constructor of particle set object with a vector of particle tags.

Parameters

- **all_info** (AllInfo) – The system information.
- **member_tags** (*vector<unsigned_int>*) – A vector of particle tags

ParticleSet combine(ParticleSet group1, ParticleSet group2)

combines two particle groups into one.

getNumMembers()

return the number of group members .

Example:

```
groupC = gala.ParticleSet(all_info,'C')
# initializes a particle set object by a particle type.

group = gala.ParticleSet(all_info,['A', 'B', 'C'])
# initializes a particle set object by particle types

groupB = gala.ParticleSet(all_info,'body')
# initializes a particle set object of body particles by 'body'.

group_e = gala.ParticleSet(all_info,'charge')
# initializes a particle set object of charged particles by 'charge'.
```

(continues on next page)

(continued from previous page)

```
groupC = gala.ParticleSet(all_info,'all')
# initializes a particle set object of all particles by 'all'.

group= gala.ParticleSet(all_info, ['A', 12, 'body'])
# initializes a particle set object of particles by mixed keywords.

groupAB= gala.ParticleSet.combine(groupA, groupB)
# combines two particle sets into one set.
```

11.5 Dynamic Particle Set

class DynamicParticleSet(*all_info*, *keyword*)

The constructor of particle set object with keyword. The member of the particle set is automatically updated every time step.

Parameters

- **all_info** (AllInfo) – The system information.
- **keyword** (*str*) – Keywords or the python list of keywords. The candidates of keyword are particle type(string).

class DynamicParticleSet(*all_info*, *lx_min*, *lx_max*, *ly_min*, *ly_max*, *lz_min*, *lz_max*)

The constructor of particle set object with space range. The member of the particle set is automatically updated every time step.

Parameters

- **all_info** (AllInfo) – The system information.
- **lx_min** (*int*) – The particles in the box with lx >= lx_min.
- **lx_max** (*int*) – The particles in the box with lx < lx_max.
- **ly_min** (*int*) – The particles in the box with ly >= ly_min.
- **ly_max** (*int*) – The particles in the box with ly < ly_max.
- **lz_min** (*int*) – The particles in the box with lz >= lz_min.
- **lz_max** (*int*) – The particles in the box with lz < lz_max.

Example:

```
groupC = gala.DynamicParticleSet(all_info, -10.0, 10.0, -10.0, 10.0, -2.0, 2.0)
# initializes a particle set object by spatial range.
```

11.6 Information computation

class ComputeInfo(*all_info*, *group*)

The constructor of an object of computing some important information, including temperature, pressure, momentum, and potential of a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setNdof(*unsigned int nfreedom*)

sets the degree of freedom.

Example:

```
comp_info = gala.ComputeInfo(all_info, group)
```

11.7 Memory sorting

class Sort(*all_info*)

The constructor of a memory sort object.

Parameters
all_info (AllInfo) – The system information

Example:

```
sort_method = gala.Sort(all_info)
sort_method.setPeriod(300)
app.add(sort_method)
```

CHAPTER

TWELVE

APPLICATION

12.1 Modules management

Usually, we only define an application object in the context of script. The modules can be added `add()` into or removed `remove()` from the application before runing `run()` the simulation.

class Application(*all_info*, *dt*)

The constructor of an application object.

Parameters

- **all_info** (AllInfo) – system information
- **dt** (*float*) – integration time step

add(*object*)

adds an object to the application.

remove(*object*)

removes an added object.

clear()

removes all objects from the application.

setDt(*float dt*)

sets integration time step.

run(*unsigned int N*)

runs the simulation for N time steps.

Example:

```
dt = 0.001
app = gala.Application(all_info, dt)
# builds up an application.
app.run(10000)
# runs the simulation for 10000 time steps.
```

12.2 Multi-stage simulation

An application can have single or multiple stage simulations. The commands in the context of script are executed sequentially. Every stage simulation is achieved with `run()`. Before a stage of simulation, the modules and parameters can be adjusted. New modules can be added into the applicaitons by `add()`. The added modules at last stage can be removed from the application, otherwise they will be kept. For example:

- First stage simulation:

```
dt = 0.001
app = gala.Application(all_info, dt)
app.add(lj)
app.add(nvt)
app.add(xml)
app.run(1000)
```

- Second state simulation:

```
app.remove(lj)
app.remove(nvt)
app.add(harmonic)
app.add(npt)
app.run(1000)
```

12.3 Two-dimensional simulation

1. Controlling script, i.e. 'file.gala' script is same for two- and three-dimensional simulations.

2. However, configuration file i.e. XML file should indicate two-dimensional system by:

   1. pointing out dimensions with dimensions="2"
   2. setting the length of box in Z diretion to zero with lz="0"
   3. specifying the position of particles in Z direction as 0.0

An example is given:

```
<?xml version="1.0" encoding="UTF-8"?>
<galamost_xml version="1.3">
<configuration time_step="0" dimensions="2" natoms="8" >
<box lx="200" ly="200" lz="0"/>
<position num="8">
    28.5678528848   -37.9327360252     0.0000000000
    28.0019705849   -37.1082499897     0.0000000000
    29.5648198865   -37.8549105956     0.0000000000
    28.1367681830   -38.8350474902     0.0000000000
   -37.5589154370   -72.8549398355     0.0000000000
   -38.4958248509   -72.5053675968     0.0000000000
   -36.7877222908   -72.2183386015     0.0000000000
   -37.3931991693   -73.8411133084     0.0000000000
</position>
</configuration>
</galamost_xml>
```

3. For molgen script to generate a two-dimensional configuration file, a specification of two dimensions and box size in Z direction as 0.0 is necessary. Such as:

```
#!/usr/bin/python
from poetry import molgen

mol=molgen.Molecule(4)
mol.setParticleTypes("A,B,B,B")
mol.setTopology("0-1,0-2,0-3")
mol.setBondLength("A","B", 1.0)
mol.setAngleDegree("B", "A", "B", 120)
mol.setInit("B", 1)
mol.setCris("A", 1)


gen=molgen.Generators(200, 200, 0.0) # box size in X, Y, and Z directions
gen.addMolecule(mol, 2000)
gen.setDimension(2)
gen.setMinimumDistance(1.0)
gen.outPutXML("pn2d")
```

CHAPTER

THIRTEEN

FORCE FIELD

13.1 Short range non-bonded interactions

Overview

The net non-bonded force of each particle is produced by summing all the non-bonded forces of neighboring particles on the basis of a neighbor list that lists the interacting particles for each particle, built beforehand. Because of the independence of parallel CUDA threads, a pair of interacting particles is inevitably included independently in neighbor list in the mode that one thread calculates and sums all non-bonded forces of a particle. The common non-bonded potential energy functions are included.

Lennard-Jones (LJ) interaction	`LjForce`
Shift Lennard-Jones (LJ) interaction	`SljForce`
Linear molecule \pi-\pi interaction	`CenterForce`
Generalized exponential model	`GEMForce`
Lennard-Jones and Ewald (short range) interaction	`LJEwaldForce`
LJ9_6 interaction	`PairForce`
Harmonic repulsion	`PairForce`
Gaussian repulsion	`PairForce`
IPL potential	`PairForce`
Short-range Coulomb potential	`PairForce`

13.1.1 Lennard-Jones (LJ) interaction

Description:

$$\begin{aligned} V_{\mathrm{LJ}}(r) = & \quad 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \alpha \left(\frac{\sigma}{r}\right)^{6} \right] & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- α - *alpha* (unitless)
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to the global r_cut specified in the pair command

class LJForce(*all_info*, *nlist*, *r_cut*)

The constructor of LJ interaction calculation object.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*)

specifies the LJ interaction parameters with type1, type2, epsilon, sigma, and alpha.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*, *float r_cut*)

specifies the LJ interaction parameters with type1, type2, epsilon, sigma, alpha, and cut-off of radius.

setEnergy_shift()

calls the function to shift LJ potential to be zero at cut-off point.

setDispVirialCorr(*bool open*)

switches the dispersion virial correction.

Example:

```
lj = gala.LJForce(all_info, neighbor_list, 3.0)
lj.setParams('A', 'A', 1.0, 1.0, 1.0)
lj.setEnergy_shift()
app.add(lj)       # Note: adds this object to the application.
```

13.1.2 Shift Lennard-Jones (LJ) interaction

Description:

$$\begin{aligned} V_{\mathrm{SLJ}}(r) = & \quad 4\epsilon\left[\left(\frac{\sigma}{r-\Delta}\right)^{12} - \alpha\left(\frac{\sigma}{r-\Delta}\right)^{6}\right] & r < (r_{\mathrm{cut}} + \Delta) \\ = & \quad 0 & r \geq (r_{\mathrm{cut}} + \Delta) \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- α - *alpha* (unitless) - *optional*: defaults to 1.0
- $\Delta = (d_i + d_j)/2 - \sigma$ - (in distance units); d_i and d_j are the diameter of particle i and j which can be input from XML file.
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to the global r_cut specified in the pair command

class SLJForce(*all_info*, *nlist*, *r_cut*)

The constructor of shift LJ interaction calculation object.

Parameters

- **all_info** (AllInfo) – The system information.

- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*)

specifies the shift LJ interaction parameters with type1, type2, epsilon, sigma, and alpha.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*, *float r_cut*)

specifies the shift LJ interaction parameters with type1, type 2, epsilon, sigma, alpha, and cut-off of radius.

setEnergy_shift()

calls the function to shift LJ potential to be zero at the cut-off point.

Example:

```
slj = gala.SLJForce(all_info, neighbor_list, 3.0)
slj.setParams('A', 'A', 1.0, 1.0, 1.0)
slj.setEnergy_shift()
app.add(slj)
```

13.1.3 Linear molecule π-π interaction

An attractive potential to mimic $\pi - \pi$ interactions of rod segments. Reference: Y.-L. Lin, H.-Y. Chang, and Y.-J. Sheng, Macromolecules 2012, 45, 7143-7156.

Description:

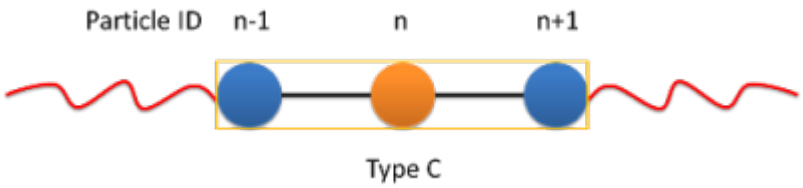
$$\begin{aligned} V_{\pi-\pi}(r,\theta) = & \quad -\epsilon \cos^2\theta(1-r) \quad & r < r_{\mathrm{cut}} \\ = & \quad 0 \quad & r \geq r_{\mathrm{cut}} \end{aligned}$$

- θ - (in radians) the angle between two linear molecules
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to the global r_cut

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)

The transitional forces are added between the center particles of linear molcules. A group of the center particles are needed for *CenterForce*. The rotational forces are added on the two neighbor particles of a center particle.



class CenterForce(*all_info*, *nlist*, *group*, *r_cut*, *epsilon*)

The constructor of a pi-pi interaction calculation object for linear molecules.

Parameters

- **all_info** (AllInfo) – The system information.

- **nlist** (NeighborList) – The neighbor list.
- **group** (ParticleSet) – The group of center particles.
- **r_cut** (*float*) – The cut-off radius.
- **epsilon** (*float*) – the depth of the potential well.

setPreNextShift(*int prev*, *int next*)

sets the previous particle and next particle of center particle with shift ID value, the default value is -1 and 1, respectively.

Example:

```
groupC = gala.ParticleSet(all_info, 'C')
cf = gala.CenterForce(all_info,neighbor_list, groupC, 1.0, 2.0)
app.add(cf)
```

13.1.4 Generalized exponential model

Description:

$\phi(r) = \epsilon \exp\left[-\left(\frac{r}{\sigma}\right)^n\right] r < r_{\mathrm{cut}}$

$= 0 \;\; \mathrm{r} \geq r_{\mathrm{cut}}$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- n - power exponent *n*
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to the global r_cut

class GEMForce(*all_info*, *nlist*, *r_cut*)

The constructor of a generalized exponential model object.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float n*)

specifies the GEM interaction parameters with type1, type2, epsilon, sigma, and n.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float n*, *float r_cut*)

specifies the GEM interaction parameters with type1, type2, epsilon, sigma, n, and cut-off radius.

Example:

```
gem = gala.GEMForce(all_info, neighbor_list, 2.0)
gem.setParams('A', 'A',  1.0,  1.0, 4.0) # epsilon, sigma, n
app.add(gem)
```

13.1.5 Lennard-Jones and Ewald (short range) interaction

Description:

$$V(r_{ij}) = \begin{aligned} & 4\epsilon\left[\left(\frac{\sigma}{r_{ij}}\right)^{12} - \alpha\left(\frac{\sigma}{r_{ij}}\right)^{6}\right] + \frac{f}{\epsilon_r}\frac{q_i q_j \mathrm{erfc}(\kappa r_{ij})}{r_{ij}} & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- α - *alpha* (unitless)
- κ - *kappa* (unitless)
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to the global r_cut specified in the pair command

class LJEwaldForce(*all_info*, *nlist*, *r_cut*)

The constructor of LJ + Ewald in real space interaction calculation object. The κ parameter could be derived automatically.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*)

specifies the LJ interaction parameters with type1, type2, epsilon, sigma, and alpha.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float alpha*, *float r_cut*)

specifies the LJ interaction parameters with type1, type2, epsilon, sigma, alpha, and cut-off of radius.

setEnergy_shift()

calls the function to shift LJ potential to be zero at cut-off point.

setDispVirialCorr(*bool open*)

switches the dispersion virial correction.

Example:

```
lj = gala.LJEwaldForce(all_info, neighbor_list, 0.9)
lj.setParams('OW', 'OW',   0.648520, 0.315365, 1.0)
lj.setParams('HW', 'HW',   0.0, 0.47, 1.0)
lj.setParams('MW',  'MW',  0.0, 0.47, 1.0)

lj.setParams('OW', 'HW',   0.0, 0.47, 1.0)
lj.setParams('OW',  'MW',  0.0, 0.47, 1.0)

lj.setParams('HW',  'MW',  0.0, 0.47, 1.0)
lj.setEnergy_shift()
lj.setDispVirialCorr(True)
app.add(lj)
```

13.1.6 Pair interaction

LJ9_6 interaction

Description:

$$\begin{aligned} V(r) = & \quad 6.75\epsilon \left[\left(\frac{\sigma}{r}\right)^9 - \alpha \left(\frac{\sigma}{r}\right)^6 \right] & r < r_{\text{cut}} \\ = & \quad 0 & r \geq r_{\text{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- α - *alpha* (unitless)
- r_{cut} - *r_cut* (in distance units)

Script commands

Harmonic repulsion

Description:

$$\begin{aligned} V_{\text{harmonic}}(r) = & \quad \tfrac{1}{2}\alpha \left(1 - \frac{r}{r_{cut}}\right)^2 & r < r_{\text{cut}} \\ = & \quad 0 & r \geq r_{\text{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- α - *alpha* (in energy units)
- r_{cut} - *r_cut* (in distance units)

Script commands

Gaussian repulsion

Description:

$$\begin{aligned} V_{\mathrm{Gaussion}}(r) = & \quad \epsilon \exp\left[-\frac{1}{2}\left(\frac{r}{\sigma}\right)^2\right] & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- r_{cut} - *r_cut* (in distance units)

Script commands

IPL potential

Description:

$$\begin{aligned} V_{\mathrm{IPL}}(r) = & \quad \epsilon \left(\frac{\sigma}{r}\right)^n & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- n - *n* (unitless)
- r_{cut} - *r_cut* (in distance units)

Script commands

Short-range Coulomb potential

Description:

$$\begin{aligned} U\left(r\right) = & \quad \frac{\alpha}{r} & r < r_{\mathrm{cut}} \\ = & \quad 0 & r \geq r_{\mathrm{cut}} \end{aligned}$$

The following coefficients must be set per unique pair of particle types:

- $\alpha = f\frac{q_i q_j}{\epsilon_r}$ - *alpha* - (in energy*distance unit): $f = 1/4\pi\epsilon_0 = 138.935\ kJ\ mol^{-1}\ nm\ e^{-2}$
- r_{cut} - *r_cut* (in distance units)

Script commands

Script commands

class PairForce(*all_info*, *nlist*)

The constructor of pair interaction calculation object.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.

setParams(*string type1*, *string type2*, *float param0*, *float param1*, *float param2*, *float r_cut*, *PairFunc function*)

specifies the interaction and its parameters with type1, type2, parameter0, parameter1, parameter2, cut-off radius, and potential type.

setShiftParams(*string type1*, *string type2*, *float param0*, *float param1*, *float param2*, *float r_cut*, *float r_shift*, *PairFunc function*)

specifies the interaction and its parameters with type1, type2, parameter0, parameter1, parameter2, cut-off radius, shift radius, and potential type. This method employs a shift function introduced by GROMACS by which potential and force are smoothed at the boundaries.

Function types	Parameter0	Parameter1	Parameter2
lj12_6	epsilon	sigma	alpha
lj9_6	epsilon	sigma	alpha
harmonic	alpha		
gauss	epsilon	sigma	
ipl	epsilon	sigma	n
Coulomb	alpha		

Example:

```
pair = gala.PairForce(all_info, neighbor_list)
pair.setParams('A', 'A', 100.0, 0.0, 0.0, 1.0, gala.PairFunc.harmonic)
pair.setParams('A', 'B',  10.0, 1.0, 0.0, 1.0, gala.PairFunc.gauss)
pair.setParams('B', 'B',  10.0, 1.0,   2, 1.0, gala.PairFunc.ipl)
app.add(pair)
```

13.2 Bonded interactions

13.2.1 Bond stretching

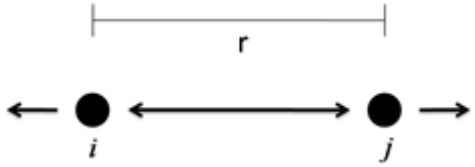
Overview

Bonds impose connected forces on specific pairs of particles to model chemical bonds. The bonds are specified in *XML format* configuration file with the format:

```
<bond>
bond_type(str)  particle_i(int)  particle_j(int)
...
</bond>
```

By themselves, bonds do nothing. Only when you specify a bond force in script(i.e. *BondForceHarmonic*), are forces actually calculated between the listed particles.

Harmonic bond potential	*BondForceHarmonic*
FENE bond potential	BondForceFene
Polynominal bond potential	*BondForcePolynomial*
Morse bond potential	*BondForceMorse*



Harmonic bond potential

Description:

$$V_{\mathrm{bond}}(r) = \frac{1}{2}k\left(r - r_0\right)^2$$

Coefficients:

- k - spring constant **k** (in units of energy/distance^2)
- r_0 - equilibrium length **r0** (in distance units)

class BondForceHarmonic(*all_info*)

The constructor of harmonic bond interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k*, *float r0*)

specifies the bond interaction parameters with bond type, spring constant, and equilibrium length.

Example:

```
bondforce = gala.BondForceHarmonic(all_info)
bondforce.setParams('polymer', 1250.000, 0.470)
app.add(bondforce)
```

FENE bond potential

Description:

$$V_{\mathrm{bond}}(r) = -\frac{1}{2}k{r_m}^2 \log\left[1 - \frac{\left(r - r_0 - \Delta\right)^2}{r_m^2}\right]$$

$$V_{\mathrm{WCA}}(r) = 4\epsilon\left[\left(\frac{\sigma}{r-\Delta}\right)^{12} - \left(\frac{\sigma}{r-\Delta}\right)^{6}\right] + \epsilon \quad ,(r-\Delta) < \sigma^{1/6}$$
$$= 0 \quad ,(r-\Delta) \geq \sigma^{1/6}$$

Coefficients:

- k - attractive force strength `k` (in units of energy/distance^2)
- r_0 - equilibrium length `r0` (in distance units) - *optional*: defaults to 0.0
- r_m - maximum bond length `rm` (in distance units)
- ϵ - *epsilon* (in energy units)
- σ - *sigma* (in distance units)
- $\Delta = (d_i + d_j)/2 - 1.0$ - (in distance units); d_i and d_j are the diameter of particle i and j which can be input from XML file.

Note:

Δ only will be considered (default value is 0.0) by calling the function `setConsiderDiameter(True)`

class BondForceFENE(*all_info*)

The constructor of FENE bond interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k*, *float rm*)

specifies the FENE bond force parameters with bond type, spring constant, and the maximum length of the bond.

setParams(*string type*, *float k*, *float rm*, *float r0*)

specifies the FENE bond force parameters with bond type, spring constant, maximum length, and equilibrium length.

setParams(*string type*, *float k*, *float rm*, *float epsilon*, *float sigma*)

specifies the FENE+WCA bond parameters with bond type, spring constant, maximum length of the bond, epsilon, sigma (the latter two parameters for WCA force between two bonded particles).

setConsiderDiameter(*bool con_dia*)

the diameter of particles will be considered or not

Example:

```
bondforcefene = gala.BondForceFENE(all_info)
bondforcefene.setParams('polymer', 10, 1.2)
app.add(bondforcefene)
```

Polynominal bond potential

Description:

$$V_{\mathrm{bond}}(r) = k_1 \left(r - r_0\right)^2 + k_2 \left(r - r_0\right)^4$$

Coefficients:

- k_1 - spring constant `k1` (in units of energy/distance^2)
- k_2 - spring constant `k2` (in units of energy/distance^4)
- r_0 - equilibrium length `r0` (in distance units)

class BondForcePolynomial(*all_info*)

The constructor of polynomial bond interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k1*, *float k2*, *float r0*)

specifies the polynomial bond force parameters with bond type, spring constant k1, spring constant k2, and equilibrium bond length r0.

Example:

```
bondforce_polynomial = gala.BondForcePolynomial(all_info)
bondforce_polynomial.setParams('polymer', 10.0, 100.0, 1.2)
app.add(bondforce_polynomial)
```

Morse bond potential

Description:

$$\begin{aligned} V_{\mathrm{bond}}(r) = & \quad k\left[1 - e^{-\alpha(r-r_0)}\right]^2 & r < r_{\mathrm{m}} \\ = & \quad 0 & r \geq r_{\mathrm{m}} \end{aligned}$$

Coefficients:

- k - well depth `k` (in units of energy)
- α - controls the 'width' of the potential `alpha` (he smaller α is, the larger the well)
- r_0 - equilibrium length `r0` (in distance units)
- r_m - maximum interaction range `rm` (in distance units)

class BondForceMorse(*all_info*)

The constructor of Morse bond interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string name*, *float k*, *float alpha*, *float r0*, *float rm*)

specifies the Morse bond force parameters with bond type, spring constant, alpha controls the 'width' of the potential, equilibrium bond length, maximum interaction range.

Example:

```
bondforce_morse = gala.BondForceMorse(all_info)
bondforce_morse.setParams('polymer', 10.0, 1.0, 1.0, 2.0)
app.add(bondforce_morse)
```

13.2.2 Angle bending

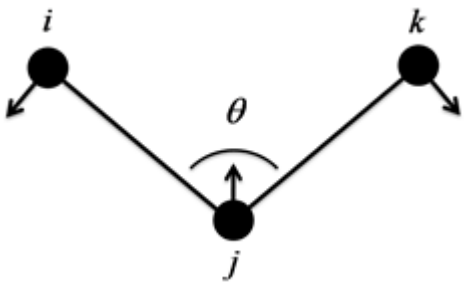
Overview

Angles impose forces on specific triplets of particles to model chemical angles between two bonds. The angles are specified in *XML format* configuration file with the format:

```
<angle>
angle_type(str)  particle_i(int)  particle_j(int)  particle_k(int)
...
</angle>
```

By themselves, angles do nothing. Only when you specify an angle force in script(i.e. *AngleForceHarmonic*), are forces actually calculated between the listed particles.

Harmonic angle potential	*AngleForceHarmonic*
Harmonic cosine angle potential	*AngleForceHarmonicCos*
Cosine angle potential	*AngleForceCos*
LnExp angle potential	*AngleForceLnExp*



Harmonic angle potential

Description:

$$V_{\text{angle}}(\theta) = \frac{1}{2} k \left(\theta - \theta_0\right)^2$$

Coefficients:

- k - potential constant `k` (in units of energy/radians^2)
- θ_0 - equilibrium angle `theta0` (in radians)

Note: The angles set in script are in the unit of degree, and the program will convert them into radian automatically.

class AngleForceHarmonic(*all_info*)

The constructor of angle harmonic interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k*, *float theta0*)

specifies the angle harmonic force parameters with angle type, potential constant, and equilibrium angle degree.

Example:

```
angleforce = gala.AngleForceHarmonic(all_info)
angleforce.setParams('P-G-G', 25.000, 120.000)
app.add(angleforce)
```

Harmonic cosine angle potential

Description:

$$V_{\text{angle}}(\theta) = \frac{1}{2}k\left[\cos\left(\theta\right) - \cos\left(\theta_0\right)\right]^2$$

Coefficients:

- k - potential constant `k` (in units of energy)
- θ_0 - equilibrium angle `theta0` (in radians)

Note: The angles set in script are in the unit of degree, and the program will convert them into radian automatically.

class AngleForceHarmonicCos(*all_info*)

The constructor of angle cosine harmonic interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k*, *float theta0*)

specifies the angle cosine harmonic force parameters with angle type, potential constant, and equilibrium angle degree.

Example:

```
angleforce = gala.AngleForceHarmonicCos(all_info)
angleforce.setParams('P-G-G',25.000, 120.000)
app.add(angleforce)
```

Cosine angle potential

Description:

$$V_{\text{angle}}(\theta) = k\left[1 - \cos\left(\theta - \theta_0\right)\right]$$

Coefficients:

- k - potential constant `k` (in units of energy)
- θ_0 - equilibrium angle `theta0` (in radians)

Note: The angles set in script are in the unit of degree, and the program will convert them into radian automatically.

---

class AngleForceCos(*all_info*)

The constructor of angle cosine interaction object.

param AllInfo all_info
The system information.

setParams(*string type*, *float k*, *float theta0*)

specifies the angle cosine force parameters with angle type, spring constant, and equilibrium angle degree.

Example:

```
angleforce = gala.AngleForceCos(all_info)
angleforce.setParams('P-G-G', 25.000, 120.000)
app.add(angleforce)
```

LnExp angle potential

Description:

$$V_{\text{angle}}(\theta) = -\frac{1}{2}k\log\left[A\exp\left(-k_1\left(\theta-\theta_1\right)^2\right) + B\exp\left(-k_2\left(\theta-\theta_2\right)^2\right)\right]$$

Coefficients:

- k - potential constant `k` (in units of energy)
- k_1, k_2, A, B - potential parameters `k1, k2, A, B`
- θ_1 - equilibrium angle `theta1` (in radians)
- θ_2 - equilibrium angle `theta2` (in radians)

Note: The angles set in script are in the unit of degree, and the program will convert them into radian automatically.

---

class AngleForceLnExp(*all_info*)

The constructor of angle cosine interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string type*, *float k*, *float k1*, *float k2*, *float theta1*, *float theta2*, *float A*, *float B*)

specifies the angle cosine force parameters with: angle type, spring constant, exponential factor1, exponential factor2, equilibrium angle degree1, equilibrium angle degree2, parameter A, and parameter B.

Example:

```
angleforce = gala.AngleForceLnExp(all_info)
angleforce.setParams('P-G-G', 25.000, 1.0, 1.0, 90.0, 180.0, 3.0, 2.0)
app.add(angleforce)
```

13.2.3 Dihedral torsion

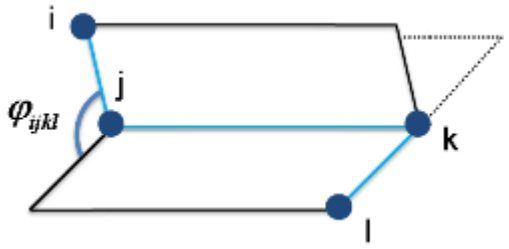
Overview

Dihedrals impose forces on specific quadruplets of particles to model the rotation about chemical bonds. The dihedrals are specified in *XML format* configuration file with the format:

```
<dihedral>
dihedral_type(str)  particle_i(int)  particle_j(int)  particle_k(int)  particle_l(int)
...
</dihedral>
```

By themselves, dihedrals do nothing. Only when you specify a dihedral force in script(i.e. *DihedralForceHarmonic*), are forces actually calculated between the listed particles.

Harmonic dihedral potential (proper)	*DihedralForceHarmonic*
Harmonic dihedral potential (improper)	*DihedralForceHarmonic*
OPLS dihedral potential (proper)	DihedralForceOplsCosine
Ryckaert-Bellemans potential (proper)	*DihedralForceRyckaertBellemans*
Amber potential (proper and improper)	*DihedralForceAmberCosine*



Harmonic dihedral potential (proper)

Description:

$$V_{\text{dihedral}}(\varphi) = k\left[1 + f\cos\left(\varphi - \delta\right)\right]$$

Coefficients:

- k - multiplicative constant `k` (in units of energy)
- δ - phase shift angle `delta` (in radians)
- f - factor `f` (unitless) - *optional*: defaults to *-1.0*

Note: The dihedral angle delta in script are in the unit of degree, and the program will convert them into radian automatically.

`class DihedralForceHarmonic`(*all_info*)

The constructor of dihedral harmonic interaction object.

Parameters
all_info (AllInfo) – The system information.

`setParams`(*string name*, *float k*, *float delta*)

specifies the dihedral harmonic force parameters with dihedral type, multiplicative constant, and phase shift angle.

`setCosFactor`(*float f*)

specifies the dihedral harmonic force parameters with factor.

Example:

```
dihedralforce = gala.DihedralForceHarmonic(all_info)
dihedralforce.setParams('A-B-B-A', 10.0, 0.0)
app.add(dihedralforce)
```

Harmonic dihedral potential (improper)

Description:

$$V_{\text{dihedral}}(\varphi) = k\left(\varphi - \delta\right)^2$$

Coefficients:

- k - potential constant `k` (in units of energy/radians^2)
- δ - phase shift angle `delta` (in radians)

Note: The dihedral angles delta in script are in the unit of degree, and the program will convert them into radian automatically.

---

class DihedralForceHarmonic(*all_info*)

The constructor of dihedral harmonic interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string name*, *float k*, *float delta*)

specifies the dihedral harmonic force parameters with dihedral type, potential constant, and phase shift angle.

setParams(*string name*, *float k*, *float delta*, *int property*)

specifies the dihedral harmonic force parameters with dihedral type, potential constant, phase shift angle, and the property of HarmonicProp::proper or HarmonicProp::improper.

Example:

```
dihedralforce = gala.DihedralForceHarmonic(all_info)
dihedralforce.setParams('A-B-B-A', 10.0, 0.0, gala.HarmonicProp.improper)
app.add(dihedralforce)
```

OPLS dihedral potential (proper)

Description:

$$V_{\text{dihedral}}(\varphi) = k_1 + k_2\left[1+\cos\left(\varphi-\delta\right)\right] + k_3\left[1-\cos\left(2\varphi-2\delta\right)\right] \\ +k_4\left[1+\cos\left(3\varphi-3\delta\right)\right] + k_5\left[1-\cos\left(4\varphi-4\delta\right)\right]$$

Coefficients:

- k_1, k_2, k_3, k_4, k_5 - multiplicative constant `k1, k2, k3, k4, k5` (in units of energy)
- δ - phase shift angle `delta` (in radians)

Note: The dihedral angles delta in script are in the unit of degree, and the program will convert them into radian automatically.

---

class DihedralForceOPLSCosine(*all_info*)

The constructor of dihedral OPLS cosine interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string name*, *float k1*, *float k2*, *float k3*, *float k4*, *float delta*)

specifies the dihedral OPLS cosine force parameters with dihedral type, k1, k2, k3, k4, and phase shift angle. In this function, the default k5 is 0.0.

setParams(*string name*, *float k1*, *float k2*, *float k3*, *float k4*, *loat k5*, *float delta*)

specifies the dihedral OPLS cosine force parameters with dihedral type, k1, k2, k3, k4, k5, and phase shift angle.

Example:

```
dihedralforce = gala.DihedralForceOPLSCosine(all_info)
dihedralforce.setParams('C_33-C_32-C_32-C_32', 0.0, 2.95188, -0.566963, 6.57940, 2.
↪432826, 0.0)
app.add(dihedralforce)
```

Ryckaert-Bellemans potential (proper)

Description:

$$V_{\mathrm{dihedral}}(\varphi) = \sum_{n=0}^{5} C_n \left(\cos\left(\varphi\right)\right)^n$$

Coefficients:

- C_n - multiplicative constant `C0, C1, C2, C3, C4, C5` (in units of energy)

class DihedralForceRyckaertBellemans(*all_info*)

The constructor of dihedral RB interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string name*, *float c0*, *float c1*, *float c2*, *float c3*, *float c4*, *float c5*)

specifies the dihedral RB force parameters with dihedral type, c0, c1, c2, c3, c4, and c5.

Example:

```
pfh = gala.DihedralForceRyckaertBellemans(all_info)
pfh.setParams("A-A-B-B", 30.334, 0.0, -30.334, 0.0, 0.0, 0.0)
app.add(pfh)
```

Amber potential (proper and improper)

Description:

$$\begin{aligned} V_{\mathrm{dihedral}}(\varphi) = k_1\left(1+\cos\left(\varphi-\delta_1\right)\right) + k_2\left(1+\cos\left(2\varphi-\delta_2\right)\right) \\ +k_3\left(1+\cos\left(3\varphi-\delta_3\right)\right) + k_4\left(1+\cos\left(4\varphi-\delta_4\right)\right) \end{aligned}$$

Coefficients:

- k_1, k_2, k_3, k_4 - multiplicative constant `k1, k2, k3, k4` (in units of energy)
- $\delta_1, \delta_2, \delta_3, \delta_4$ - multiplicative constant `delta1, delta2, delta3, delta4` (in units of energy)

Note: The delta1, delta2, delta3, delta4 in script are in the unit of degree, and the program will convert them into radian automatically.

---

class DihedralForceAmberCosine(*all_info*)

The constructor of dihedral Amber interaction object.

Parameters
all_info (AllInfo) – The system information.

setParams(*string name*, *float k1*, *float k2*, *float k3*, *float k4*, *float delta1*, *float delta2*, *float delta3*, *float delta4*, *int property*)

specifies the dihedral Amber force parameters with dihedral type, k1, k2, k3, k4, delta1, delta2, delta3, delta4, and the property of AmberProp::proper or AmberProp::improper.

Example:

```
pfh = gala.DihedralForceAmberCosine(all_info)
pfh.setParams("A-A-B-B", 1.0, 0.0, 0.0, 0.0, 180.0, 0.0, 0.0, 0.0,
          gala.AmberProp.proper)
app.add(pfh)
```

13.3 Numerical interaction

13.3.1 Theory description

The numerical non-bonded, bond, angle, and torsion potentials can be derived from iterative Boltzmann inversion (IBI) or reverse Monte Carlo (RMC) method. With IBI method, the procedure starts with the potentials of mean force as guessed potentials and then optimizes the potentials iteratively by mapping the structural distributions (i.e., radial distribution function, RDF) onto the ones obtained either from atomistic simulations or from experiments. The resulting numerical potentials usually take the form as a table in which the potential values at discrete grid points of distance are given. In the treatment of tabulated potentials, the initial inputted potential tables on grid points of r are transformed to the tables (arrays) on grid points of $z = r^2$. With this trick, the $r = SQRT(r^2)$ in the inner loop of force calculation is avoided, and the force is then calculated by

$$F = -r\frac{\partial V(r)}{\partial r}\frac{1}{r} = -2r\frac{\partial V(z)}{\partial z}$$

Within each interval between the grid points, potentials are fitted to a cubic spline function, more specifically, for each $x_i < x < x_{i+1}$, let $= x - x_i$, $V(x)$ is represented by

$$V(x) = C_0 + C_1\delta + C_2\delta^2 + C_3\delta^3$$

where x corresponds to z, θ, and φ for particle-particle distance square, bending angle, and torsion angle, respectively. i is the index of the grid point and C_0 is the starting potential value of each grid point. Other parameters C_1, C_2, and C_3 are chosen to make the values of the first derivative and the second derivative at both ends of interval x_i and x_{i+1} equal to the correct values of function V. The interval between two adjacent grids $\Delta = x_{i+1} - x_i$ should be equal.

The interaction parameters (C_0, C_1, C_2, C_3) can be read from four columns in a file by function `setParams()` with the formats, such as pair interactions with

Example:

```
<PairForcePoints>
C0  C1  C2  C3
</PairForcePoints>
```

The other node names for bond, angle and dihedral are `BondForcePoints`, `AngleForcePoints`, and `DihedralForcePoints`, respectively.

For convenience, the potentials also can be read directively from two columns in a file by function `setPotential()` with the formats, such as for pair potential

Example:

```
<PairPotential>
r  potential
</ PairPotential >
```

With the potential input format, x corresponds to r for distance and $F = -\partial V(r)/\partial r$. The distance or angle points in first column should be in equal interval and the potentials at the corresponding points are given in second column. The angles θ and φ are in radians. The other node names for bond, angle and dihedral are `BondPotential`, `AnglePotential`, and `Dihedralpotential`, respectively.

13.3.2 Non-bonded interaction

class PairForceTable(*all_info*, *nlist*, *npoint*)

The constructor of an object of numerical pair force calculation.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **npoint** (*int*) – The number of numerical points.

setParams(*string type1*, *string type2*, *float r_cut*, *string& filename*, *int scol*, *int ecol*)

specifies the numerical interaction parameters(C0,C1,C2,C3) with type1, type2, cut-off, inputting file name, start column, end column

setPotential(*string type1*, *string type2*, *std::vector<float2> potential*)

specifies the numerical potential with type1, type2, potential array(r, potential)

setPotential(*string type1*, *string type2*, *string filename*, *int scol*, *int ecol*)

specifies the numerical potential with type1, type2, inputting file name, start column, end column.

Example:

```
pair = gala.PairForceTable(all_info, neighbor_list, 1.3, 2000)
pair.setParams('A', 'A', 1.3, "table.dat", 0, 3)
app.add(pair)
```

13.3.3 Bond interaction

class BondForceTable(*all_info*, *npoint*)

The constructor of an object of numerical bond force calculation.

Parameters

- **all_info** (AllInfo) – The system information.
- **npoint** (*int*) – The number of numerical points.

setParams(*string type*, *float r_cut*, *string filename*, *int scol*, *int ecol*)

specifies the numerical bond interaction parameters(C0,C1,C2,C3) with bond type, cut-off, inputting file name, start column, end column.

setPotential(*string type*, *std::vector<float2> potential*)

specifies the numerical potential with bond type and the array of potential.

setPotential(*string type*, *string filename*, *int scol*, *int ecol*)

specifies the numerical potential with bond type, inputting file name, start column, and end column.

Example:

```
bond = gala.BondForceTable(all_info, 2000)
bond.setParams('1_1', 2.0, "table.dat", 0, 3)
app.add(bond)
```

13.3.4 Angle interaction

class AngleForceTable(*all_info*, *npoint*)

The constructor of an object of numerical angle force calculation.

Parameters

- **all_info** (AllInfo) – The system information.
- **npoint** (*int*) – The number of numerical points.

setParams(*string type*, *string file name*, *int scol*, *int ecol*)

specifies the numerical angle force parameters(C0,C1,C2,C3) with angle type, inputting file name, start column, and end column.

setPotential(*string type*, *std::vector<float2> potential*)

specifies the numerical potential with angle type and the array of potential(r, potential).

setPotential(*string type*, *string filename*, *int scol*, *int ecol*)

specifies the numerical potential with angle type, inputting file name, start column, and end column.

Example:

```
angle = gala.AngleForceTable(all_info, 500)
angle.setParams('111', "table.dat", 0, 3)
app.add(angle)
```

13.3.5 Dihedral interaction

class DihedralForceTable(*all_info*, *npoint*)

The constructor of an object of numerical dihedral force calculation.

Parameters

- **all_info** (AllInfo) – The system information.
- **npoint** (*int*) – The number of numerical points.

setParams(*string type*, *string filename*, *int scol*, *int ecol*)

specifies the numerical dihedral force parameters(C0,C1,C2,C3) with dihedral type, inputting file name, start column, end column.

setPotential(*string dihedral_type*, *std::vector<float2> potential*)

specifies the numerical potential with dihedral type and the array of potential(r, potential).

setPotential(*string dihedral_type*, *string file*, *int scol*, *int ecol*)

specifies the numerical potential with dihedral type, inputting file name, start column, end column.

Example:

```
dihedral = gala.DihedralForceTable (all_info, 500)
dihedral.setParams('111', "table.dat", 0, 3)
app.add(dihedral)
```

13.3.6 Self-defined functions

Numerical module supports self-defined functions with following codes:

```
def pair(width, func, rmin, rmax, coeff):
        ptable = gala.vector_real2()
        dr = rmax/width
        for i in range(0, width):
                r = dr * i
                if r<rmin:
                        potential = func(rmin, **coeff)
                else:
                        potential = func(r, **coeff)
                ptable.append(gala.ToReal2(r, potential))
        return ptable

def bond(width, func, rmin, rmax, coeff):
        ptable = gala.vector_real2()
        dr= rmax/width
        for i in range(0, width):
                r = dr * i
                if r<rmin:
                        potential = func(rmin, **coeff)
                else:
                        potential = func(r, **coeff)
                ptable.append(gala.ToReal2(r, potential))
        return ptable

def angle(width, func, coeff):
        ptable = gala.vector_real2()
        dth = math.pi/width
        for i in range(0, width):
                th = dth * i
                potential = func(th, **coeff)
                ptable.append(gala.ToReal2(th, potential))
        return ptable

def dihedral(width, func, coeff):
        ptable = gala.vector_real2()
        dth = 2.0*math.pi/width
        for i in range(0, width):
                th = dth * i
                potential = func(th, **coeff)
```

(continues on next page)

(continued from previous page)

```
                ptable.append(gala.ToReal2(th, potential))
        return ptable
```

Example for LJ potential:

```
from poetry import numerical

def lj(r, epsilon, sigma):
        v = 4.0 * epsilon * ( (sigma / r)**12 - (sigma / r)**6)
        return v

epsilon0 = 1.0
sigma0 = 1.0

pair = gala.PairForceTable(all_info, neighbor_list,  2000) # (,,the number of␣
↪data points)
pair.setPotential('A', 'A' , numerical.pair(width=2000, func=lj, rmin=0.3,␣
↪rmax=3.0, coeff=dict(epsilon=epsilon0, sigma=sigma0)))
app.add(pair)
# rmin < r to avoid the potential exceeding the upper limit of numerical float.
```

13.4 Coulomb interaction

13.4.1 Ewald summation theory

The Coulomb interaction between two charge particles is given by:

$$U(r) = f\frac{q_i q_j}{\epsilon_r r}$$

where electric conversion factor $f = 1/4\pi\epsilon_0 = 138.935\ kJ\ mol^{-1}\ nm\ e^{-2}$. The total electrostatic energy of N particles and their periodic images is given by

$$V = \frac{f}{2\epsilon_r}\sum_{\mathbf{n}}\sum_{i}^{N}\sum_{j}^{N}\frac{q_i q_j}{|r_{ij}+\mathbf{n}|}$$

The electrostatic potential is practically calculated by

$$U(r^*) = \frac{q_i^* q_j^*}{r^*}$$

The electric conversion factor and relative dielectric constant are considered in the reduced charge. For example, if the mass, length, and energy units are [amu], [nm], and [kJ/mol], respectively, according to *Charge units* the reduced charge is $q^* = z\sqrt{f^*/\epsilon_r}$ with $f^* = 138.935$. The z is the valence of ion.

The calculation of Coulomb interaction is split into two parts, short-range part and long-range part by adding and subtracting a Gaussian distribution.

$$G(r) = \frac{\kappa^3}{\pi^{3/2}} \exp\left(-\kappa^2 r^2\right)$$

The short-range part including *EwaldForce* and *DPDEwaldForce* (for DPD) methods is calculated directly as non-bonded interactions. The long-range part inlcuding *PPPMForce* or *ENUFForce* methods is calculated in the reciprocal sum by Fourier transform.

For Coulomb interaction calculation, a short-range method and a long-range method are both needed.

Example:

```
groupC = gala.ParticleSet(all_info, "charge")

# real space
ewald = gala.EwaldForce(all_info, neighbor_list, groupC, 3.0)#(,,r_cut)
app.add(ewald)

# reciprocal space
pppm = gala.PPPMForce(all_info, neighbor_list, groupC)
pppm.setParams(32, 32, 32, 5, 3.0)
# grid number in x, y, and z directions, spread order, r_cut in real space.
app.add(pppm)

kappa = pppm.getKappa()
# an optimized kappa can be calculated by PPPMForce and passed into EwaldForce.
ewald.setParams(kappa)
```

13.4.2 Ewald (short-range)

Description:

The short-range term is exactly handled in the direct sum.

$$V^S = \frac{f}{2\epsilon_r} \sum_{\mathbf{n}} \sum_{i}^{N} \sum_{j}^{N} \frac{q_i q_j \mathrm{erfc}\left(\kappa \left|r_{ij} + \mathbf{n}\right|\right)}{\left|r_{ij} + \mathbf{n}\right|}$$

The following coefficients must be set:

- κ - *kappa* (unitless)

class EwaldForce(*all_info*, *nlist*, *group*, *r_cut*)

The constructor of an direct Ewald force object for a group of charged particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **group** (ParticleSet) – The group of charged particles.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string typei*, *string typej*, *float kappa*)

specifies the kappa per unique pair of particle types.

setParams(*float kappa*)

specifies the kappa for all pairs of particle types.

Example:

```
group = gala.ParticleSet(all_info, "charge")
kappa=0.8
ewald = gala.EwaldForce(all_info, neighbor_list, group, 3.0)
ewald.setParams(kappa)
app.add(ewald)
```

13.4.3 Ewald for DPD (short-range)

Description:

In order to remove the divergency at $r = 0$, a Slater-type charge density is used to describe the charged DPD particles.

$$\rho(r) = \frac{q}{\pi\lambda^3} e^{-2r/\lambda}$$

- λ - the decay length of the charge (in distance units)

The short-range term is exactly handled in the direct sum.

$$V^S = \frac{f}{2\epsilon_r} \sum_{\mathbf{n}} \sum_{i}^{N} \sum_{j}^{N} \frac{q_i q_j \mathrm{erfc}\left(\kappa \left|r_{ij} + \mathbf{n}\right|\right)}{\left|r_{ij} + \mathbf{n}\right|} \left[1 - \left(1 + \beta r_{ij} \mathrm{e}^{-2\beta r_{ij}}\right]\right.$$

The following coefficients must be set:

- κ - *kappa* (unitless)
- $\beta = 1/\lambda$ - *beta* (in inverse distance units)

class DPDEwaldForce(*all_info*, *nlist*, *group*, *r_cut*)

The constructor of an direct Ewald force object for a group of charged particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **group** (ParticleSet) – The group of charged particles.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string typei*, *string typej*, *float kappa*)

specifies the kappa per unique pair of particle types.

setParams(*float kappa*)

specifies the kappa for all pairs of particle types.

setBeta(*float beta*)

specifies the beta for all pairs of particle types.

Example:

```
group = gala.ParticleSet(all_info, "charge")
kappa=0.8
dpd_ewald = gala.DPDEwaldForce(all_info, neighbor_list, group, 3.0)
dpd_ewald.setParams(kappa)
app.add(dpd_ewald)
```

13.4.4 PPPM (long-range)

Description:

The long-range term is exactly handled in the reciprocal sum.

$$V^L = \frac{1}{2V\epsilon_0\epsilon_r}\sum_{\mathbf{k}\neq 0}\frac{\exp(-\mathbf{k}^2/4\kappa^2)}{\mathbf{k}^2}|S(\mathbf{k})|^2$$

$$S(\mathbf{k}) = \sum_{i=1}^{N} q_i \exp^{i\mathbf{k}\cdot\mathbf{r}_i}$$

The self-energy term.

$$V^{self} = \frac{1}{f}\frac{\kappa}{\sqrt{\pi}}\sum_{i=1}^{N} q_i^2$$

- κ - *kappa* (unitless)

class PPPMForce(*all_info*, *nlist*, *group*)

The constructor of a PPPM force object for a group of charged particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **group** (ParticleSet) – The group of charged particles.

setParams(*int nx*, *int ny*, *int nz*, *int order*, *float r_cut*)

specifies the PPPM force with the number of grid points in x, y, and z direction, the order of interpolation, and the cutoff radius of direct force.

setParams(*float fourierspace*, *int order*, *float r_cut*)

specifies the PPPM force with the fourier space, the order of interpolation, and the cutoff radius of direct force. The number of grid points will be derived automatically.

float getKappa()

return the kappa calculated by PPPM force.

Example:

```
group = gala.ParticleSet(all_info, "charge")
pppm = gala.PPPMForce(all_info, neighbor_list, group)
pppm.setParams(32, 32, 32, 5, 3.0)
app.add(pppm)
```

13.4.5 ENUF (long-range)

class ENUFForce(*all_info*, *nlist*, *group*)

The constructor of an ENUF force object for a group of charged particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **group** (ParticleSet) – The group of charged particles.

setParams(*float alpha*, *float sigma*, *int precision*, *int Nx*, *int Ny*, *int Nz*)

specifies the ENUF force with alpha, hyper sampling factor sigma, precision determine the order of interpolation (precision*2+2), and the number of grid points in x, y, and z direction.

Example:

```
group = gala.ParticleSet(all_info, "charge")
kappa=0.8
enuf = gala.ENUFForce(all_info, neighbor_list, group)
enuf.setParams(kappa, 2.0, 2, 32, 32, 32)
app.add(enuf)
```

13.5 Read force parameters

13.5.1 Force field format

Force field for non-boned interactions:

```
<pair_params>
particle_type1  particle_type2  epsilon  sigma  alpha
</pair_params>
```

For bond, angle, and dihedral interactions:

```
<bond_params>
bond_type  spring_constant  equilibrium_length  function_type
</bond_params>

<angle_params>
angle_type  spring_constant  equilibrium_length  function_type
</angle_params>

<dihedral_params>
dihedral_type  parameter1  parameter2 ...  function_type
</dihedral_params>
```

For bond constraint and virtual site:

```
<constraint_params>
constraint_type  equilibrium_length  function_type
</constraint_params>
```

(continues on next page)

(continued from previous page)

```
<vsite_params>
virtual_site_type  a  b  c  function_type
</vsite_params>
```

An example of force field file:

```
<pair_params>
Qa Qa 2.3 0.6 1.0
Qa Q0 2.0 0.6 1.0
Qa Na 0.5 0.47 1.0
Qa C4 0.5 0.47 1.0
Qa C1 2.0 0.62 1.0
Qa SC3 0.5 0.47 1.0
Qa SC1 2.0 0.62 1.0
Qa SP1c 2.7 0.47 1.0
Q0 Q0 2.0 0.6 1.0
Q0 Na 0.5 0.47 1.0
Q0 C4 0.5 0.47 1.0
Q0 C1 2.0 0.62 1.0
Q0 SC3 0.5 0.47 1.0
Q0 SC1 2.0 0.62 1.0
Q0 SP1c 2.7 0.47 1.0
Na Na 2.3 0.47 1.0
Na C4 2.7 0.47 1.0
Na C1 2.7 0.47 1.0
Na SC3 2.7 0.47 1.0
Na SC1 2.7 0.47 1.0
Na SP1c 2.3 0.47 1.0
C4 C4 4.5 0.47 1.0
C4 C1 4.0 0.47 1.0
C4 SC3 4.5 0.47 1.0
C4 SC1 4.0 0.47 1.0
C4 SP1c 2.7 0.47 1.0
C1 C1 4.5 0.47 1.0
C1 SC3 4.5 0.47 1.0
C1 SC1 4.5 0.47 1.0
C1 SP1c 2.3 0.47 1.0
SC3 SC3 3.4 0.43 1.0
SC3 SC1 3.4 0.43 1.0
SC3 SP1c 2.7 0.47 1.0
SC1 SC1 3.4 0.43 1.0
SC1 SP1c 2.3 0.47 1.0
SP1c SP1c 2.3 0.47 1.0
</pair_params>

<constraint_params>
SP1c-SC3 0.4904 1
SP1c-SC1 0.6019 1
SC3-SC1 0.2719 1
SC1-SC3 0.7237 1
SC1-SC1 0.5376 1
```

(continues on next page)

(continued from previous page)

```
</constraint_params>

<vsite_params>
SC1-SC1-SC3-SC1 0.9613 0.6320 0.0 1
SC1-SC3-SP1c-SC1 0.5207 0.2882 -1.03168 4
SC1-SC1-SC3-SC1_1 0.2287 0.4111 1.41920 4
</vsite_params>

<bond_params>
Q0-Qa 1250.0 0.450 1
Qa-Na 1250.0 0.450 1
Na-Na 1250.0 0.370 1
Na-C1 1250.0 0.480 1
C1-C1 1250.0 0.480 1
C1-C4 1250.0 0.480 1
C4-C4 1250.0 0.480 1
C4-C1 1250.0 0.480 1
SC1-C1 1250.0 0.425 1
</bond_params>

<angle_params>
Qa-Na-Na 25.0 120.000 2
Qa-Na-C1 25.0 180.000 2
Na-C1-C1 35.0 180.000 2
C1-C1-C1 35.0 180.000 2
Na-C1-C4 35.0 180.000 2
C1-C4-C4 20.0 95.000 2
C4-C4-C1 45.0 120.000 2
SC1-SC1-C1 25.0 180.0 2
</angle_params>

<dihedral_params>
SP1c-SC3-SC1-SC1_F2 -179.7 50.0 2
</dihedral_params>
```

13.5.2 Use force fields

Description:

Force fields in the format could be read by `force_field_gala` module. The classes of `force_field_gala` module are listed as following.

class LJCoulombShiftForce(*all_info*, *nlist*, *rcut*, *rshift*, *epsilon_r*, *file*)

Constructor of an object to simutaneously calculate modified Lennard-Jones and Coulomb interactions which are smoothed by a shift function same to GROMACS.

Parameters

- **all_info** (AllInfo) – System information.
- **nlist** (NeighborList) – Neighbor list.
- **rcut** (*float*) – Cut-off radius.
- **rshift** (*float*) – Shift radius.

- **epsilon_r** (*float*) – Relative dielectric constant.
- **file** (*string*) – Force field file.

Example:

```
import force_field_gala
e_r = 15.0
lj = force_field_gala.LJCoulombShiftForce(all_info, nlist, 1.2, 0.9, e_r, "Equ.
↪force_field")
app.add(lj)
```

class LJEwaldForce(*all_info*, *nlist*, *rcut*, *file*)

Constructor of an object to simutaneously calculate Lennard-Jones and the short-part Coulomb interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **nlist** (NeighborList) – Neighbor list.
- **rcut** (*float*) – Cut-off radius.
- **file** (*string*) – Force field file.

setEnergy_shift()

calls the function to shift LJ potential to be zero at cut-off point.

setDispVirialCorr(*bool open*)

switches the dispersion virial correction.

Example:

```
import force_field_gala
lj = force_field_gala.LJEwaldForce(all_info, neighbor_list, 1.0, "ffnonbonded.force_
↪field")
lj.setEnergy_shift()
lj.setDispVirialCorr(True)#dispersion virial correction
app.add(lj)
```

class BondForceHarmonic(*all_info*, *file*)

Constructor of an object to calculate harmonic bond interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
bondforce = force_field_gala.BondForceHarmonic(all_info, "ffbonded.force_field")
app.add(bondforce)
```

class AngleForceHarmonicCos(*all_info*, *file*)

Constructor of an object to calculate harmonic cosine angle interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
angleforce = force_field_gala.AngleForceHarmonicCos(all_info, "ffbonded.force_field
↪")
app.add(angleforce)
```

class AngleForceHarmonic(*all_info*, *file*)

Constructor of an object to calculate harmonic angle interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
angleforce = force_field_gala.AngleForceHarmonic(all_info, "ffbonded.force_field")
app.add(angleforce)
```

class DihedralForceAmberCosine(*all_info*, *file*)

Constructor of an object to calculate Amber cosine dihedral interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
dihedralforce = force_field_gala.DihedralForceAmberCosine(all_info, "ffbonded.force_
↪field")
app.add(dihedralforce)
```

class DihedralForceHarmonic(*all_info*, *file*)

Constructor of an object to calculate harmonic dihedral interactions.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
dihedralforce = force_field_gala.DihedralForceHarmonic(all_info, "ffbonded.force_
↪field")
app.add(dihedralforce)
```

class BondConstraint(*all_info*, *file*)

Constructor of an object to implement bond constraints.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

setNumIters(*int ncycles*)

specifies the number of iterations of calcuation.

setExpansionOrder(*int order*)

specifies the spread order.

Example:

```
bond_constraint = force_field_gala.BondConstraint(all_info, "Equ.force_field")
bond_constraint.setExpansionOrder(4)
bond_constraint.setNumIters(1)
app.add(bond_constraint)
```

class Vsite(*all_info*, *file*)

Constructor of an object to implement virtual sites using a same method to GROMACS.

Parameters

- **all_info** (AllInfo) – System information.
- **file** (*string*) – Force field file.

Example:

```
vs = force_field_gala.Vsite(all_info, "Equ.force_field")
app.add(vs)
```

13.5.3 Use GROMACS force fields

Description:

Force fields in GROMACS format are supported by `force_field_itp` module. The usage and methods are same to `force_field_gala` module, but for reading the force fields in GROMACS format from itp files.

An example:

```
import force_field_itp
lj = force_field_itp.LJEwaldForce(all_info, neighbor_list, 1.0, "ffnonbonded.itp")
lj.setEnergy_shift()
lj.setDispVirialCorr(True)#dispersion virial correction
app.add(lj)
```

13.5.4 Convert GROMACS files

Description:

Convert GROMACS files to GALA files including configuration and force fields by `gro_to_xml` module. Execution command is `python gro_to_xml.py` with two necessary parameters `--gro=` and `--top=` to set the GROMACS file names.

An example:

```
python gro_to_xml.py --top=Topol.top --gro=Equ.gro
```

Then two files 'Equ.xml' of configuration and 'Equ.force_field' of force field will be generated.

CHAPTER

FOURTEEN

INTEGRATION

14.1 NVE ensemble

Overview

NVE thermostat	`NVE`
NVE for rigid body	`NVERigid`
NVE for rigid body with tunable freedoms	`TranRigid`

14.1.1 NVE thermostat

class NVE(*all_info*, *group*)

The constructor of a NVE thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setZeroForce(*bool switch*)

switches the function of making all force to be zero (the default is False).

Example:

```
group = gala.ParticleSet(all_info,'all')
thermo = gala.NVE(all_info,group)
app.add(thermo)
```

14.1.2 NVE for rigid body

class NVERigid(*all_info*, *group*)

The constructor of a NVE thermostat object for rigid bodies.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

Example:

```
bgroup = gala.ParticleSet(all_info, 'body')
rigidnve = gala.NVERigid(all_info, bgroup)
app.add(rigidnve)
```

14.1.3 NVE for rigid body with tunable freedoms

class TranRigid(*all_info*, *group*)

The constructor of a NVE thermostat object for rigid bodies for defined freedoms.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setTraDimension(*bool x*, *bool y*, *bool z*)

switches the freedoms of translocation in x y z directions.

setRotDimension(*bool x*, *bool y*, *bool z*)

switches the freedoms of rotation in x y z directions.

Example:

```
rigidnve = gala.TranRigid (all_info, bgroup)
rigidnve.setTraDimension(True, True, True)
rigidnve.setRotDimension(True, True, True)
app.add(rigidnve)
```

14.2 NVT ensemble

Overview

Nose Hoover thermostat	*NoseHooverNVT*
Berendsen thermostat	*BerendsenNVT*
Andersen thermostat	*AndersenNVT*
Langevin dynamic thermostat	*LangevinNVT*
NVT for rigid body	*NVTRigid*
Langevin dynamic for rigid body	*LangevinNVTRigid*

14.2.1 Nose Hoover thermostat

class NoseHooverNVT(*all_info*, *group*, *comp_info*, *T*, *tauT*)

The constructor of a NVT NoseHoover thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info** (ComputeInfo) – The object of calculation of collective information.

- **T** (*float*) – The temperature.
- **tauT** (*float*) – The thermostat coupling.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a defined varying value by time step.

Example:

```
group = gala.ParticleSet(all_info, 'all')
comp_info = gala.ComputeInfo(all_info, group)
nh = gala.NoseHooverNVT(all_info, group, comp_info, 1.0, 0.5)
app.add(nh)
```

14.2.2 Berendsen thermostat

class BerendsenNVT(*all_info*, *group*, *comp_info*, *T*, *tauT*)

The constructor of a NVT Berendsen thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info** (ComputeInfo) – The object of calculation of collective information.
- **T** (*float*) – The temperature.
- **tauT** (*float*) – The thermostat coupling parameter.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time steps.

14.2.3 Andersen thermostat

class AndersenNVT(*all_info*, *group*, *T*, *gamma*, *seed*)

The constructor of a NVT Andersen thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **T** (*float*) – The temperature.
- **gamma** (*float*) – The collision frequency.
- **seed** (*int*) – The seed of random number generator.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time steps.

Example:

```
an = gala.AndersenNVT(all_info,group,1.0,10.0, 12345)
app.add(an)
```

14.2.4 Langevin dynamic thermostat

Description:

The particles are integrated forward in time according to the Langevin equations of motion:

$$m\frac{d\vec{v}}{dt} = \vec{F}_{\mathrm{C}} - \gamma \cdot \vec{v} + \vec{F}_{\mathrm{R}}$$
$$\langle \vec{F}_{\mathrm{R}} \rangle = 0$$
$$\langle |\vec{F}_{\mathrm{R}}|^2 \rangle = 2dkT\gamma/\delta t$$

- γ - *gamma* (unitless) - *optional*: defaults to 1.0

where $\vec{F}_{\mathrm{C}}$ is the force on the particle from all potentials and constraint forces, γ is the drag coefficient, $\vec{v}$ is the particle's velocity, $\vec{F}_{\mathrm{R}}$ is a uniform random force, and d is the dimensionality of the system (2 or 3). The magnitude of the random force is chosen via the fluctuation-dissipation theorem to be consistent with the specified drag and temperature, T. When $kT = 0$, the random force $\vec{F}_{\mathrm{R}} = 0$.

class LangevinNVT(*all_info*, *group*, *T*, *seed*)

The constructor of a Langevin NVT thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **T** (*float*) – The temperature.
- **seed** (*int*) – The seed of random number generator.

setGamma(*float gamma*)

specifies the gamma with a constant value.

setGamma(*string type*, *float gamma*)

specifies the gamma of a particle type.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

Example:

```
group = gala.ParticleSet(all_info, 'all')
lnvt = gala.LangevinNVT(all_info, group, 1.0, 123)
app.add(lnvt)
```

14.2.5 NVT for rigid body

class NVTRigid(*AllInfo all_info*, *ParticleSet group*, *float T*, *float tauT*)

The constructor of a NVT thermostat object for rigid bodies.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **T** (*float*) – The temperature.
- **tauT** (*float*) – The thermostat coupling parameter.

setT(*float T*)

specifies the temperature with a fixed value.

setT(*Variant vT*)

pecifies the temperature with a varying value by time step.

Example:

```
bgroup = gala.ParticleSet(all_info, 'body')
rigidnvt = gala.NVTRigid(all_info, bgroup, 1.0, 10.0)
app.add(rigidnvt)
```

14.2.6 Langevin dynamic for rigid body

Please see *Langevin dynamic thermostat* for the theory.

class LangevinNVTRigid(*all_info*, *group*, *T*, *seed*)

The constructor of a Langevin NVT thermostat object for rigid bodies.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **T** (*float*) – The temperature.
- **seed** (*int*) – The seed of random number generator.

setGamma(*float gamma*)

specifies the gamma of Langevin method with a constant value.

setGamma(*const std::string & type*, *float gamma*)

specifies the gamma of Langevin method of a particle type.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

Example:

```
bgroup = gala.ParticleSet(all_info, 'body')
lrigidnvt = gala.LangevinNVTRigid(all_info, bgroup, 1.0, 123)
app.add(lrigidnvt)
```

14.3 NPT ensemble

Overview

Andersen barostat	*NPT*
NPT for rigid body	*NPTRigid*

14.3.1 Andersen barostat

Reference: H. C. Andersen, J. Chem. Phys., 1980, 72(4), 2384-2393.

class NPT(*all_info*, *group*, *comp_info_group*, *comp_info_all*, *T*, *P*, *tauT*, *tauP*)

The constructor of a NPT thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info_group** (ComputeInfo) – The calculation of collective information of group particles.
- **comp_info_all** (ComputeInfo) – The calculation of collective information of all particles.
- **T** (*float*) – The temperature.
- **P** (*float*) – The pressure.
- **tauT** (*float*) – The thermostat coupling.
- **tauP** (*float*) – The barostat coupling.

setP(*float P*)

specifies the pressure with a constant value.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

Example:

```
npt =gala.NPT(all_info, group, comp_info, comp_info, 1.0, 0.2, 0.5, 0.1)
app.add(npt)
```

14.3.2 NPT for rigid body

class NPTRigid(*all_info*, *group*, *comp_info_group*, *comp_info_all*, *T*, *P*, *tauT*, *tauP*)

The constructor of a NPT thermostat object for rigid bodies.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info_group** (ComputeInfo) – The calculation of collective information of group particles.
- **comp_info_all** (ComputeInfo) – The calculation of collective information of all particles.
- **T** (*float*) – The temperature.
- **P** (*float*) – The pressure.
- **tauT** (*float*) – The thermostat coupling.
- **tauP** (*float*) – The barostat coupling.

setT(*float T*)

specifies the temperature with a fixed value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

setP(*float P*)

specifies the pressure with a fixed value.

Example:

```
group = gala.ParticleSet(all_info,'all')
comp_info = gala.ComputeInfo(all_info, group)

bgroup = gala.ParticleSet(all_info, 'body')
comp_info_b = gala.ComputeInfo(all_info, bgroup)

rigidnpt = gala.NPTRigid(all_info, bgroup, comp_info_b, comp_info, 1.0, 0.1, 1.0, 1.
↪0)
app.add(rigidnpt)
```

14.3.3 Martyna-Tobias-Klein barostat

Reference: G. J. Martyna, D. J. Tobias, and M. L. Klein, J. Chem. Phys., 1994, 101(5), 4177-4189.

class NPTMTK(*all_info*, *group*, *comp_info_group*, *comp_info_all*, *T*, *P*, *tauT*, *tauP*)

The constructor of a NPTMTK thermostat object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info_group** (ComputeInfo) – The calculation of collective information of group particles.

- **comp_info_all** (ComputeInfo) – The calculation of collective information of all particles.
- **T** (*float*) – The temperature.
- **P** (*float*) – The pressure.
- **tauT** (*float*) – The thermostat coupling.
- **tauP** (*float*) – The barostat coupling.

setT(*float T*)

specifies the temperature with a fixed value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

setSemiisotropic(*float pressxy*, *float pressz*)

specifies the pressure with fixed values for XY and Z directions, respectively.

setSemiisotropic(*float pressxy*, *Variant vpressz*)

specifies the pressure with a fixed value for XY direction and a varying value for Z direction, respectively.

setAnisotropic(*float pressx*, *float pressy*, *float pressz*)

specifies the pressure with fixed values for X, Y and Z directions, respectively.

Example:

```
group = gala.ParticleSet(all_info,'all')
comp_info = gala.ComputeInfo(all_info, group)

npt = gala.NPTMTK(all_info, group, comp_info, comp_info, 1.0, 0.1, 0.5, 1.0)
npt.setSemiisotropic(0.1, 0.1)
app.add(npt)
```

14.3.4 Martyna-Tobias-Klein barostat for rigid body

Reference: G. J. Martyna, D. J. Tobias, and M. L. Klein, J. Chem. Phys., 1994, 101(5), 4177-4189.

class NPTMTKRigid(*all_info*, *group*, *comp_info_group*, *comp_info_all*, *T*, *P*, *tauT*, *tauP*)

The constructor of a NPTMTK thermostat object for rigid bodies.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info_group** (ComputeInfo) – The calculation of collective information of group particles.
- **comp_info_all** (ComputeInfo) – The calculation of collective information of all particles.
- **T** (*float*) – The temperature.
- **P** (*float*) – The pressure.
- **tauT** (*float*) – The thermostat coupling.
- **tauP** (*float*) – The barostat coupling.

setT(*float T*)

specifies the temperature with a fixed value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

setSemiisotropic(*float pressxy*, *float pressz*)

specifies the pressure with fixed values for XY and Z directions, respectively.

setSemiisotropic(*float pressxy*, *Variant vpressz*)

specifies the pressure with a fixed value for XY direction and a varying value for Z direction, respectively.

setAnisotropic(*float pressx*, *float pressy*, *float pressz*)

specifies the pressure with fixed values for X, Y and Z directions, respectively.

Example:

```
group = gala.ParticleSet(all_info,'all')
comp_info = gala.ComputeInfo(all_info, group)

groupb = gala.ParticleSet(all_info, 'body')
comp_info_b = gala.ComputeInfo(all_info, groupb)

npt = gala.NPTMTKRigid(all_info, groupb, comp_info_b, comp_info, 1.0, 0.1, 0.5, 1.0)
npt.setSemiisotropic(0.1, 0.1)
app.add(npt)
```

CHAPTER

FIFTEEN

CONSTRAINT

15.1 Variant

15.1.1 Variant Const

class VariantConst(*value*)

The constructor of a constant value method.

Parameters
value (*float*) – The constant value.

Example:

```
v = gala.VariantConst(1.0)
# set the constant value.
```

15.1.2 Variant Linear

class VariantLinear

The constructor of a linearly varying value method.

setPoint(*unsigned int timestep*, *double value*)

specifies the value at the time step.

Example:

```
v = gala.VariantLinear()
v.setPoint(0, 1.0)
v.setPoint(100000, 2.0)
# set the value at the time step. The value at a time step
# varies by linear interpolation.
```

15.1.3 Variant Sin

class VariantSin

The constructor of a sinusoidal curve varying object.

setPoint(*unsigned int timestep*, *double period*, *double ubd*, *double lbd*)

Function: specifies the period, upper, and lower bounds at the time step.

Example:

```
v = gala.VariantSin()
v.setPoint(0, 1000, 1.0, -1.0)
v.setPoint(100000, 1000, 2.0, -2.0)
# set the parameters of sinusoid at the time step and the parameters
# at any time step can be gotten by linear interpolation.
```

15.1.4 Variant Well

class VariantWell

The constructor of a well curve varying object.

setPoint(*unsigned int timestep*, *double period*, *double ubd*, *double lbd*)

specifies the period, upper, and lower bounds at the time step.

Example:

```
v = gala.VariantWell()
v.setPoint(0, 1000, 1.0, -1.0)
v.setPoint(100000, 1000, 1.0, -1.0)
# set the parameters of periodic well at the time step and the parameters
# at any time step can be gotten by linear interpolation.
```

15.2 Space constraint

15.2.1 Bounce back condition

class BounceBackConstrain(*all_info*, *group*)

The constructor of a bounce back wall object with a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of charged particles.

addWall(*float o_x*, *float o_y*, *float o_z*, *float d_x*, *float d_y*, *float d_z*)

add wall with original point(o_x, o_y, o_z) and normal direction(d_x, d_y, d_z).

addCylinder(*float o_x*, *float o_y*, *float o_z*, *float d_x*, *float d_y*, *float d_z*, *float r*)

add cylinder with original point (o_x, o_y, o_z) , axis direction (d_x, d_y, d_z) and radius.

addSphere(*float o_x*, *float o_y*, *float o_z*, *float r*)

sphere with center point(o_x, o_y, o_z) and radius.

clcearWall()

clear the walls.

clearCylinder()

clear the cylinders

clearSphere()

clear the spheres.

Example:

```
bbc = gala.BounceBackConstrain(all_info, group)
bbc.addWall(0.0, 10.0, 0.0, 0.0, 1.0, 0.0)
bbc.addWall(0.0, -10.0, 0.0, 0.0, 1.0, 0.0)
app.add(bbc)
```

15.2.2 LJ surface force

class LJConstrainForce(*all_info*, *group*, *r_cut*)

The constructor of a LJ interaction surface object for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of charged particles.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type*, *float epsilon*, *float sigma*, *float alpha*)

sets the interaction parameters of particle type for LJ surface.

addWall(*float o_x*, *float o_y*, *float o_z*, *float d_x*, *float d_y*, *float d_z*)

adds wall with original point(o_x, o_y, o_z) and normal direction(d_x, d_y, d_z)

addCylinder(*float o_x*, *float o_y*, *float o_z*, *float d_x*, *float d_y*, *float d_z*, *float r*)

adds cylinder with original point(o_x, o_y, o_z) ,axis direction(d_x, d_y, d_z), and radius.

addSphere(*float o_x*, *float o_y*, *float o_z*, *float r*)

adds sphere with center point(o_x, o_y, o_z) and radius.

clcearWall()

clear the walls.

clearCylinder()

clear the cylinders

clearSphere()

clear the spheres.

Example:

```
ljc = gala.LJConstrainForce(all_info, group, 1.0)
ljc.addWall(0.0, 10.0, 0.0, 0.0, 1.0, 0.0)
ljc.addWall(0.0, -10.0, 0.0, 0.0, 1.0, 0.0)
ljc.setParams("A", 1.0, 1.0, 1.0)
app.add(ljc)
```

15.3 Remove CM momentum

class ZeroMomentum(*all_info*)

The constructor of an object of removing the momentum of center mass of all particles.

Parameters

all_info (AllInfo) – The system information.

class ZeroMomentum(*all_info*, *group*)

specifies the method of removing the momentum of center mass of a group of particles.

Parameters

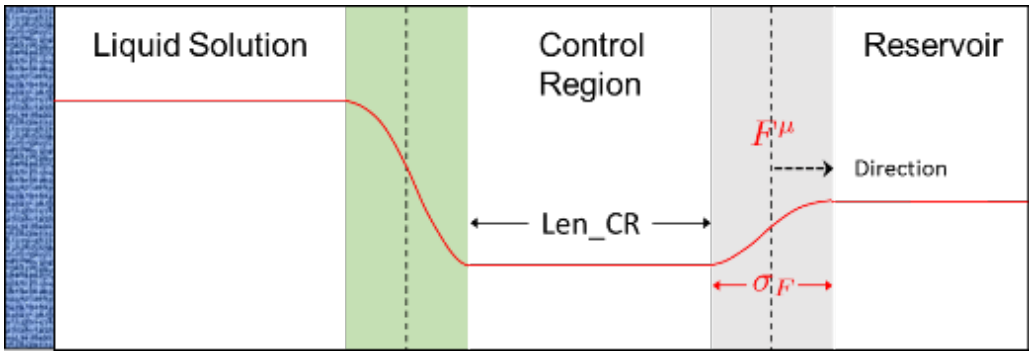
- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of charged particles.

Example:

```
zm = gala.ZeroMomentum(all_info)
zm.setPeriod(10)
app.add(zm)
```

15.4 Constant chemical potential

Constant Chemical Potential Molecular Dynamics (C μ MD) method introduces an external force that controls the environment of the chemical process of interest. This external force, drawing molecules from a finite reservoir, maintains the chemical potential constant in the region where the process takes place. This method is able to study crystal growth dynamics under constant supersaturation conditions or evaporation dynamics under constant vapor pressure. Reference: C. Perego, M. Salvalaglio, and M. Parrinello, J. Chem. Phys., 2015, 142, 144113.



Description:

$$F^{\mu}(z) = k(n^{CR} - n_0)G(z, Z_F)$$
$$G(z - Z_F) = \frac{1}{4\omega}\left[1 + cosh\left(\frac{z - Z_F}{\omega}\right)\right]^{-1}$$

Coefficients:

- n_0 - target constant concentration `n0` (in reduced units)
- k - spring constant `k` (in units of energy/distance^2)
- σ - width of external force region `sigma` (in units of distance)
- ω - an intensity peak proportional to `1/omega` and a width proportional to `omega` (in units of distance)

class CCPMD(*all_info*, *group*)

The constructor of a constant chemical potential object of a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setWall(*float ox*, *float oy*, *float oz*, *float dx*, *float dy*, *float dz*)

specifies external force region with plane center (ox, oy, oz) and direction (dx, dy, dz). If the normal direction of wall is in Z direction, the center position of plane is (0.0, 0.0, Z_F).

setParams(*float n0*, *float k*, *float sigma*, *float omega*, *float Len_CR*)

specifies target concentration, spring constant, sigma, omega, and the length of control region.

Example:

```
groupS = gala.ParticleSet(all_info, 'S')
ccp = gala.CCPMD(all_info, groupS)
ccp.setWall(0.0, 0.0, -25.0, 0.0, 0.0, -1.0)
ccp.setParams(0.5, 1000.0, 1.0, 0.1, 5.0)
app.add(ccp)
```

15.5 Bond constraint

Description:

LINCS algorithm

class BondConstraint(*all_info*)

Constructor of a bond bonstraint object.

Parameters

all_info (AllInfo) – System information.

setParams(*string type*, *float k*, *float r0*)

specifies the bond constraint parameters with bond type and equilibrium length.

setNumIters(*int ncycles*)

specifies the number of iterations of calcuation.

setExpansionOrder(*int order*)

specifies the spread order.

Example:

```
bondconstraint = gala.BondConstraint(all_info)# bond constraints using LINCS␣
↪algorithm
bondconstraint.setParams('oh', 0.09572)#(type, r0)
bondconstraint.setParams('hh', 0.15139)#(type, r0)
bondconstraint.setExpansionOrder(4)
bondconstraint.setNumIters(1)
app.add(bondconstraint)
```

15.6 Virtual site

Description:

GROMACS method

class Vsite(*all_info*)

Constructor of a virtual site object.

Parameters
all_info (AllInfo) – System information.

setParams(*string type*, *float a*, *float b*, *float c*, *VST vst*)

specifies the virtual site parameters with a, b, c, and virtual site type. The candidates of VST are 'v2', 'v3', 'v3fd', 'v3fad', 'v3out', and 'v4fdn'.

Example:

```
vs = gala.Vsite(all_info)#virtual interaction sites
vs.setParams('v', 0.128012065, 0.128012065, 0.0, gala.VST.v3 )
app.add(vs)
```

CHAPTER

SIXTEEN

EXTERNAL FIELD

16.1 External force

class ExternalForce(*all_info*, *group*)

The constructor of an external force object for a group of particles. The external force will be added on each particle.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setForce(*Variant vf*, *std::string direction*)

specifies the force magnitude varying by time steps and direction (the candidates are "X", "Y", and "Z").

setForce(*Variant vf*, *float x*, *float y*, *float z*)

specifies the force magnitude varying by time steps and direction vector (x, y, z).

setParams(*string type*, *float factor*)

specifies the factor of external force for a particle type (the default value is 1.0).

setParams(*unsigned int index*, *float factor*)

specifies the factor of external force for a particle with index.

Example:

```
v = gala.VariantSin()
v.setPoint(0, 1000, 1, -1)
v.setPoint(1000000, 1000, 1, -1)
# set the parameters of sinusoid force by time step, period, max and min value where
    # the latter three parameters are linearly varying by time step.

groupA = gala.ParticleSet(all_info, "A")
ef = gala.ExternalForce(all_info, groupA)
#initializes an external force object with system information and particle group.
ef.setForce(v, "X")
# sets parameters with force and direction.
app.add(ef)
```

16.2 Axial stretching

class AxialStretching(*all_info*, *group*)

The constructor of a stretching object of the box for a group of particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setBoxLength(*Variant vL*, *string direction*)

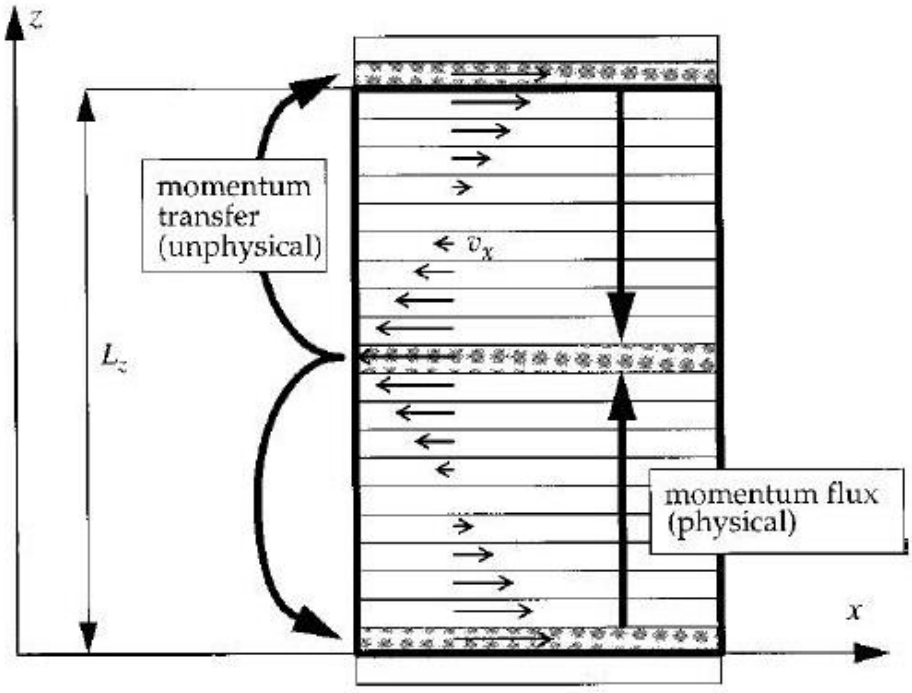
specifies the change of box length and its direction with Variant. The candidates are "X", "Y", "Z".

Example:

```
v = gala.VariantLinear()
v.setPoint(0, 31) # time step, box length.
v.setPoint(100000, 60)

axs = gala.AxialStretching(all_info, group)
axs.setBoxLength(v, 'Y')
app.add(axs)
```

16.3 RNEMD

A nonequilibrium method for calculating the shear viscosity is presented. It reverses the cause-and-effect picture customarily used in nonequilibrium molecular dynamics: the effect, the momentum flux or stress, is imposed, whereas the cause, the velocity gradient or shear rate, is obtained from the simulation. Reference: F. Müller-Plathe, Phys. Rev. E 59, 4894, 1999.



class RNEMD(*all_info*, *np_per_swap*, *nslabs*, *filename*)

The constructor of a RNEMD object to implement a shear field.

Parameters

- **all_info** (AllInfo) – The system information.
- **np_per_swap** – The number of particles selected for velocity swap.
- **nslabs** – The number of divided slabs of simulation box.

Filename

The name of file for data output including velocity profile, momentum plus and viscosity.

setSwapPeriod(*unsigned int swap_period*)

specifies the period of particle velocity swap.

setProfVelPeriod(*unsigned int profvel_period*)

specifies the period of profiling velocity of slabs, the default value is 10.

setSwapGroup(*ParticleSet group*)

specifies the group of particles for velocity swap.

setVelProfile(*bool vel_profile*)

True or False for profiling velocity of slabs.

setPeriod(*unsigned int period*)

specifies the period of data output in which the velocity profile and momentum plus will be averaged.

Example:

```
rnemd = gala.RNEMD(all_info, 1, 30, 'velocity_profile.data')
rnemd.setSwapPeriod(50)
```

(continues on next page)

(continued from previous page)

```
rnemd.setSwapGroup(group)
rnemd.setVelProfile(True)
rnemd.setPeriod(10000)
app.add(rnemd)
```

CHAPTER

SEVENTEEN

MODULES

17.1 MD-SCF

17.1.1 MDSCF force

Description:

Using hybrid particle-field technique to accelerate CGMD simulations (G. Milano and T. Kawakatsu, J. Chem. Phys. 130, 214106, 2009). This method could largely speed up some slowly evolving processes in CGMD simulations, such as microphase separation and self-assembly of polymeric systems.

class MDSCFForce(*AllInfo all_info*, *int nx*, *int ny*, *int nz*, *float comp*)

Constructor of an object of MD-SCF force.

Parameters

- **all_info** (AllInfo) – System information.
- **nx** (*int*) – Number of grid in x direction.
- **ny** (*int*) – Number of grid in y direction.
- **nz** (*int*) – Number of grid in z direction.
- **comp** (*float*) – Compressibility.

setPeriodScf(*int idl2_step*, *int idl_step*)

sets the periods of computing density field and updating density field.

setParams(*string type1*, *string type2*, *float chi*)

specifies the MD-SCF interaction parameters by pair types with type1, type2, chi parameter.

setNewVersion(*bool switch*)

switches the function of newly developed method of implementation.

Example:

```
scf = gala.MDSCFForce(all_info, 22, 22, 22, 0.100)
scf.setParams('N', 'N',0.000)
scf.setParams('N', 'P',-1.500)
scf.setParams('P', 'P',0.000)
scf.setPeriodScf(1,300)
scf.setNewVersion(True)
# switches to newly developed version.
app.add(scf)
# adds this object to the application.
```

17.1.2 MDSCF electrostatic force

class PFMEForce(*AllInfo all_info*, *int nx*, *int ny*, *int nz*, *float kappa*, *float epsilonr*)

Constructor of an object to calculate electrostatic forces in MD-SCF framework.

Parameters

- **all_info** (AllInfo) – System information.
- **nx** (*int*) – Number of grid in x direction.
- **ny** (*int*) – Number of grid in y direction.
- **nz** (*int*) – Number of grid in z direction.
- **kappa** (*float*) – kappa = 1/(sqrt(2)sigma) where sigma is the standard deviation of the Gaussian charge distribution.
- **epsilonr** (*float*) – Relative dielectric constant.

setPeriodPFME(*int idl2_step*, *int idl_step*)

sets the periods of computing density field and updating density field.

setNewVersion(*bool switch*)

switches the function of newly developed method of implementation.

Example:

```
pfme = gala.PFMEForce(all_info,    32,   32,   36,  2.45,  epsilonr)#(mx,my,mz,␣
↪kappa, epsilonr)
pfme.setPeriodPFME(1,  100)#(idl2_step, idl_step)
app.add(pfme)
```

17.2 Polymerization

17.2.1 Polymerization model

class Polymerization(*all_info*, *nlist*, *r_cut*, *seed*)

The constructor of an object of polymerization.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.
- **seed** (*int*) – The seed of random number generator.

class Polymerization(*all_info*, *type*, *percent*, *nlist*, *r_cut*, *seed*)

The constructor of an object of polymerization with a percent of initiator.

Parameters

- **all_info** (AllInfo) – The system information.
- **type** (*str*) – The particle type of initiators.
- **percent** (*float*) – The percent of initiators on target particle type.

- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.
- **seed** (*int*) – The seed of random number generator

setPr(*float prob*)

specifies reaction probability.

setPr(*string type1*, *string type2*, *float prob*)

specifies reaction probability between particle type1 and type2.

setPrFactor(*float prob_factor*)

specifies the reaction probability factor of (factor)n where n is reaction times.

setPrFactor(*cstring type1*, *string type2*, *float prob_factor*)

specifies the reaction probability factor between particle type1and type2.

setExchangePr(*string type1*, *string type2*, *string type3*, *float probability*)

specifies the reaction probability of replacing type3 by type1 in the connection to type 2. The init of type2 should be 1. This function will activate ExchMode

setInsertionPr(*string type1*, *string type2*, *string type3*, *float probability*)

specifies the reaction probability of inserting type1 to type2-type3 and forming type2-type1-typ3. The init of type2 should be 1. This function will activate setInsertionMode.

setChangeTypeInReaction(*string name_origin*, *string name_new*)

specifies the type change in reaction in which the particle type will be changed from name_origin to name_new.

setInitInitReaction(*bool reaction*)

allows the reaction between particle i with init=1 and particle j with init=1.

setMaxCris(*string type*, *unsigned int cris_max*)

specifies the upper limit number of bonds generated by reaction. This function will activate SgapMode.

setNewBondType(*string bondtype*)

specifies the type of newly generated bonds by reaction.

setNewAngleType(*string angletype*)

specifies the type of newly generated angles by reaction.

setNewDihedralType(*string dihedraltype*)

specifies the type of newly generated dihedrals by reaction.

setNewBondTypeByPairs()

specifies the type of newly generated bonds by reaction named by the two particle types.

setNewAngleTypeByPairs()

specifies the type of newly generated angles by reaction named by the three particle types.

generateAngle(*bool generate_angle*)

switches on generating angles in reaction.

generateDihedral(*bool generate_dihedral*)

switches on generating dihedrals in reaction.

setFuncReactRule(*bool switch*, *float K*, *float r_0*, *float b_0*, *float epsilon0*, *PolyFunc function*)

switches the rule of the reaction according to energy and specifies the rule with spring constant K, the maximum length for FENE r_0, the equilibrium length of bond b_0, the energy to shift epsilon0, and bond potential type (the candidates are harmonic and FENE). Please refer to "Macromolecules 2016, 49, 75107524".

setEnergyBar(*float ebar*)

specifies the energy bar for reaction instead of Pr.

setMinDisReactRule(*bool switch*)

switches the rule of the reaction only with the nearest particle.

setInitDieProb()

specifies the die probability of initiators.

initExPoint()

switches on initializing reactive point for exchange reaction.

setFrpMode()

specifies the mode of polymerization to be chain growth (such as free redical polymerization) in which the active site continuely moves to the head of chain in the chain growth. This is default mode.

setSgapMode()

specifies the mode of polymerization to be step growth (such as polycondensation and polyaddition). setMaxCris function will activate this mode.

setExchMode()

specifies the mode of polymerization to be chain exchange. setExchangePr function will activate this mode.

setInsertionMode()

specifies the mode of polymerization to be insertion. setInsertionPr function will activate this mode.

Example:

```
reaction = gala.Polymerization(all_info, neighbor_list, 1.12246 ,16361)
reaction.setFuncReactRule(True, 1250.000, 1.0,0.470, 10.0, gala.PolyFunc.harmonic)
reaction.setPr(0.002)
reaction.setMaxCris('B',3)
# sets the connected bond upper limited number.
reaction.setPeriod(50)
app.add(reaction)
```

17.2.2 Depolymerization model

class DePolymerization(*all_info*, *T*, *seed*)

The constructor of an object of depolymerization.

Parameters

- **all_info** (*AllInfo*) – The system information.
- **T** (*float*) – The temperature.
- **seed** (*int*) – The seed of random number generator

setParams(*string type*, *float K*, *float r_0*, *float b_0*, *float epsilon0*, *float Pr*, *DePolyFunc function*)

specifies the depolymerization probability with bond type, spring constant K, the maximum length for FENE r_0, the equilibrium length of bond b_0, the energy to shift epsilon0, and bond potential type (the candidates are harmonic, FENE, and NoFunc. For "NoFunc", only probability works for the judgement of bond rupture).

setT(*float T*)

specifies the temperature with a fixed value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

setCrisQualify()

switches on the checking on the cris value of two connected particles i and j. Only when the cris_i >0 or cris_j > 0, the bond will be judged for breaking.

setChangeTypeInReaction(*string origin_type*, *string new_type*)

The type of particle will be changed after depolymerization from origin_type to new_type.

setDegradeAngle()

swithes on the function of degrading angle.

setDegradeDihedral()

swithes on the function of degrading dihedral.

setCountUnbonds(*int period*)

swithes on the function of counting the number of broken bonds with a period for outputing the number.

Example:

```
reaction = gala.DePolymerization(all_info, 1.0, 16361)
reaction.setParams('sticky', 10.0, 1.5, 0.96, 10.0, 0.2, gala.DePolyFunc.harmonic)
# sets bondname, K, r_0, b_0, epsilon0, Pr, and function.
reaction.setPeriod(1)
# sets how many steps to react.
app.add(reaction)
```

17.3 Anisotropic particle

17.3.1 Gay-Berne model

Uniaxial GB interaction

class GBForce(*all_info*, *nlist*, *r_cut*)

The constructor of a method of Gay-Berne force.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float epsilon0*, *float sigma0*, *float nu*, *float mu*, *float sigma_e*, *float sigma_s*, *float epsilon_e*, *float epsilon_s*, *float Ps*)

specifies the GB force parameters with type1, type2, epsilon0, sigma0, nu, mu, end-to-end length (sigma_e), side-by-side length (sigma_s), end-to-end energy (epsilon_e), side-by-side energy (epsilon_s), Ps.

Example:

```
gb = gala.GBForce(all_info, neighbor_list, 10.0)
gb.setParams('A', 'A' , 1.5, 1.5, 1.0, 2.0,3.0, 1.0, 0.5, 3.0, 1.0)
# sets parameters: type1, type2, epsilon0, sigma0, nu, mu, sigma_e,
# sigma_s, epsilon_e, epsilon_s, Ps.
app.add(gb)
```

Bond force of uniaxial GB particles

class BondForceAni(*all_info*)

The constructor of a method of bond force calculation of anisotropic particles.

Parameters

all_info (AllInfo) – The system information.

setParams(*string bondtype*, *float Kbond*, *float rbond*, *float Kangle*, *float dangle*)

specifies the bond force parameters with bond type, bond spring constant, end-to-end length of GB particle, angle spring constant, equilibrium angle degree.

Example:

```
bondani = gala.BondForceAni(all_info)
bondani.setParams('A-A', 100.0 , 4.498, 30.0, 0.0)
app.add(bondani)
```

Biaxial GB interaction

class PBGBForce(*all_info*, *nlist*)

The constructor of a method of Gay-Berne force.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.

setGUM(*float gamma*, *float nu*, *float mu*)

specifies the GB force parameters with gamma, nu, mu.

setParams(*string type1*, *string type2*, *float epsilon*, *float sigma*, *float r_cut*)

specifies the GB force parameters with type1, type2, epsilon, sigma, cutoff radius.

setAspheres(*string filename*)

specifies the file for shape parameters.

setPatches(*string filename*)

specifies the file for Patch parameters.

Example:

```
pbgb = gala.PBGBForce(all_info, neighbor_list)
pbgb.setGUM(1.0, 3.0, 1.0);#(gamma, niu, miu)
pbgb.setParams('B', 'B' , 1.0, 1.0, 5.0)#(,,epsilon, sigma, rcut)
pbgb.setParams('A', 'A' , 1.0, 1.0, 5.0)#(,,epsilon, sigma, rcut)
pbgb.setParams('A', 'B' , 2.0, 1.0, 5.0)#(,,epsilon, sigma, rcut)
pbgb.setAspheres('patch.log')#(,a,b,c,eia_one,eib_one,eic_one)
pbgb.setPatches('patch.log')
app.add(pbgb)
```

File 'patch.log':

```
<Patches>
B 2                              #particle type, patch number
p1 60  0         0   1           #patch type, beta(degree) which is half of the␣
↪opening angle-
p1 60  0         0   -1          #of the attractive patch, patch position(x, y, z)␣
↪in unit vector
</Patches>
<PatchParams>
p1 p1 88.0 0.5                   #patch type, patch type, alpha_A, and gamma_epsilon
</PatchParams>
<Aspheres>
A 1.0 1.0 1.0 3.0 3.0 3.0        #a,b,c,eia_one,eib_one,eic_one
B 1.0 1.0 3.0 1.0 1.0 0.2        #a,b,c,eia_one,eib_one,eic_one
</Aspheres>
```

17.3.2 Soft anisotropic model

Janus particle model

class LZWForce(*all_info*, *nlist*, *r_cut*)

The constructor of a method of LZW force calculation.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float alphaR*, *float mu*, *float nu*, *float alphaA*, *float beta*)

specifies the LZW force parameters with type1, type2, alphaR, mu, nu, alphaA, and beta.

setMethod(*string method*)

chooses a method of 'Disk', 'Janus', ABAtriJanus', 'BABtriJanus'.

Example:

```
lzw = gala.LZWForce(all_info, neighbor_list, 1.0)
lzw.setParams('A', 'A' , 396.0, 1.0, 0.5, 88.0,60.0/180.0*3.1415926)
lzw.setMethod('ABAtriJanus')
# sets method with the choice of ABAtriJanus.
app.add(lzw)
```

Thermostat for Janus particle model

class BerendsenAniNVT(*AllInfo all_info*, *ParticleSet group*, *ComputeInfo comp_info*, *float T*, *float tauT*, *float tauR*)

The constructor of a Berendsen NVT thermostat for anisotropic particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.
- **comp_info** (ComputeInfo) – The object of calculation of collective information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setTau(*float tauT*, *float tauR*)

specifies the Berendsen NVT thermostat with tauT and tauR.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

Example:

```
bere = gala.BerendsenAniNVT(all_info, group, comp_info, 1.0, 0.3, 0.1)
app.add(bere)
```

Multiple patch particle model

Description:

$$U_{ij} = \begin{cases} \frac{\alpha_{ij}^R d_{ij}}{\mu}\left(1-\frac{r_{ij}}{d_{ij}}\right)^{\mu} - \sum_{\kappa=1}^{M_i}\sum_{\lambda=1}^{M_j} f^{\nu}\left(\mathbf{n}_i^{\kappa}, \mathbf{n}_j^{\lambda}, \mathbf{r}_{ij}\right) \frac{\alpha_{ij}^A d_{ij}}{\mu}\left[\frac{r_{ij}}{d_{ij}} - \left(\frac{r_{ij}}{d_{ij}}\right)^{\mu}\right] & r_{ij} \leq d_{ij} \\ 0 & r_{ij} > d_{ij}, \end{cases}$$

$$f\left(\mathbf{n}_i^{\kappa}, \mathbf{n}_j^{\lambda}, \mathbf{r}_{ij}\right) = \begin{cases} \cos\frac{\pi\theta_i^{\kappa}}{2\theta_m^{\kappa}} \cos\frac{\pi\theta_j^{\lambda}}{2\theta_m^{\lambda}} & \text{if } \cos\theta_i^{\kappa} \geq \cos\theta_m^{\kappa} \text{ and } \cos\theta_j^{\lambda} \geq \cos\theta_m^{\lambda} \\ 0 & \text{otherwise.} \end{cases}$$

The following coefficients must be set per unique pair of particle types:

- α^R - *alphaR*, repulsive strength
- μ - *mu*, the power (unitless)
- α^A - *alphaA*, attractive strength
- d - the diameter defaults to the r_cut (in distance units)
- ν - *nu*, the angular width of attraction (unitless)

class AniForce(*all_info*, *nlist*, *r_cut*)

The constructor of force calculation of multiple patch particle model.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.

setParams(*string type1*, *string type2*, *float alphaR*, *float mu*)

specifies the force parameters with type1, type2, alphaR, mu.

setPatches(*string filename*)

specifies the file for Patch parameters.

Example:

```
ani = gala.AniForce(all_info, neighbor_list, 1.0)
ani.setParams('A', 'A' , 396.0, 2.0)#(,,alpha_R,mu)
ani.setParams('A', 'B' , 396.0, 2.0)#(,,alpha_R,mu)
ani.setParams('B', 'B' , 396.0, 2.0)#(,,alpha_R,mu)
ani.setPatches('patch-3.log')
app.add(ani)
```

File 'patch-3.log':

```
<Patches>
A 0                             #particle type, patch number
B 3                             #particle type, patch number
p1 45  0         0   1          #patch type, beta(degree) which is half of the␣
↪opening angle-
p2 45  0.866025  0   -0.5       #of the attractive patch, patch position(x, y, z)␣
↪in unit vector
p3 45 -0.866025  0   -0.5
</Patches>
<PatchParams>
p1 p1 220.0 0.5                 #patch type, patch type, alpha_A, and nu
p2 p2 220.0 0.5                 #patch type, patch type, alpha_A, and nu
p3 p3 220.0 0.5                 #patch type, patch type, alpha_A, and nu
p1 p2 220.0 0.5                 #patch type, patch type, alpha_A, and nu
p1 p3 220.0 0.5                 #patch type, patch type, alpha_A, and nu
p2 p3 220.0 0.5                 #patch type, patch type, alpha_A, and nu
</PatchParams>
```

Thermostat for multiple patch particle model

The motion of anisotropic particles with multiple patches are integrated by rigid body method with a body index in XML file. The solvent particles with a body index of -1 are integrated by normal methods.

Example:

```
bgroup = gala.ParticleSet(all_info, 'body')
rigidnvt = gala.NVTRigid(all_info, bgroup, 1.0, 0.2)
app.add(rigidnvt)
```

(continues on next page)

(continued from previous page)

```
nbgroup = gala.ParticleSet(all_info,'non_body')
comp_info_nb = gala.ComputeInfo(all_info, nbgroup)
nh = gala.NoseHooverNVT(all_info, nbgroup, comp_info_nb, 1.0, 1.0)#( ,
↪temperature, tau)
app.add(nh)
```

17.4 Dissipative particle dynamics

17.4.1 DPD force

Description:

The DPD force consists of pair-wise conservative, dissipative and random terms.

$$\begin{aligned} \vec{F}_{ij}^{C} &= \alpha \left(1 - \frac{r_{ij}}{r_{cut}}\right) \vec{e}_{ij} \\ \vec{F}_{ij}^{D} &= -\gamma \omega^{D}(r_{ij})(\vec{e}_{ij} \cdot \vec{v}_{ij})\vec{e}_{ij} \\ \vec{F}_{ij}^{R} &= T\sigma \omega^{R}(r_{ij})\xi_{ij}\vec{e}_{ij} \end{aligned}$$

- $\gamma = \sigma^2/2k_BT$
- $\omega^D(r_{ij}) = [\omega^R(r_{ij})]^2 = (1 - r_{ij}/r_{cut})^2$
- ξ_{ij} - a random number with zero mean and unit variance
- T - *temperature* - *optional*: defaults to 1.0
- r_{cut} - *r_cut* (in distance units) - *optional*: defaults to 1.0

The following coefficients must be set per unique pair of particle types:

- α - *alpha* (in energy units)
- σ - *sigma* (unitless)

class DPDForce(*all_info*, *nlist*, *r_cut*, *temperature*, *rand_num*)

The constructor of a DPD interaction object.

Parameters

- **all_info** (AllInfo) – The system information.
- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.
- **temperature** (*float*) – The temperature.
- **rand_num** (*int*) – The seed of random number generator.

class DPDForce(*all_info*, *nlist*, *r_cut*, *rand_num*)

The constructor of a DPD interaction object. The default temperature is 1.0.

Parameters

- **all_info** (AllInfo) – The system information.

- **nlist** (NeighborList) – The neighbor list.
- **r_cut** (*float*) – The cut-off radius.
- **rand_num** (*int*) – The seed of random number generator.

setParams(*string typei*, *string typej*, *float alpha*, *float sigma*)

specifies the DPD interaction parameters per unique pair of particle types.

setT(*float T*)

specifies the temperature with a constant value.

setT(*Variant vT*)

specifies the temperature with a varying value by time step.

setDPDVV()

calls the function to enable DPDVV method (the default is GWVV).

Example:

```
dpd = gala.DPDForce(all_info, neighbor_list, 1.0, 12345)
dpd.setParams('A', 'A', 25.0, 3.0)
app.add(dpd)
```

17.4.2 GWVV integration

Description:

Integration algorithm.

$$
\begin{aligned}
v_i^0 &\leftarrow v_i + \lambda \frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t}) \\
v_i &\leftarrow v_i + \frac{1}{2}\frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t}) \\
r_i &\leftarrow r_i + v_i \Delta t \\
&Calculate \quad F_i^c\{r_j\}, F_i^d\{r_j, v_j^0\}, F_i^r\{r_j\} \\
v_i &\leftarrow v_i + \frac{1}{2}\frac{1}{m}(F_i^c \Delta t + F_i^d \Delta t + F_i^r \sqrt{\Delta t})
\end{aligned}
$$

- λ - *lambda* (unitless) - *optional*: defaults to 0.65

class DPDGWVV(*AllInfo all_info*, *ParticleSet group*)

The constructor of a GWVV NVT thermostat for a group of DPD particles.

Parameters

- **all_info** (AllInfo) – The system information.
- **group** (ParticleSet) – The group of particles.

setLambda(*float lambda*)

specifies lambda.

Example:

```
gwvv = gala.DPDGWVV(all_info, group)
app.add(gwvv)
```

17.4.3 Coulomb interaction in DPD

Description:

In order to remove the divergency at $r = 0$, a Slater-type charge density is used to describe the charged DPD particles. Thereby, *Ewald for DPD (short-range)* (`DPDEwaldForce`) method can be employed to calculated the short-range part of Ewald summation. The long-range part of Ewald summation can be calculated by *PPPM (long-range)* (`PPPMForce`) or *ENUF (long-range)* (`ENUFForce`). And the *ENUF (long-range)* (`ENUFForce`) is suggested.

Example:

```
group_charge = gala.ParticleSet(all_info, "charge")
kappa=0.2

# real space
ewald = gala.DPDEwaldForce(all_info, neighbor_list, group_charge, 3.64)#(,,
↪rcut)
ewald.setParams(kappa)
app.add(ewald)

# reciprocal space
enuf = gala.ENUFForce(all_info, neighbor_list, group_charge)
enuf.setParams(kappa, 2, 2, 20, 20, 20)
app.add(enuf)
```

Reduced charges:

The charges should be converted into the ones in reduced units according to *Charge units*. Typically, the fundamental length and energy are $\sigma = 0.646\,nm$ and $\epsilon = k_BT$ with $T = 300\,K$, respectively, in DPD. The reduced charges are $q^* = z\sqrt{f/(\sigma k_B T \epsilon_r)}$. The z is the valence of ion.

Here is a *molgen* script for polyelectrolyte.

Example:

```
#!/usr/bin/python
import sys

import molgen
import math

er=78.0
kBT=300.0*8.314/1000.0
r=0.646
gama=138.935
dpdcharge=math.sqrt(gama/(er*kBT*r))

mol1=molgen.Molecule(50)
mol1.setParticleTypes("P*50")
topo="0-1"
for i in range(1,50-1):
  c=","+str(i)+"-"+str(i+1)
  topo+=c
mol1.setTopology(topo)
```

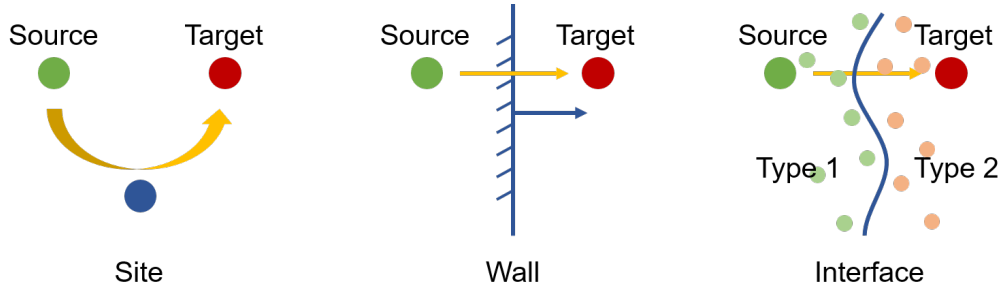
(continues on next page)

(continued from previous page)

```
mol1.setBondLength(0.7)
mol1.setMass(1.0)

mol2=molgen.Molecule(1)
mol2.setParticleTypes("C")
mol2.setMass(1.0)
mol2.setCharge(dpdcharge)

mol3=molgen.Molecule(1)
mol3.setParticleTypes("A")
mol3.setMass(1.0)
mol3.setCharge(-dpdcharge)

mol4=molgen.Molecule(1)
mol4.setParticleTypes("W")
mol4.setMass(1.0)

gen=molgen.Generators(15,15,15)
gen.addMolecule(mol1,1)
gen.addMolecule(mol2,75)
gen.addMolecule(mol3,75)
gen.addMolecule(mol4,9925)

gen.outPutXml("ps0")
```

17.5 Particle type change

A particle type change method can be used for the coarse-grained simulation of catalytic reaction of small molecules.



17.5.1 ChangeType

class ChangeType(*all_info*, *source_type*, *target_type*)

The constructor of an object of ChangeType.

Parameters

- **all_info** (AllInfo) – The system information.
- **source_type** (*string*) – The type of source particles.
- **target_type** (*string*) – The type of target particles.

setPr(*float prob*)

specifies reaction probability.

setSite(*NeighborList nlist*, *string site_type*, *float rcut*)

changing source particles to target particles, triggered by the sites (with site_type) in the neighbor list (nlist) of source particles within a cutoff of radius (rcut).

setWall(*float o_x*, *float o_y*, *float o_z*, *float d_x*, *float d_y*, *float d_z*)

specifies the wall with original point(o_x, o_y, o_z) and normal direction(d_x, d_y, d_z), going through which the source particles will be changed to target particles.

setInterface(*NeighborList nlist*, *string type1*, *string type2*, *float rcut*)

specifies the interface between type1 particles and type2 particles, going through which the source particles will be changed to target particles. The interface is calculated from the neighbor list of sorce particles with a cutoff radius(rcut).

setNPTargetType(*int np*)

specifies the number of target particles.

Example:

```
ct = gala.ChangeType(all_info, "A", "B")
ct.setSite(nlist, "C", 1.0)
ct.setPr(0.002)
ct.setPeriod(50)
app.add(ct)
```

CHAPTER

EIGHTEEN

MOLGEN

18.1 Description of molgen

The initial configuration of simulation systems could be generated by employing a plugin of PYGAMD (named **molgen**, the abbreviation of "molecule generator"). The format of output files could be MST, Mol2 or XML.

The **molgen** is composed of two parts:

1. Defination of molecules and objects where the molecules describe flexible chains and the objects describe rigid objects with fixed shapes.
2. Generator of system with box size and the number of defined molecules and objects.

A molecule is defined by "Molecule" class with the number of particles, particle types, topology, and so on. An object is defined by "Object" class with the number of particles and shape. One or multiple molecules or objects can be defined in this way.

Head of script:

```
from poetry import molgen
# imports an extended Python module of molgen.
```

Example for molecule:

```
mol1=molgen.Molecule(4)
mol1.setParticleTypes("A,A,A,A")
mol1.setTopology("0-1,1-2,2-3")
mol1.setBondLength(0.75)
mol1.setMass(1.0)
mol1.setAngleDegree("A","A","A",90.0)
```

"Generators" class could specify box size and the number of defined molecule.

Example for generator:

```
gen=molgen.Generators(10,10,10)
gen.addMolecule(mol1,10)
gen.setMinimumDistance(0.7)
gen.outPutMST("test")
```

The configuration of a molecule can be partially read from a MST file and partially defined in the script.

Example:

```
mol0=molgen.Molecule("sphere.mst",65)
# total 65 particles with 60 read from "sphere.mst".
mol0.setParticleTypes("B*5")
# define the types of the particles which will be generated.
mol0.setTopology("59-60,60-61,61-62,62-63,63-64")
# add the bonds.
```

If the configuration of a molecule (including particle positions, types, topology and so on) has been completely given in a MST or XML file, we could generate the system by employing "object" to randomaly place and rotate the molecule.

Example:

```
mol1 = molgen.Object("mol1.mst", 65, molgen.Shape.none)
gen=molgen.Generators(30,30,30)
gen.addMolecule(mol1,20)
gen.outPutMST("test")
or
mol1 = molgen.Object("mol1.xml", 65, molgen.Shape.none)
gen=molgen.Generators(30,30,30)
gen.addMolecule(mol1,20)
gen.outPutXML("test")
```

18.2 Molecule definition

class Molecule(*np*)

The constructor of a molecule with the number of particles.

Parameters

np (*int*) – The number of particles.

class Molecule(*filename*, *np*)

The constructor of a molecule and reads particles data from the MST file with file name and the number of particles.

Parameters

- **filename** (*str*) – The name of inputting file.
- **np** (*int*) – The number of particles.

setParticleTypes(*string type*)

specifies the particle types separated by comma according to particle index form 0 to N-1 in sequence.

setTopology(*string topo*)

specifies the bonds separated by comma which connect two particles separated by crossband.

setIsotactic(*bool switch*)

switches the isotactic configuration of molecule.

setBondLength(*double bl*)

specifies the bond length of all bonds.

setBondLength(*string type1*, *string type2*, *double bl*)

specifies the bond length of the bond which connect two kind particles with particle type1, type2, and bond length.

setAngleDegree(*string type1*, *sstring type2*, *string type2*, *double degree*)

specifies the angle with particle type 1, type2, type3, and degree. When angle degree is set as zero, the angles will not be fixed in configuration, but the angle information will still be generated.

setAngleDegree(*unsigned int idx1*, *unsigned int idx2*, *unsigned int idx3*, *double degree*)

specifies the angle with particle idx1, idx2, idx3, and degree.

setDihedralDegree(*string type1*, *string type2*, *string type3*, *string type4*, *double degree*)

specifies the dihedral with particle type1, type2, type3, type4, and degree.

setDihedralDegree(*unsigned int idx1*, *unsigned int idx2*, *unsigned int idx3*, *unsigned int idx4*, *double degree*)

specifies the dihedral with particle idx1, idx2, idx3, idx4, and degree.

setMass(*double mass*)

specifies the mass of all particles.

setMass(*string type*, *double mass*)

specifies the mass of a kind of particles.

setMass(*unsigned int particle_index*, *double mass*)

specifies the mass of a particle.

setCharge(*double charge*)

specifies the charge of all particles.

setCharge(*string type*, *double charge*)

specifies the charge of a kind of particles.

setCharge(*unsigned int particle_index*, *double charge*)

specifies the charge of a particle.

setOrientation()

specifies all particles having orientation.

setOrientation(*string type*)

specifies a kind of particles having orientation.

setOrientation(*unsigned int particle_index*)

specifies a particle having orientation.

setInert(*double inertx*, *double inerty*, *double inertz*)

specifies the inert in x, y, z directions of all particles.

setInert(*string type*, *double inertx*, *double inerty*, *double inertz*)

specifies the inert in x, y, z directions of a kind of particles.

setInert(*unsigned int particle_index*, *double inertx*, *double inerty*, *double inertz*)

specifies the inert in x, y, z directions of a particle.

setQuaternion()

specifies all particles having quaternion.

setQuaternion(*string type*)

specifies a kind of particles having quaternion.

setQuaternion(*unsigned int particle_index*)

specifies a particle having quaternion.

setDiameter(*double di*)

specifies the diameter of all particles.

setDiameter(*string type*, *double di*)

specifies the diameter of a kind of particles.

setDiameter(*unsigned int particle_index*, *double di*)

specifies the diameter of a particle.

setCris(*unsigned int cris*)

specifies the cris of all particles.

setCris(*string type*, *unsigned int cris*)

specifies the cris of a kind of particles.

setCris(*unsigned int particle_index*, *unsigned int cris*)

specifies the cris of a particle.

setInit(*unsigned int init*)

specifies the init of all particles.

setInit(*string type*, *unsigned int init*)

specifies the init of a kind of particles.

setInit(*unsigned int particle_index*, *unsigned int init*)

specifies the init of a particle.

setBody(*unsigned int body_id*)

specifies the body id of all particles (start form 0).

setBody(*string type*, *unsigned int body_id*)

specifies the body id of a kind of particles (start form 0).

setBody(*unsigned int particle_index*, *unsigned int body_id*)

specifies the body id of a particle (start form 0).

setMolecule(*unsigned int mol_id*)

specifies the molecule id of all particles (start form 0).

setMolecule(*string type*, *unsigned int mol_id*)

specifies the mlecule id of a kind of particles (start form 0).

setMolecule(*unsigned int particle_index*, *unsigned int mol_id*)

specifies the molecule id of a particle (start form 0).

setBox(*double lx*, *double ly*, *double lz*)

specifies the size of box where the molecules are generated.

setBox(*double lx_min*, *double lx_max*, *double ly_min*, *double ly_max*, *double lz_min*, *double lz_max*)

specifies the box where the molecules are generated with box boundaries: lx_min, lx_max, ly_min, ly_max, lz_min, lz_max.

setSphere(*double sx*, *double sy*, *double sz*, *double r_min*, *double r_max*)

specifies the sphere where the molecules are generated with sphere center position(sx, sy, sz), spherical shell radius r_min, and r_max. The molecules are generated in the range r_min< r < r_max.

setCylinder(*double px*, *double py*, *double pz*, *double ax*, *double ay*, *double az*, *double r_min*, *double r_max*)

specifies the cylinder where the molecules are generated with cylinder center position(px, py, pz), cylinder axe vecter(ax, ay, ax), cyliner radius r_min, and r_max. The molecules are generated in the range r_min< r<r_max.

setBodyEvacuation()

specifies the generation of molecules outside bodies.

Example:

```
mol0=molgen.Molecule(8)
# initializes a molecule object with the number of particles.
mol0.setParticleTypes("A,A,A,A,A,A,A,A")
# sets particle types.
mol0.setTopology("0-1,0-3,0-4,2-3,1-2,1-5,2-6,3-7,4-5,4-7,5-6,6-7")
# sets topology.
mol0.setBondLength(0.75)
# sets bond length for all bonds.
mol0.setMass(1.0)
# sets mass for all particle.
mol0.setAngleDegree("A","A","A",90.0)
# sets the degree of the angle of particles with the type 1, 2 and 3.
```

18.3 Objects definition

class Object(*np*, *shape*)

The constructor of an object with the number of particles and shape.

Parameters

- **np** (*int*) – The number of particles.
- **shape** (*Shape*) – The shape of object.

class Object(*string filename*, *unsigned int*, *Shape*)

The constructor of an object by reading partial data from a file with file name, the number of particles, and shape (the candidates are "none" and "sphere").

Parameters

- **filename** (*str*) – The name of inputting file.
- **np** (*int*) – The number of particles.
- **shape** (*Shape*) – The shape of object.

setRadius(*double radius*)

specifies the radius of the sphere which will be generated(only works for "sphere" shape) with radius.

Example:

```
mol0 = molgen.Object("sphere.MST", 65, molgen.Shape.none)
# initializes an object by the reading file (containing 60 particles),
# the number of particles, and object shape.
mol0.setParticleTypes("A*5")
# sets particle types (the former 60 types can be read form the file).
```

(continues on next page)

(continued from previous page)

```
mol0.setTopology("59-60,60-61,61-62,62-63,63-64")
# sets topology.
mol0.setBody("C", 0)
# sets body index (the type "C" particles are thereby rigid body particles).
```

18.4 Generator definition

class Generators(*double lx*, *double ly*, *double lz*)

The constructor of a molecule generator with box length in x y z directions.

Parameters

- **lx** (*float*) – The box length in x direction.
- **ly** (*float*) – The box length in y direction.
- **lz** (*float*) – The box length in z direction.

addMolecule(*Molecule mol*, *unsigned int num*)

adds a molecule into generator with molecule object and number.

setMinimumDistance(*double min_dis*)

sets the minimum separated distance of all particles.

setMinimumDistance(*string type1*, *string type2*, *double min_dis*)

sets the minimum separated distance between two particle types with particle type 1, particle type 2 and minimum distance.

setParam(*string type1*, *string type2*, *double epsilon*, *double sigma*, *double r_cut*)

sets the LJ potential parameters between two particle types for Rosenblueth method with particle type1, particle type2, epsilon, sigma, and cut-off radius.

setDimension(*unsigned int dimension*)

specifies system dimension, the default value is 3.

outPutMST(*string filename*)

switch the function of outputting MST filename.

outPutMol2(*string filename*)

switch the function of outputting Mol2 files.

outPutXML(*string filename*)

switch the function of outputting XML filename.

Example:

```
gen=molgen.Generators(10, 10, 10)
# initializes a generator object by box length in x, y, and z direction.
gen.addMolecule(mol0, 10)
# adds a molecule by molecule name and the number of molecules.
gen.setParam("A","A", 1.0, 0.7, 1.0)
# sets the parameters of LJ potential which is used for Rosenblueth method.
gen.setMinimumDistance(0.7)
# sets the minimum separated distance of all particles.
```

(continues on next page)

(continued from previous page)

```
gen.setMinimumDistance("A","A", 0.7)
# sets the minimum separated distance between the particle types.
gen.outPutMST("test")
# sets the name of output MST file.
```

CHAPTER

NINETEEN

DATATACKLE

19.1 Load dataTackle

The **dataTackle** can be loaded by the command "from poetry import dataTackle" in Python3. Then you can use **dataTackle** locally by the command "./dataTackle".

19.2 Usage

The **dataTackle** is a plugin of PYGAMD to analyze some important properties by reading the generated configuration files. The **dataTackle** can be compiled and installed by "sh compile.sh". You can use this tool to analyze one or more files at a time.

Examples for running:

```
./dataTackle particle.mst
./dataTackle particle0.mst particle1.mst
./dataTackle *.mst
```

After pressing enter, a menu of function options will be listed. You can choose one or more functions by the number indexes separated by blank. The parameters for a function can be input after ":" and seperated by "|".

Such as:

```
4:npot=1000 3:gpu=0|rmax=2.0
```

The **dataTackle** also can be called and passed parameters in a Shell script.

Such as:

```
echo -e "18:gpu=1|rmax=3.0" | dataTackle particles.*.mst
```

The **dataTackle** plugin supports the configuration files with MST and DCD formats. The MST files can be tackled independently. However, a DCD trajectory file has to be tackled along with a MST file for particles attributes and topological information.

Examples:

```
dataTackle particle.mst trajectory.dcd
```

For the help about the parameters, you could input "function number:h" after the function list shown.

Examples:

```
14:h
```

19.3 Functions

Function list:

```
-------------------------------------------------------------
1  Rg^2              2  Ed^2               3  RDF
4  bond_distri       5  angle_distri       6  dihedral_distri
7  stress tensor     8  density            9  unwrapping
10 MSD               11 RDF-CM             12 MSD-CM
13 ents              14 strfac             15 domain size
16 dynamic strfac    17 config check       18 RDF between types
19 MST conversion    20 patch/spot display 21 SSF
22 ADF               23 CND                24 MSAD
25 RMSAD             26 ISF                27 OACF
28 Q4Q6              29 VORONOI            30 NGP
31 RNGP              32 VHF                33 RVHF
34 fpSus             35 RfpSus             36 OvlaF
37 CISF              38 CAGEISF            39 CAGEMSD
40 RMSD              41 P2P4               42 CRYSTALLINITY
43 G6_3D             44 W4W6
-------------------------------------------------------------
```

19.3.1 1 Rg^2:

Description:

The mean square of gyration radius is calculated and output to `rg2.log`.

$$R_g^2 = \frac{1}{N}\sum_{i=1}^{N}(\vec{R}_i - \vec{R}_{cm})^2$$
$$\vec{R}_{cm} = \frac{1}{N}\sum_{i=1}^{N}\vec{R}_i$$

Coefficients:

- $\vec{R}_i$ - monomer position vector

19.3.2 2 Ed^2:

Description:

The mean square of end-to-end distance is calculated and output to `ed2.log`.

$$E_d^2 = (\vec{R}_0 - \vec{R}_{N-1})^2$$

Coefficients:

- $\vec{R}_i$ - monomer position vector

19.3.3 3 RDF:

Description:
The radial distribution function of all particles is calculated and output to `filename.rdf`. Averaged value among files will be output to `rdf.log`.

Parameters:
:maxbin=100|gpu=0|rmax=Lx/2|bondex=false|angleex=false|molex=false

19.3.4 4 bond_distri:

Description:

The distribution of bond lengths is calculated and output to `bond_distr.log`.

$$bond\_distri(i \cdot dr) = N(i)/(N \cdot dr)$$

Coefficients:

- dr - the space of bond length *L/(2npot)*, where *L* is the box size
- $N(i)$ - the number of bonds in the range of *idr < r < (i+1)dr*, where *i* is an integer
- N - the total number of bonds

Parameters:
:npot=2001

19.3.5 5 angle_distri:

Description:

The distribution of angle degrees is calculated and output to `angle_distr.log`.

$$angle\_distri(i \cdot da) = N(i)/(N \cdot da)$$

Coefficients:

- da - the space of angle radian *pi/npot*
- $N(i)$ - the number of angles in the range of *ida < angle < (i+1)da*, where *i* is an integer
- N - the total number of angles

Parameters:
:npot=2001

19.3.6 6 dihedral_distri:

Description:

The distribution of dihedral degrees is calculated and output to `dihedral_distr.log`.

$$dihedral\_distri(i \cdot da) = N(i)/(N \cdot da)$$

Coefficients:

- da - the space of dihedral angle radian *2pi/npot*
- $N(i)$ - the number of dihedrals in the range of *ida < dihedral angle < (i+1)da*, where *i* is an integer
- N - the total number of dihedrals

Parameters:
:npot=2001

19.3.7 7 stress tensor:

Description:
Stress tensor is calculated by inputing the parameters for force calculation. Result will be output to `stress_tensor.log`.

Parameters:
:bondex=true|bodyex=true|diameter=true

19.3.8 8 density:

Description:
Real density (g/cm^3) with basic units [amu] and [nm] is calculated and output to `density.log`.

19.3.9 9 unwrapping:

Description:
This function would unwrap or shift molecules by changing image or position of particles. New configuration will be output to `filename.reimage.mst`

Parameters:
:unwrap_molecule=true|label_free_particle=particle type|molecule_center_in_box=false|shiftx=0.0|shifty=0.0|shiftz=0.0|remove_image=false|add_image_to_pos=true|remove_bond_cross_box=false|body_keep=false

19.3.10 10 MSD:

Description:
The mean square displacement of all particles is calculated and output to `msd.log`.

Parameters:
:direction=XYZ (candidates are X,Y,Z,XY,YZ,XZ,XYZ)

19.3.11 11 RDF-CM:

Description:
The radial distribution function of the mass center of molecules is calculated and output to `rdf_cm.log`.

Parameters:
:maxbin=100|gpu=0|rmax=Lx/2

19.3.12 12 MSD-CM:

Description:
The mean square displacement of the mass center of molecules is calculated and output to `msd_cm.log`.

Parameters:
:direction=XYZ (candidates are X,Y,Z,XY,YZ,XZ,XYZ)

19.3.13 13 ents:

Description:
This function would analyze the entanglements of polymers. Result will be output to `ents.log`.

19.3.14 14 strfac:

Description:
The structure factor of particles is calculated and output to `filename.strf`. The averaged value among files is output to `strf.log`.

Parameters:
:qmax=160pi/Lmin|gpu=0|deltaq=2pi/Lmin|direction=XYZ|2D=false

19.3.15 15 domain size:

Description:
The domain size of components in mixtures is calculated and output to `domsize.log`.

Parameters:
:kmax=20|qc=0.4690|gpu=0

19.3.16 16 dynamic strfac:

Description:
Dynamic structure factor (incoherent intermediate) measures the decorrelation of the positions of individual monomers with the time on length scale $1/q$, where $q = 2\pi\sqrt{x^2 + y^2 + z^2}/L$, and L is cubic box length. kmax limits the space in which the q with possible combinations of x, y, z will be generated.

Results will be output to `dstrf.log`.

Parameters:
:kmax=int(L)|q=7.0

Maintainer: Shu-Jia Li

19.3.17 17 config check:

Description:
This function would check the configuration including the minimum distance of particles, and the maximum and minimum length of bonds, etc. Result will be output to `config_check.log`.

Parameters:
:bondex=true|angleex=true|dihedralex=true|bodyex=true|rcut=2.0

19.3.18 18 RDF between types:

Description:
The radial distribution function between types is calculated and output to `filename.type.rdf`. The averaged value among files will be output to `rdf_by_type.log`.

Parameters:
:maxbin=100|gpu=0|rmax=Lx/2|bondex=false|angleex=false|molex=false

19.3.19 19 MST conversion:

Description:
This function would convert MST files into new files with another format.

Parameters:
:lammps=false|gromacs=false|xml=false

CHAPTER

TWENTY

LICENSE

PYGAMD - Python GPU-Accelerated Molecular Dynamics Software

Version 1

COPYRIGHT

PYGAMD Copyright (c) (2021) You-Liang Zhu and Zhong-Yuan Lu

LICENSE

This program is a free software: you can redistribute it and/or modify it under the terms of the GNU General Public License. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANT ABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the General Public License v3 for more details. You should have received a copy of the GNU General Public License along with this program. If not, see GNU.

DISCLAIMER

The authors of PYGAMD do not guarantee that this program and its derivatives are free from error. In no event shall the copyright holder or contributors be liable for any indirect, incidental, special, exemplary, or consequential loss or damage that results from its use. We also have no responsibility for providing the service of functional extension of this program to general users. USER OBLIGATION

If any results obtained with PYGAMD are published in the scientific literature, the users have an obligation to distribute this program and acknowledge our efforts by citing the paper "Y.-L. Zhu et al., J. Comput. Chem. 2013, 34, 2197-2211" in their article.

CORRESPONDENCE

Dr. You-Liang Zhu; Email: ylzhu@pygamd.com

CHAPTER

TWENTYONE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

A

AllInfo (*built-in class*), 50
AllInfo.addAngleType()
    built-in function, 50
AllInfo.addAngleTypeByPairs()
    built-in function, 50
AllInfo.addBondType()
    built-in function, 50
AllInfo.addBondTypeByPairs()
    built-in function, 50
AllInfo.addParticleType()
    built-in function, 50
AllInfo.setNDimensions()
    built-in function, 50
AndersenNVT (*built-in class*), 93
AndersenNVT.setT()
    built-in function, 93
AngleForceCos (*built-in class*), 72
AngleForceCos.setParams()
    built-in function, 72
AngleForceHarmonic (*built-in class*), 71, 89
AngleForceHarmonic.setParams()
    built-in function, 71
AngleForceHarmonicCos (*built-in class*), 71, 88
AngleForceHarmonicCos.setParams()
    built-in function, 71
AngleForceLnExp (*built-in class*), 72
AngleForceLnExp.setParams()
    built-in function, 73
AngleForceTable (*built-in class*), 79
AngleForceTable.setParams()
    built-in function, 79
AngleForceTable.setPotential()
    built-in function, 79
AniForce (*built-in class*), 118
AniForce.setParams()
    built-in function, 119
AniForce.setPatches()
    built-in function, 119
Application (*built-in class*), 55
Application.add()
    built-in function, 55
Application.clear()
    built-in function, 55
application.dynamics (*built-in class*), 17
application.dynamics.add()
    built-in function, 17
application.dynamics.remove()
    built-in function, 17
application.dynamics.run()
    built-in function, 17
Application.remove()
    built-in function, 55
Application.run()
    built-in function, 55
Application.setDt()
    built-in function, 55
AxialStretching (*built-in class*), 108
AxialStretching.setBoxLength()
    built-in function, 108

B

BerendsenAniNVT (*built-in class*), 118
BerendsenAniNVT.setT()
    built-in function, 118
BerendsenAniNVT.setTau()
    built-in function, 118
BerendsenNVT (*built-in class*), 93
BerendsenNVT.setT()
    built-in function, 93
BinaryDump (*built-in class*), 46
BinaryDump.enableCompression()
    built-in function, 47
BinaryDump.setOutput()
    built-in function, 46
BinaryDump.setOutputAll()
    built-in function, 46
BinaryDump.setOutputForRestart()
    built-in function, 47
BinaryReader (*built-in class*), 41
BondConstraint (*built-in class*), 89, 105
BondConstraint.setExpansionOrder()
    built-in function, 89, 105
BondConstraint.setNumIters()

built-in function, 89, 105
BondConstraint.setParams()
built-in function, 105
BondForceAni (*built-in class*), 116
BondForceAni.setParams()
built-in function, 116
BondForceFENE (*built-in class*), 68
BondForceFENE.setConsiderDiameter()
built-in function, 68
BondForceFENE.setParams()
built-in function, 68
BondForceHarmonic (*built-in class*), 67, 88
BondForceHarmonic.setParams()
built-in function, 67
BondForceMorse (*built-in class*), 69
BondForceMorse.setParams()
built-in function, 69
BondForcePolynomial (*built-in class*), 69
BondForcePolynomial.setParams()
built-in function, 69
BondForceTable (*built-in class*), 78
BondForceTable.setParams()
built-in function, 78
BondForceTable.setPotential()
built-in function, 78, 79
BounceBackConstrain (*built-in class*), 102
BounceBackConstrain.addCylinder()
built-in function, 102
BounceBackConstrain.addSphere()
built-in function, 102
BounceBackConstrain.addWall()
built-in function, 102
BounceBackConstrain.clcearWall()
built-in function, 103
BounceBackConstrain.clearCylinder()
built-in function, 103
BounceBackConstrain.clearSphere()
built-in function, 103
built-in function
AllInfo.addAngleType(), 50
AllInfo.addAngleTypeByPairs(), 50
AllInfo.addBondType(), 50
AllInfo.addBondTypeByPairs(), 50
AllInfo.addParticleType(), 50
AllInfo.setNDimensions(), 50
AndersenNVT.setT(), 93
AngleForceCos.setParams(), 72
AngleForceHarmonic.setParams(), 71
AngleForceHarmonicCos.setParams(), 71
AngleForceLnExp.setParams(), 73
AngleForceTable.setParams(), 79
AngleForceTable.setPotential(), 79
AniForce.setParams(), 119
AniForce.setPatches(), 119
Application.add(), 55
Application.clear(), 55
application.dynamics.add(), 17
application.dynamics.remove(), 17
application.dynamics.run(), 17
Application.remove(), 55
Application.run(), 55
Application.setDt(), 55
AxialStretching.setBoxLength(), 108
BerendsenAniNVT.setT(), 118
BerendsenAniNVT.setTau(), 118
BerendsenNVT.setT(), 93
BinaryDump.enableCompression(), 47
BinaryDump.setOutput(), 46
BinaryDump.setOutputAll(), 46
BinaryDump.setOutputForRestart(), 47
BondConstraint.setExpansionOrder(), 89, 105
BondConstraint.setNumIters(), 89, 105
BondConstraint.setParams(), 105
BondForceAni.setParams(), 116
BondForceFENE.setConsiderDiameter(), 68
BondForceFENE.setParams(), 68
BondForceHarmonic.setParams(), 67
BondForceMorse.setParams(), 69
BondForcePolynomial.setParams(), 69
BondForceTable.setParams(), 78
BondForceTable.setPotential(), 78, 79
BounceBackConstrain.addCylinder(), 102
BounceBackConstrain.addSphere(), 102
BounceBackConstrain.addWall(), 102
BounceBackConstrain.clcearWall(), 103
BounceBackConstrain.clearCylinder(), 103
BounceBackConstrain.clearSphere(), 103
CCPMD.setParams(), 105
CCPMD.setWall(), 105
CellList.setDataReproducibility(), 51
CellList.setNominalDim(), 51
CellList.setNominalWidth(), 51
CenterForce.setPreNextShift(), 62
ChangeType.setInterface(), 124
ChangeType.setNPTargetType(), 124
ChangeType.setPr(), 123
ChangeType.setSite(), 124
ChangeType.setWall(), 124
ComputeInfo.setNdof(), 54
DCDDump.unpbc(), 46
DCDDump.unwrap(), 46
DePolymerization.setChangeTypeInReaction(), 115
DePolymerization.setCountUnbonds(), 115
DePolymerization.setCrisQualify(), 115
DePolymerization.setDegradeAngle(), 115

DePolymerization.setDegradeDihedral(), 115
DePolymerization.setParams(), 114
DePolymerization.setT(), 115
DihedralForceAmberCosine.setParams(), 76
DihedralForceHarmonic.setCosFactor(), 74
DihedralForceHarmonic.setParams(), 74, 75
DihedralForceOPLSCosine.setParams(), 75
DihedralForceRyckaertBellemans.setParams(), 76
DihedralForceTable.setParams(), 79
DihedralForceTable.setPotential(), 79
DPDEwaldForce.setBeta(), 83
DPDEwaldForce.setParams(), 83
DPDForce.setDPDVV(), 121
DPDForce.setParams(), 121
DPDForce.setT(), 121
DPDGWVV.setLambda(), 121
DumpInfo.dumpAnisotropy(), 42
DumpInfo.dumpBoxSize(), 42
DumpInfo.dumpParticleForce(), 42
DumpInfo.dumpParticlePosition(), 42
DumpInfo.dumpPotential(), 42
DumpInfo.dumpPressTensor(), 42
DumpInfo.dumpTypeTemp(), 42
DumpInfo.dumpVirial(), 42
DumpInfo.dumpVirialMatrix(), 42
DumpInfo.setPeriod(), 42
ENUFForce.setParams(), 85
EwaldForce.setParams(), 82
ExternalForce.setForce(), 107
ExternalForce.setParams(), 107
force.angle.setParams(), 29
force.bond.setParams(), 26
force.dihedral.setCosFactor(), 31
force.dihedral.setParams(), 31
force.dpd.setParams(), 35
force.nonbonded.setParams(), 22
force.nonbonded_c.setParams(), 24
GBForce.setParams(), 115
GEMForce.setParams(), 62
Generators.addMolecule(), 130
Generators.outPutMol2(), 130
Generators.outPutMST(), 130
Generators.outPutXML(), 130
Generators.setDimension(), 130
Generators.setMinimumDistance(), 130
Generators.setParam(), 130
integration.bd.setParams(), 34
integration.nvt.setT(), 33
LangevinNVT.setGamma(), 94
LangevinNVT.setT(), 94
LangevinNVTRigid.setGamma(), 95
LangevinNVTRigid.setT(), 95
LJConstrainForce.addCylinder(), 103
LJConstrainForce.addSphere(), 103
LJConstrainForce.addWall(), 103
LJConstrainForce.clcearWall(), 103
LJConstrainForce.clearCylinder(), 103
LJConstrainForce.clearSphere(), 103
LJConstrainForce.setParams(), 103
LJEwaldForce.setDispVirialCorr(), 63, 88
LJEwaldForce.setEnergy_shift(), 63, 88
LJEwaldForce.setParams(), 63
LJForce.setDispVirialCorr(), 60
LJForce.setEnergy_shift(), 60
LJForce.setParams(), 60
LZWForce.setMethod(), 117
LZWForce.setParams(), 117
MDSCFForce.setNewVersion(), 111
MDSCFForce.setParams(), 111
MDSCFForce.setPeriodScf(), 111
MOL2Dump.deleteBoundaryBond(), 43
MOL2Dump.setChangeFreeType(), 43
Molecule.setAngleDegree(), 126, 127
Molecule.setBody(), 128
Molecule.setBodyEvacuation(), 129
Molecule.setBondLength(), 126
Molecule.setBox(), 128
Molecule.setCharge(), 127
Molecule.setCris(), 128
Molecule.setCylinder(), 128
Molecule.setDiameter(), 127, 128
Molecule.setDihedralDegree(), 127
Molecule.setInert(), 127
Molecule.setInit(), 128
Molecule.setIsotactic(), 126
Molecule.setMass(), 127
Molecule.setMolecule(), 128
Molecule.setOrientation(), 127
Molecule.setParticleTypes(), 126
Molecule.setQuaternion(), 127
Molecule.setSphere(), 128
Molecule.setTopology(), 126
NeighborList.addExclusionsFromAngles(), 51
NeighborList.addExclusionsFromBodies(), 51
NeighborList.addExclusionsFromBonds(), 51
NeighborList.addExclusionsFromDihedrals(), 51
NeighborList.setDataReproducibility(), 51
NeighborList.setFilterDiameters(), 51
NeighborList.setNsq(), 51
NeighborList.setRCut(), 51
NeighborList.setRCutPair(), 51
NoseHooverNVT.setT(), 93
NPT.setP(), 96

NPT.setT(), 96
NPTMTK.setAnisotropic(), 98
NPTMTK.setSemiisotropic(), 98
NPTMTK.setT(), 98
NPTMTKRigid.setAnisotropic(), 99
NPTMTKRigid.setSemiisotropic(), 99
NPTMTKRigid.setT(), 98, 99
NPTRigid.setP(), 97
NPTRigid.setT(), 97
NVE.setZeroForce(), 91
NVTRigid.setT(), 95
Object.setRadius(), 129
PairForce.setParams(), 66
PairForce.setShiftParams(), 66
PairForceTable.setParams(), 78
PairForceTable.setPotential(), 78
ParticleSet.getNumMembers(), 52
PBGBForce.setAspheres(), 116
PBGBForce.setGUM(), 116
PBGBForce.setParams(), 116
PBGBForce.setPatches(), 116
PFMEForce.setNewVersion(), 112
PFMEForce.setPeriodPFME(), 112
Polymerization.generateAngle(), 113
Polymerization.generateDihedral(), 113
Polymerization.initExPoint(), 114
Polymerization.setChangeTypeInReaction(), 113
Polymerization.setEnergyBar(), 114
Polymerization.setExchangePr(), 113
Polymerization.setExchMode(), 114
Polymerization.setFrpMode(), 114
Polymerization.setFuncReactRule(), 113
Polymerization.setInitDieProb(), 114
Polymerization.setInitInitReaction(), 113
Polymerization.setInsertionMode(), 114
Polymerization.setInsertionPr(), 113
Polymerization.setMaxCris(), 113
Polymerization.setMinDisReactRule(), 114
Polymerization.setNewAngleType(), 113
Polymerization.setNewAngleTypeByPairs(), 113
Polymerization.setNewBondType(), 113
Polymerization.setNewBondTypeByPairs(), 113
Polymerization.setNewDihedralType(), 113
Polymerization.setPr(), 113
Polymerization.setPrFactor(), 113
Polymerization.setSgapMode(), 114
PPPMForce.setParams(), 84
RNEMD.setPeriod(), 109
RNEMD.setProfVelPeriod(), 109
RNEMD.setSwapGroup(), 109
RNEMD.setSwapPeriod(), 109
RNEMD.setVelProfile(), 109
setHead(), 42
setPrecision(), 42
SLJForce.setEnergy_shift(), 61
SLJForce.setParams(), 61
TranRigid.setRotDimension(), 92
TranRigid.setTraDimension(), 92
VariantLinear.setPoint(), 101
VariantSin.setPoint(), 102
VariantWell.setPoint(), 102
Vsite.setParams(), 106
XMLDump.setOutput(), 43
XMLDump.setOutputAngle(), 45
XMLDump.setOutputBody(), 44
XMLDump.setOutputBond(), 45
XMLDump.setOutputCharge(), 44
XMLDump.setOutputConstraint(), 45
XMLDump.setOutputCris(), 45
XMLDump.setOutputDiameter(), 44
XMLDump.setOutputDihedral(), 45
XMLDump.setOutputEllipsoid(), 45
XMLDump.setOutputForce(), 44
XMLDump.setOutputImage(), 44
XMLDump.setOutputInert(), 44
XMLDump.setOutputInit(), 45
XMLDump.setOutputLocalForce(), 45
XMLDump.setOutputLocalVirial(), 45
XMLDump.setOutputLocalVirialMatrix(), 45
XMLDump.setOutputMass(), 44
XMLDump.setOutputOrientation(), 44
XMLDump.setOutputPatch(), 45
XMLDump.setOutputPosition(), 44
XMLDump.setOutputPotential(), 44
XMLDump.setOutputQuaternion(), 44
XMLDump.setOutputRotangle(), 44
XMLDump.setOutputRotation(), 44
XMLDump.setOutputTorque(), 44
XMLDump.setOutputType(), 44
XMLDump.setOutputVelocity(), 44
XMLDump.setOutputVirial(), 44
XMLDump.setOutputVsite(), 45

C

CCPMD (*built-in class*), 105
CCPMD.setParams()
    built-in function, 105
CCPMD.setWall()
    built-in function, 105
CellList (*built-in class*), 51
CellList.setDataReproducibility()
    built-in function, 51
CellList.setNominalDim()
    built-in function, 51
CellList.setNominalWidth()

built-in function, 51
CenterForce (*built-in class*), 61
CenterForce.setPreNextShift()
built-in function, 62
ChangeType (*built-in class*), 123
ChangeType.setInterface()
built-in function, 124
ChangeType.setNPTargetType()
built-in function, 124
ChangeType.setPr()
built-in function, 123
ChangeType.setSite()
built-in function, 124
ChangeType.setWall()
built-in function, 124
chare.particle_set (*built-in class*), 16
ComputeInfo (*built-in class*), 54
ComputeInfo.setNdof()
built-in function, 54

D

DCDDump (*built-in class*), 46
DCDDump.unpbc()
built-in function, 46
DCDDump.unwrap()
built-in function, 46
DePolymerization (*built-in class*), 114
DePolymerization.setChangeTypeInReaction()
built-in function, 115
DePolymerization.setCountUnbonds()
built-in function, 115
DePolymerization.setCrisQualify()
built-in function, 115
DePolymerization.setDegradeAngle()
built-in function, 115
DePolymerization.setDegradeDihedral()
built-in function, 115
DePolymerization.setParams()
built-in function, 114
DePolymerization.setT()
built-in function, 115
DihedralForceAmberCosine (*built-in class*), 76, 89
DihedralForceAmberCosine.setParams()
built-in function, 76
DihedralForceHarmonic (*built-in class*), 74, 89
DihedralForceHarmonic.setCosFactor()
built-in function, 74
DihedralForceHarmonic.setParams()
built-in function, 74, 75
DihedralForceOPLSCosine (*built-in class*), 75
DihedralForceOPLSCosine.setParams()
built-in function, 75
DihedralForceRyckaertBellemans (*built-in class*), 76
DihedralForceRyckaertBellemans.setParams()
built-in function, 76
DihedralForceTable (*built-in class*), 79
DihedralForceTable.setParams()
built-in function, 79
DihedralForceTable.setPotential()
built-in function, 79
DPDEwaldForce (*built-in class*), 83
DPDEwaldForce.setBeta()
built-in function, 83
DPDEwaldForce.setParams()
built-in function, 83
DPDForce (*built-in class*), 120
DPDForce.setDPDVV()
built-in function, 121
DPDForce.setParams()
built-in function, 121
DPDForce.setT()
built-in function, 121
DPDGWVV (*built-in class*), 121
DPDGWVV.setLambda()
built-in function, 121
dump.data (*built-in class*), 15
dump.mst (*built-in class*), 15
dump.xml (*built-in class*), 15
DumpInfo (*built-in class*), 42
DumpInfo.dumpAnisotropy()
built-in function, 42
DumpInfo.dumpBoxSize()
built-in function, 42
DumpInfo.dumpParticleForce()
built-in function, 42
DumpInfo.dumpParticlePosition()
built-in function, 42
DumpInfo.dumpPotential()
built-in function, 42
DumpInfo.dumpPressTensor()
built-in function, 42
DumpInfo.dumpTypeTemp()
built-in function, 42
DumpInfo.dumpVirial()
built-in function, 42
DumpInfo.dumpVirialMatrix()
built-in function, 42
DumpInfo.setPeriod()
built-in function, 42
DynamicParticleSet (*built-in class*), 53

E

ENUFForce (*built-in class*), 85
ENUFForce.setParams()
built-in function, 85
EwaldForce (*built-in class*), 82
EwaldForce.setParams()

built-in function, 82
ExternalForce (*built-in class*), 107
ExternalForce.setForce()
built-in function, 107
ExternalForce.setParams()
built-in function, 107

F

force.angle (*built-in class*), 29
force.angle.setParams()
built-in function, 29
force.bond (*built-in class*), 26
force.bond.setParams()
built-in function, 26
force.dihedral (*built-in class*), 31
force.dihedral.setCosFactor()
built-in function, 31
force.dihedral.setParams()
built-in function, 31
force.dpd (*built-in class*), 35
force.dpd.setParams()
built-in function, 35
force.nonbonded (*built-in class*), 22
force.nonbonded.setParams()
built-in function, 22
force.nonbonded_c (*built-in class*), 24
force.nonbonded_c.setParams()
built-in function, 24

G

GBForce (*built-in class*), 115
GBForce.setParams()
built-in function, 115
GEMForce (*built-in class*), 62
GEMForce.setParams()
built-in function, 62
Generators (*built-in class*), 130
Generators.addMolecule()
built-in function, 130
Generators.outPutMol2()
built-in function, 130
Generators.outPutMST()
built-in function, 130
Generators.outPutXML()
built-in function, 130
Generators.setDimension()
built-in function, 130
Generators.setMinimumDistance()
built-in function, 130
Generators.setParam()
built-in function, 130

I

integration.bd (*built-in class*), 34
integration.bd.setParams()
built-in function, 34
integration.gwvv (*built-in class*), 34, 36
integration.nvt (*built-in class*), 33
integration.nvt.setT()
built-in function, 33

L

LangevinNVT (*built-in class*), 94
LangevinNVT.setGamma()
built-in function, 94
LangevinNVT.setT()
built-in function, 94
LangevinNVTRigid (*built-in class*), 95
LangevinNVTRigid.setGamma()
built-in function, 95
LangevinNVTRigid.setT()
built-in function, 95
LJConstrainForce (*built-in class*), 103
LJConstrainForce.addCylinder()
built-in function, 103
LJConstrainForce.addSphere()
built-in function, 103
LJConstrainForce.addWall()
built-in function, 103
LJConstrainForce.clcearWall()
built-in function, 103
LJConstrainForce.clearCylinder()
built-in function, 103
LJConstrainForce.clearSphere()
built-in function, 103
LJConstrainForce.setParams()
built-in function, 103
LJCoulombShiftForce (*built-in class*), 87
LJEwaldForce (*built-in class*), 63, 88
LJEwaldForce.setDispVirialCorr()
built-in function, 63, 88
LJEwaldForce.setEnergy_shift()
built-in function, 63, 88
LJEwaldForce.setParams()
built-in function, 63
LJForce (*built-in class*), 59
LJForce.setDispVirialCorr()
built-in function, 60
LJForce.setEnergy_shift()
built-in function, 60
LJForce.setParams()
built-in function, 60
LZWForce (*built-in class*), 117
LZWForce.setMethod()
built-in function, 117
LZWForce.setParams()
built-in function, 117

M

MDSCFForce (*built-in class*), 111
MDSCFForce.setNewVersion()
    built-in function, 111
MDSCFForce.setParams()
    built-in function, 111
MDSCFForce.setPeriodScf()
    built-in function, 111
MOL2Dump (*built-in class*), 43
MOL2Dump.deleteBoundaryBond()
    built-in function, 43
MOL2Dump.setChangeFreeType()
    built-in function, 43
Molecule (*built-in class*), 126
Molecule.setAngleDegree()
    built-in function, 126, 127
Molecule.setBody()
    built-in function, 128
Molecule.setBodyEvacuation()
    built-in function, 129
Molecule.setBondLength()
    built-in function, 126
Molecule.setBox()
    built-in function, 128
Molecule.setCharge()
    built-in function, 127
Molecule.setCris()
    built-in function, 128
Molecule.setCylinder()
    built-in function, 128
Molecule.setDiameter()
    built-in function, 127, 128
Molecule.setDihedralDegree()
    built-in function, 127
Molecule.setInert()
    built-in function, 127
Molecule.setInit()
    built-in function, 128
Molecule.setIsotactic()
    built-in function, 126
Molecule.setMass()
    built-in function, 127
Molecule.setMolecule()
    built-in function, 128
Molecule.setOrientation()
    built-in function, 127
Molecule.setParticleTypes()
    built-in function, 126
Molecule.setQuaternion()
    built-in function, 127
Molecule.setSphere()
    built-in function, 128
Molecule.setTopology()
    built-in function, 126

N

NeighborList (*built-in class*), 50
NeighborList.addExclusionsFromAngles()
    built-in function, 51
NeighborList.addExclusionsFromBodies()
    built-in function, 51
NeighborList.addExclusionsFromBonds()
    built-in function, 51
NeighborList.addExclusionsFromDihedrals()
    built-in function, 51
NeighborList.setDataReproducibility()
    built-in function, 51
NeighborList.setFilterDiameters()
    built-in function, 51
NeighborList.setNsq()
    built-in function, 51
NeighborList.setRCut()
    built-in function, 51
NeighborList.setRCutPair()
    built-in function, 51
NoseHooverNVT (*built-in class*), 92
NoseHooverNVT.setT()
    built-in function, 93
NPT (*built-in class*), 96
NPT.setP()
    built-in function, 96
NPT.setT()
    built-in function, 96
NPTMTK (*built-in class*), 97
NPTMTK.setAnisotropic()
    built-in function, 98
NPTMTK.setSemiisotropic()
    built-in function, 98
NPTMTK.setT()
    built-in function, 98
NPTMTKRigid (*built-in class*), 98
NPTMTKRigid.setAnisotropic()
    built-in function, 99
NPTMTKRigid.setSemiisotropic()
    built-in function, 99
NPTMTKRigid.setT()
    built-in function, 98, 99
NPTRigid (*built-in class*), 97
NPTRigid.setP()
    built-in function, 97
NPTRigid.setT()
    built-in function, 97
NVE (*built-in class*), 91
NVE.setZeroForce()
    built-in function, 91
NVERigid (*built-in class*), 91
NVTRigid (*built-in class*), 95
NVTRigid.setT()
    built-in function, 95

O

Object (*built-in class*), 129
Object.setRadius()
    built-in function, 129

P

PairForce (*built-in class*), 66
PairForce.setParams()
    built-in function, 66
PairForce.setShiftParams()
    built-in function, 66
PairForceTable (*built-in class*), 78
PairForceTable.setParams()
    built-in function, 78
PairForceTable.setPotential()
    built-in function, 78
ParticleSet (*built-in class*), 52
ParticleSet.getNumMembers()
    built-in function, 52
PBGBForce (*built-in class*), 116
PBGBForce.setAspheres()
    built-in function, 116
PBGBForce.setGUM()
    built-in function, 116
PBGBForce.setParams()
    built-in function, 116
PBGBForce.setPatches()
    built-in function, 116
PerformConfig (*built-in class*), 49
PFMEForce (*built-in class*), 112
PFMEForce.setNewVersion()
    built-in function, 112
PFMEForce.setPeriodPFME()
    built-in function, 112
Polymerization (*built-in class*), 112
Polymerization.generateAngle()
    built-in function, 113
Polymerization.generateDihedral()
    built-in function, 113
Polymerization.initExPoint()
    built-in function, 114
Polymerization.setChangeTypeInReaction()
    built-in function, 113
Polymerization.setEnergyBar()
    built-in function, 114
Polymerization.setExchangePr()
    built-in function, 113
Polymerization.setExchMode()
    built-in function, 114
Polymerization.setFrpMode()
    built-in function, 114
Polymerization.setFuncReactRule()
    built-in function, 113
Polymerization.setInitDieProb()
    built-in function, 114
Polymerization.setInitInitReaction()
    built-in function, 113
Polymerization.setInsertionMode()
    built-in function, 114
Polymerization.setInsertionPr()
    built-in function, 113
Polymerization.setMaxCris()
    built-in function, 113
Polymerization.setMinDisReactRule()
    built-in function, 114
Polymerization.setNewAngleType()
    built-in function, 113
Polymerization.setNewAngleTypeByPairs()
    built-in function, 113
Polymerization.setNewBondType()
    built-in function, 113
Polymerization.setNewBondTypeByPairs()
    built-in function, 113
Polymerization.setNewDihedralType()
    built-in function, 113
Polymerization.setPr()
    built-in function, 113
Polymerization.setPrFactor()
    built-in function, 113
Polymerization.setSgapMode()
    built-in function, 114
PPPMForce (*built-in class*), 84
PPPMForce.setParams()
    built-in function, 84

R

Reader (*built-in class*), 41
RNEMD (*built-in class*), 109
RNEMD.setPeriod()
    built-in function, 109
RNEMD.setProfVelPeriod()
    built-in function, 109
RNEMD.setSwapGroup()
    built-in function, 109
RNEMD.setSwapPeriod()
    built-in function, 109
RNEMD.setVelProfile()
    built-in function, 109

S

setHead()
    built-in function, 42
setPrecision()
    built-in function, 42
SLJForce (*built-in class*), 60
SLJForce.setEnergy_shift()
    built-in function, 61
SLJForce.setParams()

built-in function, 61
snapshot (*built-in class*), 14
snapshot.read (*built-in class*), 14
Sort (*built-in class*), 54

T

TranRigid (*built-in class*), 92
TranRigid.setRotDimension()
built-in function, 92
TranRigid.setTraDimension()
built-in function, 92

V

VariantConst (*built-in class*), 101
VariantLinear (*built-in class*), 101
VariantLinear.setPoint()
built-in function, 101
VariantSin (*built-in class*), 102
VariantSin.setPoint()
built-in function, 102
VariantWell (*built-in class*), 102
VariantWell.setPoint()
built-in function, 102
Vsite (*built-in class*), 90, 106
Vsite.setParams()
built-in function, 106

X

XMLDump (*built-in class*), 43
XMLDump.setOutput()
built-in function, 43
XMLDump.setOutputAngle()
built-in function, 45
XMLDump.setOutputBody()
built-in function, 44
XMLDump.setOutputBond()
built-in function, 45
XMLDump.setOutputCharge()
built-in function, 44
XMLDump.setOutputConstraint()
built-in function, 45
XMLDump.setOutputCris()
built-in function, 45
XMLDump.setOutputDiameter()
built-in function, 44
XMLDump.setOutputDihedral()
built-in function, 45
XMLDump.setOutputEllipsoid()
built-in function, 45
XMLDump.setOutputForce()
built-in function, 44
XMLDump.setOutputImage()
built-in function, 44
XMLDump.setOutputInert()
built-in function, 44
XMLDump.setOutputInit()
built-in function, 45
XMLDump.setOutputLocalForce()
built-in function, 45
XMLDump.setOutputLocalVirial()
built-in function, 45
XMLDump.setOutputLocalVirialMatrix()
built-in function, 45
XMLDump.setOutputMass()
built-in function, 44
XMLDump.setOutputOrientation()
built-in function, 44
XMLDump.setOutputPatch()
built-in function, 45
XMLDump.setOutputPosition()
built-in function, 44
XMLDump.setOutputPotential()
built-in function, 44
XMLDump.setOutputQuaternion()
built-in function, 44
XMLDump.setOutputRotangle()
built-in function, 44
XMLDump.setOutputRotation()
built-in function, 44
XMLDump.setOutputTorque()
built-in function, 44
XMLDump.setOutputType()
built-in function, 44
XMLDump.setOutputVelocity()
built-in function, 44
XMLDump.setOutputVirial()
built-in function, 44
XMLDump.setOutputVsite()
built-in function, 45
XMLReader (*built-in class*), 41

Z

ZeroMomentum (*built-in class*), 104