galamost Documentation

Release 4.0.1

Youliang Zhu

Nov 15, 2019

General introduction

1 Introduction **1**

2 Functionalities **3**
2.1 1 Unique features . . . 3
2.2 2 Unique features among GPU packages . . . 3
2.3 3 Common features . . . 4

3 Installation **5**
3.1 Source code . . . 5

4 Usage **7**

5 Units **9**
5.1 Fundamental Units . . . 9
5.2 Temperature units (thermal energy) . . . 9
5.3 Charge units . . . 9
5.4 Common derived units . . . 10
5.5 Example physical units . . . 10

6 Data **11**
6.1 Data format . . . 11
6.1.1 XML format . . . 11
6.2 Data input . . . 15
6.2.1 XML reader . . . 15
6.2.2 Binary reader . . . 15
6.3 Data output . . . 15
6.3.1 Collective information . . . 15
6.3.2 MOL2 dump . . . 16
6.3.3 XML dump . . . 17
6.3.4 DCD trajectory dump . . . 19
6.3.5 GALAMOST binary dump . . . 19

7 System **21**
7.1 Information . . . 21
7.1.1 Perform configuration . . . 21
7.1.2 All information . . . 21
7.2 Particle list . . . 22

7.2.1 Neighbor list . . . 22
7.2.2 Cell list . . . 23
7.3 Particle set . . . 23
7.4 Information computation . . . 24
7.5 Memory sorting . . . 24

8 Force field **27**
8.1 Short range non-bonded interactions . . . 27
8.1.1 Lennard-Jones (LJ) interaction . . . 27
8.1.2 Shift Lennard-Jones (LJ) interaction . . . 28
8.1.3 Linear molecule π-π interaction . . . 29
8.1.4 Generalized exponential model . . . 30
8.1.5 Lennard-Jones and Ewald (short range) interaction . . . 31
8.1.6 Pair interaction . . . 32
8.2 Bonded interactions . . . 34
8.2.1 Bond stretching . . . 34
8.2.2 Angle bending . . . 37
8.2.3 Dihedral torsion . . . 40
8.3 Numerical interaction . . . 42
8.3.1 Theory description . . . 42
8.3.2 Non-bonded interaction . . . 43
8.3.3 Bond interaction . . . 43
8.3.4 Angle interaction . . . 44
8.3.5 Dihedral interaction . . . 44
8.4 Coulomb interaction . . . 45
8.4.1 Ewald summation theory . . . 45
8.4.2 Ewald (short-range) . . . 46
8.4.3 Ewald for DPD (short-range) . . . 47
8.4.4 PPPM (long-range) . . . 47
8.4.5 ENUF (long-range) . . . 48
8.5 Read force parameters . . . 49
8.5.1 GALAMOST force field format . . . 49
8.5.2 Use GALAMOST force fields . . . 51
8.5.3 Use GROMACS force fields . . . 53
8.5.4 Convert GROMACS files . . . 54

9 Constraint **55**
9.1 Variant . . . 55
9.1.1 Variant Const . . . 55
9.1.2 Variant Linear . . . 55
9.1.3 Variant Sin . . . 56
9.1.4 Variant Well . . . 56
9.2 External interaction . . . 56
9.2.1 Axial stretching . . . 56
9.2.2 External force . . . 57
9.3 Space constraint . . . 57
9.3.1 Bounce back condition . . . 57
9.3.2 LJ surface force . . . 58
9.4 Remove CM momentum . . . 58
9.5 Constant chemical potential . . . 59
9.6 Bond constraint . . . 60
9.7 Virtual site . . . 60

10 Modules **63**

10.1 MD-SCF . . . 63
10.1.1 MDSCF force . . . 63
10.1.2 MDSCF electrostatic force . . . 64
10.2 Polymerization . . . 64
10.2.1 Polymerization model . . . 64
10.2.2 Depolymerization model . . . 66
10.3 Anisotropic particle . . . 66
10.3.1 Gay-Berne model . . . 66
10.3.2 Soft anisotropic model . . . 68
10.4 Dissipative particle dynamics . . . 70
10.4.1 DPD force . . . 70
10.4.2 GWVV integration . . . 72
10.4.3 Coulomb interaction in DPD . . . 72

11 Integration **75**
11.1 NVE ensemble . . . 75
11.1.1 NVE thermostat . . . 75
11.1.2 NVE for rigid body . . . 75
11.1.3 NVE for rigid body with tunable freedoms . . . 76
11.2 NVT ensemble . . . 76
11.2.1 Nose Hoover thermostat . . . 76
11.2.2 Berendsen thermostat . . . 77
11.2.3 Andersen thermostat . . . 77
11.2.4 Brownian dynamic thermostat . . . 78
11.2.5 NVT for rigid body . . . 79
11.2.6 Brownian dynamic for rigid body . . . 79
11.3 NPT ensemble . . . 80
11.3.1 Andersen barostat . . . 80
11.3.2 NPT for rigid body . . . 80
11.3.3 Martyna-Tobias-Klein barostat . . . 81
11.3.4 Martyna-Tobias-Klein barostat for rigid body . . . 82

12 Application **85**
12.1 Modules management . . . 85
12.2 Multi-stage simulation . . . 86
12.3 Two-dimensional simulation . . . 86

13 Molgen **89**
13.1 Molecule generator description . . . 89
13.2 Molecule definition . . . 90
13.3 Objects definition . . . 93
13.4 Generator definition . . . 93

14 GalaTackle **95**
14.1 Usage . . . 95
14.2 Functions . . . 96
14.2.1 1 Rg^2: . . . 96
14.2.2 2 Ed^2: . . . 96
14.2.3 3 RDF: . . . 96
14.2.4 4 bond_distri: . . . 97
14.2.5 5 angle_distri: . . . 97
14.2.6 6 dihedral_distri: . . . 97
14.2.7 7 stress tensor: . . . 98
14.2.8 8 density: . . . 98
14.2.9 9 unwrapping: . . . 98

14.2.10 10 MSD: . . . 98
14.2.11 11 RDF-CM: . . . 98
14.2.12 12 MSD-CM: . . . 98
14.2.13 13 ents: . . . 98
14.2.14 14 strfac: . . . 98
14.2.15 15 domain size: . . . 99
14.2.16 16 dynamic strfac: . . . 99
14.2.17 17 config check: . . . 99
14.2.18 18 RDF between types: . . . 99
14.2.19 19 XML conversion: . . . 99

15 License **101**

16 Original sources **103**

17 Indices and tables **107**

Index **109**

CHAPTER 1

Introduction

Coarse-grained molecular dynamics (CGMD) simulations are exceptionally important in the research field of soft matter. In general, CGMD targets problems typically at nano- to meso-scales that are not easily coped with using all-atom molecular dynamics simulations. Directly inspired and derived from pioneering work of HOOMD-blue developed by Prof. Glotzer's group in the University of Michigan, also referred to some other powerful simulation tools such as LAMMPS, GROMACS, MDynamix, and OCCAM, GALAMOST was designed to provide a set of open-source and specific tools of employing NVIDIA GPUs, to accelerate CGMD simulations.

These tools are, for example:

1. Modelling "polymerizations" in a stochastic way in CGMD simulations. The topological connections between beads could be changed according to pre-defined probability. This tool can be used in the studies of chain-growth polymerization, reversible reaction, exchange reaction, and so on.
2. Using anisotropic particle models to describe soft particles as well as rigid ellipsoidal particles with/without patches by combining harmonic repulsive potential/Gay-Berne potential with anisotropic "patchy" potentials.
3. 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.
4. Reading in numerical potentials derived from iterative Boltzmann inversion, inverse Monte Carlo, or other structure-based bottom-up coarse-graining methods (Mirzoev et al., Comput. Phys. Commun. 237, 263, 2019), and applying the potentials in CGMD simulations.

In GALAMOST, it is also possible to perform conventional CGMD, Brownian dynamics, and dissipative particle dynamics simulations with various potential forms. The trajectories obtained in GALAMOST could be visualized and analyzed by OVITO.

CHAPTER 2

Functionalities

2.1 1 Unique features

1.1 Polymerization reaction model

- 1.1.1 Chain-growth polymerization
- 1.1.2 Step-growth polymerization
- 1.1.3 Ligand exchange and chain migration reaction
- 1.1.4 Insertion polymerization
- 1.1.5 Depolymerization model

1.2 Anisotropic particle model

- 1.2.1 Janus particle model
- 1.2.2 Patchy particle model
- 1.2.3 Uniaxial and biaxial Gay-Berne model with patches
- 1.2.4 Hybrid chain model of isotropic particles and anisotropic particles

2.2 2 Unique features among GPU packages

2.1 Electrostatics model

- 2.1.1 Electrostatics in MD-SCF
- 2.1.2 Electrostatics in DPD
- 2.1.3 Ewald summation method, based on Non-Uniform FFTs

2.2 MD-SCF: a hybrid particle-field molecular dynamics technique

2.3 Pi-Pi interaction model

2.4 Dissipative particle dynamics with GWVV integration

2.5 Numerical potential method with spline interpolation

2.6 Multiple particle collision dynamics

2.7 Axial stretching method for measuring mechanical properties of materials

2.8 Constant Chemical Potential Molecular Dynamics

2.9 Coarse-grained three-site-per-nucleotide model of DNA

2.10 Double-time step integration method

2.11 LINCS bond constraint method

2.12 Virtual interaction site method

2.13 Integrated tempering sampling method

2.14 Many-body dissipative particle dynamics

2.15 Reverse non-equilibrium molecular dynamics

2.16 Bounce back boundary conditions for plane, sphere, and cylinder

2.3 3 Common features

3.1 Rigid body model

3.2 Size-defined particle model (with shifted LJ potential)

3.3 Particle-Particle Particle-Mesh

3.4 Brownian dynamics

CHAPTER 3

Installation

3.1 Source code

The entire GALAMOST package is a Free Software under the GNU General Public License. The package is mainly distributed as source code and binary program. The code and binary program can be downloaded from our website http://galamost.ciac.jl.cn/Download.php. The developing codes are available from https://bitbucket.org/galamostdevelopergroup/source-code/src/master/. At present, only Linux operating systems are supported. Here is the guide for installation by the codes.

1. Requrements:

```
1. Python >=2.6
2. Boost library >=1.53.0
3. NVIDIA CUDA Toolkit >= 7.0
4. MPI(MVAPICH2 >= 2.3 or OpenMPI >= 4.0.0)
   For MVAPICH2, environment variable "export MV2_ENABLE_AFFINITY=0" is necessary␣
↪to
   avoid multiple processes running on a same CPU core

Note: MPI is only needed for version 4 with '--mpi=on' in configuration by␣
↪default.
The MPI is not need with '--mpi=off'.
```

2. Before compiling and installing of source code, the compiling system should be configured first by `configure`.

   Examples:

```
./configure --prefix=/opt/galamost4
```

More configuration options are given here:

Options	Functions	Examples	Defaults
`--prefix`	Installation path	`--prefix=/opt/galamost4`	`--prefix=/opt/galamost4`
`--cuda_arch`	Compute capability of GPU	`--cuda_arch=75`	Automatically detecting
`--precision`	Precision format	`--precision=double`	“–precision=single
`--gprof`	Profiling tool	`--gprof=on`	`--gprof=off`
`--gdb`	GDB tool	`--gdb=on`	`--gdb=off`
`--cuda`	CUDA toolkit path	`--cuda=/usr/local/cuda-7.5`	Automatically detecting
`--gpu_mpi`	GPU direct communication	`--gpu_mpi=on`	`--gpu_mpi=off`
`--mpi`	Switch MPI	`--mpi=on`	`--mpi=on`
`--mpi_dir`	MPI path	`--mpi_dir=/usr/local`	Automatically detecting
`--boost`	Boost path	`--boost=/usr`	Automatically detecting
`--python`	Python path	`--python=/usr`	Automatically detecting

If the automatical detection is failed, please set the value explicitly. The compute capability will be automatically detected. If the detection failed or the machine equipped with multiple GPUs with different compute capabilitied, you could manually set the compute capability by `--cuda_arch`. The compute capability of some NVIDIA GPU is listed. For more GPUs, please visit https://developer.nvidia.com/cuda-gpus.

GPUs	Compute Capability
Tesla V100	70
Tesla P100	60
Tesla P40	61
Tesla P4	61
NVIDIA TITAN V	70
NVIDIA TITAN Xp	61
NVIDIA TITAN X	61
GeForce GTX 1080 Ti	61

3. After configuration, a Makefile file will be generated in your current directory. Then you can compile and install the package by `make install`.

   Examples:

```
make install -j4
# where -j indicates the number of threads to compile the code.
```

CHAPTER 4

Usage

More GALAMOST tutorials and examples are available from https://nbviewer.jupyter.org/github/zhuyouliang/GALAMOST/blob/master/tutorials/index.ipynb.

The C++ and CUDA C code has been written as an extended module of Python. We can use the functions of GALAMOST by loading and calling its modules through a Python script. With the prepared script, you should now be able to try running GALAMOST as:

Examples:

```
./yourscript.gala --gpu=0 >a.log&
```

Where you may specify the GPU id with the `--gpu=` option and output the screen information into `a.log` file. If the script file has no executive permission, the command of `chmod +x yourscript.gala` should be executed before running above command. An alternative running command which could ignore the executive permission of the script:

Examples:

```
python yourscript.gala --gpu=0 >a.log&
```

For version 4, you could use multiple GPU for parallel computation by following commands:

Examples:

```
mpirun -n 4 python yourscript.gala --gpu=0,1,2,3 >a.log&
```

Here is an example of script for DPD simulation. The head of GALAMOST script usually is:

Examples:

```
#!/usr/bin/python
import sys
sys.path.append('/opt/galamost3/lib')
import galamost
```

(continues on next page)

(continued from previous page)

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

where the first paragraph sets the path of the installed library of GALAMOST `galamost.so` for loading the Python extensible modules of GALAMOST. The second paragraph is used for parsing GPU id from the executive command.

Then, reading the configuration from a prepared XML file by *XmlReader* and building up perform configuration of GPU by *PerformConfig*, a system information (*AllInfo*) can be built up by

Examples:

```
filename = 'A.xml'
build_method = galamost.XmlReader(filename)
perform_config = galamost.PerformConfig(_options.gpu)
all_info = galamost.AllInfo(build_method,perform_config)
```

After that, we need to build up an application with *Application* which will call following defined and added objects.

Examples:

```
dt = 0.01
app = galamost.Application(all_info, dt)
```

Further, we should define the wanted objects by the classes of GALAMOST and pass them to the application, such as the following example: non-bonded DPD force (*DpdForce*), NVT thermostat with GWVV algorithm (*DpdGwvv*), and information analysis methods (*ComputeInfo*).

Examples:

```
neighbor_list = galamost.NeighborList(all_info, 1.0 ,0.05) # (,rcut,rbuffer)
dpd=galamost.DpdForce(all_info,neighbor_list,1.0, 12345) # (,,rcut,seed)
dpd.setParams('A', 'A', 25.0, 3.0) # (type1, type2, alpha, sigma)
dpd.setParams('A', 'B', 40.0, 3.0) # (type1, type2, alpha, sigma)
dpd.setParams('B', 'B', 25.0, 3.0) # (type1, type2, alpha, sigma)
app.add(dpd)

group = galamost.ParticleSet(all_info, "all" )
comp_info = galamost.ComputeInfo(all_info, group)
Gwvv = galamost.DpdGwvv(all_info, group)
app.add(Gwvv)

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

The tail of script usually sets the number of time steps to run, and the function of analysis of neighbor list (*NeighborList*) etc.

Examples:

```
app.run( 10000)
neighbor_list.printStats()
```

CHAPTER 5

Units

GALAMOST 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.

5.1 Fundamental Units

The three fundamental units are:

- distance - σ
- energy - ε
- mass - ⇳

5.2 Temperature units (thermal energy)

GALAMOST 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 $T^* = k_B T/\varepsilon$.

5.3 Charge units

The charge used in GALAMOST 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 GALAMOST.

5.4 Common derived units

Here are some commonly used derived units:

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

5.5 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 - $⇳ = \mathrm{amu}$

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

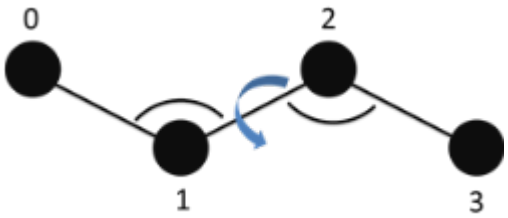
CHAPTER 6

Data

6.1 Data format

6.1.1 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>
```

(continues on next page)

(continued from previous page)

```
<velocity num="4">
1 2 3
1 0 0
3 -2 1
0 1 1
</velocity>
<type num="4">
A
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
```

(continues on next page)

(continued from previous page)

```
 </charge>

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

6.2 Data input

class Reader
The basic class of *XmlReader* and *BinaryReader*.

6.2.1 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 = galamost.XmlReader(filename)
# builds up a parser object for the input XML file.
```

6.2.2 Binary reader

class BinaryReader(*filename*)
The constructor of GALAMOST binary file parser object.

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

Example:

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

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

6.3 Data output

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

6.3.2 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 = galamost.Mol2Dump(all_info, 'particles')
mol2.setPeriod(100000)
mol2.deleteBoundaryBond(True)
app.add(mol2)
```

6.3.3 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 positions (default value is true).

setOutputType(*bool switch*)
Outputs particle types (default value is true).

setOutputImage(*bool switch*)
Outputs images.

setOutputVelocity(*bool switch*)
Outputs velocities.

setOutputMass(*bool switch*)
Outputs masses.

setOutputCharge(*bool switch*)
Outputs charges.

setOutputDiameter(*bool switch*)
Outputs diameters.

setOutputBody(*bool switch*)
Outputs bodies.

setOutputVirial(*bool switch*)
Outputs virials.

setOutputForce(*bool switch*)
Outputs forces.

setOutputOrientation (*bool switch*)
Outputs orientations.

setOutputQuaternion (*bool switch*)
Outputs quaternions.

setOutputRotation (*bool switch*)
Outputs rotation velocities.

setOutputTorque (*bool switch*)
Outputs torques.

setOutputInert (*bool switch*)
Outputs inert tensors.

setOutputInit (*bool switch*)
Outputs initiator indicators.

setOutputCris (*bool switch*)
Outputs cross-linking indicators.

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 forces for a Force object.

setOutputLocalVirial (*Force object*)
Outputs particle virials for a Force object.

setOutputLocalVirialMatrix (*Force object*)
Outputs particle virial matrixes for 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 = galamost.XmlDump(all_info, 'particles')
xml.setOutput(['image', 'bond'])
xml.setPeriod(100000)
app.add(xml)
```

6.3.4 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 = galamost.DcdDump(all_info, 'particles', True)
dcd.unwrap(True)
dcd.setPeriod(100000)
app.add(dcd)
```

6.3.5 GALAMOST binary dump

class BinaryDump(*all_info*, *filename*)
Constructor of an object to dump GALAMOST 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 = galamost.BinaryDump(all_info, 'particle')
binary.setOutput(['image', 'bond'])
binary.setPeriod(10000)
app.add(binary)
```

CHAPTER 7

System

7.1 Information

7.1.1 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 = galamost.PerformConfig(_options.gpu)
```

7.1.2 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 = galamost.XmlReader(filename)
perform_config = galamost.PerformConfig(_options.gpu)
all_info = galamost.AllInfo(build_method,perform_config)
```

7.2 Particle list

7.2.1 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.

addExclusionsFromBodys()
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 = galamost.NeighborList(all_info, 3.0 ,0.4)
```

7.2.2 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 = galamost.CellList(all_info)
```

7.3 Particle set

class ParticleSet(*all_info*, *keyword*)
The constructor of particle set object with keyword.

Parameters

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

class ParticleSet(*all_info*, *min*, *max*)
The constructor of particle set object with keyword.

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.

Example:

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

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

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

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

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

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

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

7.4 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 = galamost.ComputeInfo(all_info, group)
```

7.5 Memory sorting

class Sort(*all_info*)
The constructor of a memory sort object.

Parameters **all_info** (AllInfo) – The system information

Example:

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

CHAPTER 8

Force field

8.1 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 in GALAMOST.

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`

8.1.1 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 = galamost.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.
```

8.1.2 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 = galamost.SljForce(all_info, neighbor_list, 3.0)
slj.setParams('A', 'A', 1.0, 1.0, 1.0)
slj.setEnergy_shift()
app.add(slj)
```

8.1.3 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:

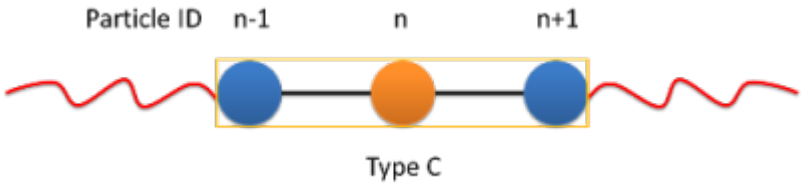
$$\begin{aligned} V_{\pi-\pi}(r,\theta) = & \quad -\epsilon \cos^2\theta(1-r) \quad r < r_{\mathrm{cut}} \\ = & \quad 0 \quad r \geq r_{\mathrm{cut}} \end{aligned}$$

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

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

8.1.5 Lennard-Jones and Ewald (short range) interaction

Description:

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

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 = galamost.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)
```

8.1.6 Pair interaction

LJ9_6 interaction

Description:

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

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)

Script commands

Harmonic repulsion

Description:

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

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

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

8.2 Bonded interactions

8.2.1 Bond stretching

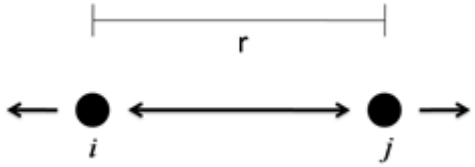
Overview

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

FENE bond potential

Description:

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

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

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

8.2.2 Angle bending

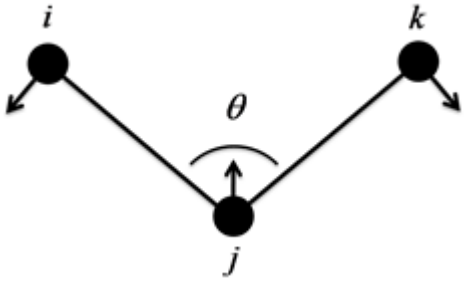
Overview

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*



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

Harmonic cosine angle potential

Description:

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

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

Cosine angle potential

Description:

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

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.

Parameters **all_info** (AllInfo) – 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 = galamost.AngleForceCos(all_info)
angleforce.setParams('P-G-G', 25.000, 120.000)
app.add(angleforce)
```

8.2.3 Dihedral torsion

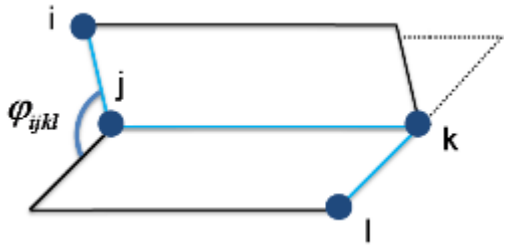
Overview

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	`DihedralForceHarmonic`
Harmonic dihedral(improper) potential	`DihedralForceHarmonic`
OPLS dihedral potential	`DihedralForceOplsCosine`



Harmonic dihedral potential

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

Harmonic dihedral(improper) potential

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 set 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.

Example:

```
dihedralforce = galamost.DihedralForceHarmonic(all_info)
dihedralforce.setParams('A-B-B-A', 10.0, 0.0, galamost.DihedralForceHarmonic.Prop.
↪improper)
app.add(dihedralforce)
```

OPLS dihedral potential

Description:

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

Coefficients:

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

Note: The dihedral angles set 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.

Example:

```
dihedralforce = galamost.DihedralForceOplsCosine(all_info)
dihedralforce.setParams('C_33-C_32-C_32-C_32', 0.0, 2.95188, -0.566963, 6.57940, 0.0)
app.add(dihedralforce)
```

8.3 Numerical interaction

8.3.1 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.

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

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

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

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

8.4 Coulomb interaction

8.4.1 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 in GALAMOST 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 = galamost.ParticleSet(all_info, "charge")

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

# reciprocal space
pppm = galamost.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)
```

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

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

8.4.4 PPPM (long-range)

Description:

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

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

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

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

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

8.5 Read force parameters

8.5.1 GALAMOST force field format

GALAMOST format of 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>

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

An example of GALAMOST 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
```

(continues on next page)

(continued from previous page)

```
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
</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
```

(continues on next page)

(continued from previous page)

```
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>
```

8.5.2 Use GALAMOST force fields

Description:

Force fields in GALAMOST 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)
```

8.5.3 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)
```

8.5.4 Convert GROMACS files

Description:

Convert GROMACS files to GALAMOST 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 GALAMOST force field will be generated.

CHAPTER 9

Constraint

9.1 Variant

9.1.1 Variant Const

class VariantConst(*value*)

The constructor of a constant value method.

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

Example:

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

9.1.2 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 = galamost.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.
```

9.1.3 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 = galamost.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.
```

9.1.4 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 = galamost.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.
```

9.2 External interaction

9.2.1 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 = galamost.VariantLinear()
v.setPoint(0, 31) # time step, box length.
v.setPoint(100000, 60)
```

(continues on next page)

(continued from previous page)

```
axs = galamost.AxialStretching(all_info, group)
axs.setBoxLength(v, 'Y')
app.add(axs)
```

9.2.2 External force

class ExternalForce(*all_info*, *group*)

The constructor of an external force object for a group of particles. The external force F will be added on the group of particles and each particle have a F/N, where N is the number of particles.

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 = galamost.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 = galamost.ParticleSet(all_info, "A")
ef = galamost.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)
```

9.3 Space constraint

9.3.1 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.

Example:

```
bbc = galamost.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)
```

9.3.2 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.

Example:

```
ljc = galamost.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)
```

9.4 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.

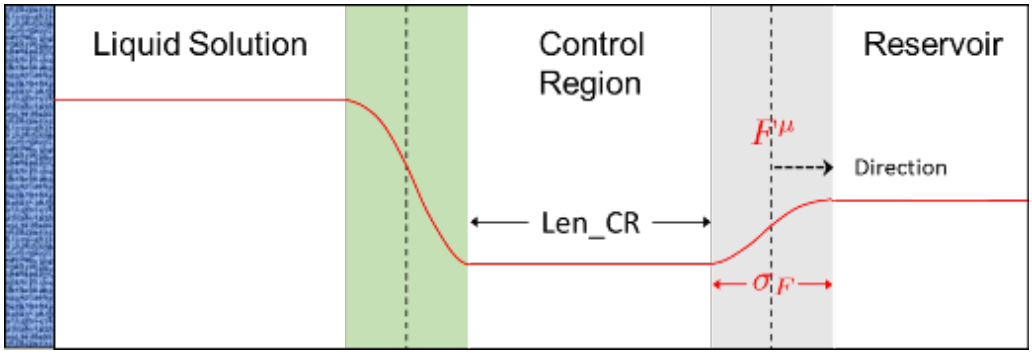
Example:

```
zm = galamost.ZeroMomentum(all_info)
zm.setPeriod(10)
app.add(zm)
```

9.5 Constant chemical potential

Note: GALAMOST v4.0.0's function

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:

$$
\begin{aligned}
F^{\mu}(z) &= k(n^{CR} - n_0)G(z, Z_F) \\
G(z - Z_F) &= \frac{1}{4\omega}\left[1 + cosh\left(\frac{z - Z_F}{\omega}\right)\right]^{-1}
\end{aligned}
$$

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 `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 = galamost.ParticleSet(all_info, 'S')
ccp = galamost.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)
```

9.6 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 = galamost.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)
```

9.7 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 = galamost.Vsite(all_info)#virtual interaction sites
vs.setParams('v', 0.128012065, 0.128012065, 0.0, galamost.Vsite.VST.v3 )
app.add(vs)
```

CHAPTER 10

Modules

10.1 MD-SCF

10.1.1 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 = galamost.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.
```

10.1.2 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 = galamost.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)
```

10.2 Polymerization

10.2.1 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 to connect to type 2 by type1.

setMaxCris(*string type*, *unsigned int cris_max*)

specifies the upper limit number of bonds generated by reaction.

setNewBondType(*string bondtype*)

specifies the type of newly generated bonds by reaction.

setFuncReactRule(*bool switch*, *float K*, *float r_0*, *float b_0*, *float epsilon0*, *Func 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).

setMinDisReactRule(*bool switch*)

switches the rule of the reaction only with the nearest particle.

initExPoint()

switches on initializing reactive point for exchange reaction.

Example:

```
reaction = galamost.Polymerization(all_info, neighbor_list, 1.12246 ,16361)
reaction.setFuncReactRule(True, 1250.000, 1.0,0.470, 10.0, galamost.
↪Polymerization.Func.harmonic)
reaction.setPr(0.002)
reaction.setMaxCris('B',3)
# sets the connected bond upper limited number.
reaction.setPeriod(50)
app.add(reaction)
```

10.2.2 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*, *Func 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.

Example:

```
reaction = galamost.DePolymerization(all_info, 1.0, 16361)
reaction.setParams('sticky', 10.0, 1.5, 0.96, 10.0, 0.2, galamost.
↪DePolymerization.Func.harmonic)
# sets bondname, K, r_0, b_0, epsilon0, Pr, and function.
reaction.setPeriod(1)
# sets how many steps to react.
app.add(reaction)
```

10.3 Anisotropic particle

10.3.1 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 = galamost.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 = galamost.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 = galamost.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>
```

10.3.2 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 = galamost.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 = galamost.BerendsenAniNvt(all_info, group, comp_info, 1.0, 0.3, 0.1)
app.add(bere)
```

Multiple patch particle model

Description:

$$U_{ij} = \begin{cases} \frac{\alpha_{ij}^R d_{ij}}{\mu}\left(1-\frac{r_{ij}}{d_{ij}}\right)^{\mu} - \sum_{\kappa=1}^{M_i}\sum_{\lambda=1}^{M_j} f^{\nu}\left(\mathbf{n}_i^{\kappa}, \mathbf{n}_j^{\lambda}, \mathbf{r}_{ij}\right) \frac{\alpha_{ij}^A d_{ij}}{\mu}\left[\frac{r_{ij}}{d_{ij}} - \left(\frac{r_{ij}}{d_{ij}}\right)^{\mu}\right] & r_{ij} \le d_{ij} \\ 0 & r_{ij} > d_{ij}, \end{cases}$$

$$f\left(\mathbf{n}_i^{\kappa}, \mathbf{n}_j^{\lambda}, \mathbf{r}_{ij}\right) = \begin{cases} \cos\frac{\pi\theta_i^{\kappa}}{2\theta_m^{\kappa}} \cos\frac{\pi\theta_j^{\lambda}}{2\theta_m^{\lambda}} & \text{if } \cos\theta_i^{\kappa} \ge \cos\theta_m^{\kappa} \text{ and } \cos\theta_j^{\lambda} \ge \cos\theta_m^{\lambda} \\ 0 & \text{otherwise.} \end{cases}$$

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 = galamost.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 = galamost.ParticleSet(all_info, 'body')
rigidnvt = galamost.NvtRigid(all_info, bgroup, 1.0, 0.2)
app.add(rigidnvt)

nbgroup = galamost.ParticleSet(all_info,'non_body')
comp_info_nb = galamost.ComputeInfo(all_info, nbgroup)
nh = galamost.NoseHooverNvt(all_info, nbgroup, comp_info_nb, 1.0, 1.0)#( ,
↪temperature, tau)
app.add(nh)
```

10.4 Dissipative particle dynamics

10.4.1 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 = galamost.DpdForce(all_info, neighbor_list, 1.0, 12345)
dpd.setParams('A', 'A', 25.0, 3.0)
app.add(dpd)
```

10.4.2 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 = galamost.DpdGwvv(all_info, group)
app.add(gwvv)
```

10.4.3 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 in GALAMOST 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 = galamost.ParticleSet(all_info, "charge")
kappa=0.2

# real space
ewald = galamost.DPDEwaldForce(all_info, neighbor_list, group_charge, 3.64)
↪#(,,rcut)
```

(continues on next page)

(continued from previous page)

```
ewald.setParams(kappa)
app.add(ewald)

# reciprocal space
enuf = galamost.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* for the input of GALAMOST. 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 $q^* = z\sqrt{f/(\sigma k_BT\epsilon_r)}$. The z is the valence of ion.
>
> Here is a *Molgen* script for polyelectrolyte.

Example:

```
#!/usr/bin/python
import sys
sys.path.append('/opt/galamost3/lib')
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)
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)
```

(continues on next page)

(continued from previous page)

```
gen.addMolecule(mol2,75)
gen.addMolecule(mol3,75)
gen.addMolecule(mol4,9925)

gen.outPutXml("ps0")
```

CHAPTER 11

Integration

11.1 NVE ensemble

Overview

NVE thermostat	`Nve`
NVE for rigid body	`NveRigid`
NVE for rigid body with tunable freedoms	`TranRigid`

11.1.1 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 = galamost.ParticleSet(all_info,'all')
thermo = galamost.Nve(all_info,group)
app.add(thermo)
```

11.1.2 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 = galamost.ParticleSet(all_info, 'body')
rigidnve = galamost.NveRigid(all_info, bgroup)
app.add(rigidnve)
```

11.1.3 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:

```
bdrigidnvt = galamost.TranRigid (all_info, bgroup)
bdrigidnvt.setTraDimension(True, True, True)
bdrigidnvt.setRotDimension(True, True, True)
app.add(bdrigidnvt)
```

11.2 NVT ensemble

Overview

Nose Hoover thermostat	*NoseHooverNvt*
Berendsen thermostat	*BerendsenNvt*
Andersen thermostat	*AndersenNvt*
Brownian dynamic thermostat	*BdNvt*
NVT for rigid body	*NvtRigid*
Brownian dynamic for rigid body	*BdNvtRigid*

11.2.1 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 = galamost.ParticleSet(all_info, 'all')
comp_info = galamost.ComputeInfo(all_info, group)
nh = galamost.NoseHooverNvt(all_info, group, comp_info, 1.0, 0.5)
app.add(nh)
```

11.2.2 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.

11.2.3 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 = galamost.AndersenNvt(all_info,group,1.0,10.0, 12345)
app.add(an)
```

11.2.4 Brownian 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 BdNvt (*all_info*, *group*, *T*, *seed*)
The constructor of a Brownian 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 = galamost.ParticleSet(all_info, 'all')
bdnvt = galamost.BdNvt(all_info, group, 1.0, 123)
app.add(bdnvt)
```

11.2.5 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 = galamost.ParticleSet(all_info, 'body')
rigidnvt = galamost.NvtRigid(all_info, bgroup, 1.0, 10.0)
app.add(rigidnvt)
```

11.2.6 Brownian dynamic for rigid body

Please see *Brownian dynamic thermostat* for the theory.

class BdNvtRigid(*all_info*, *group*, *T*, *seed*)
The constructor of a Brownian 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 Brownian method with a constant value.

setGamma(*const std::string & type*, *float gamma*)
specifies the gamma of Brownian 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 = galamost.ParticleSet(all_info, 'body')
bdrigidnvt = galamost.BdNvtRigid(all_info, bgroup, 1.0, 123)
app.add(bdrigidnvt)
```

11.3 NPT ensemble

Overview

Andersen barostat	`Npt`
NPT for rigid body	`NptRigid`

11.3.1 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 =galamost.Npt(all_info, group, comp_info, comp_info, 1.0, 0.2, 0.5, 0.1)
app.add(npt)
```

11.3.2 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 = galamost.ParticleSet(all_info,'all')
comp_info = galamost.ComputeInfo(all_info, group)

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

rigidnpt = galamost.NptRigid(all_info, bgroup, comp_info_b, comp_info, 1.0, 0.1,␣
↪1.0, 1.0)
app.add(rigidnpt)
```

11.3.3 Martyna-Tobias-Klein barostat

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

Note: GALAMOST v4.0.0's function

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

npt = galamost.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)
```

11.3.4 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.

Note: GALAMOST v4.0.0's function

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

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

npt = galamost.NPTMTK(all_info, groupb, comp_infob, comp_info, 1.0, 0.1, 0.5, 1.0)
npt.setSemiisotropic(0.1, 0.1)
app.add(npt)
```

CHAPTER 12

Application

12.1 Modules management

GALAMOST is a modular program. Application manages and calls modules and run simulation. 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(*boost::shared_ptr<*> object*)
adds an object to the application.

remove(*boost::shared_ptr<*> 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 = galamost.Application(all_info, dt)
# builds up an application.
```

(continues on next page)

(continued from previous page)

```
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 = galamost.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
```

(continues on next page)

(continued from previous page)

```
   -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
import sys
sys.path.append('/opt/galamost4/lib')
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 13

Molgen

13.1 Molecule generator description

The initial configurations can be generated by employing a Plug-Ins (named Molgen) of GALAMOST and output in XML format.

The head of script:

```
#!/usr/bin/python
import sys
sys.path.append('/opt/galamost3/lib')
import molgen
# sets the path of molgen.so and imports an extended Python module of molgen.
```

Two steps to employ this tool: 1. Defines each single molecule or object (the object module is usually used to describe the objects with various shape). 2. Sets 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.

Examples:

```
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)
```

One or more molecules or objects can be defined in this way. Then the box size and the number of these molecules should be specified by "Generators".

Examples:

```
gen=molgen.Generators(10,10,10)
gen.addMolecule(mol1,10)
```

(continues on next page)

(continued from previous page)

```
gen.setMinimumDistance(0.7)
gen.outPutXml("test")
```

The configuration of a molecule can be partially read from a XML file and partially defined in the script.

Examples:

```
mol0=molgen.Molecule("sphere.xml",65)
# total 65 particles with 60 read from "sphere.xml".
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 XML file, we only need to randomly put it by employing "object".

Examples:

```
mol1 = molgen.Object("mol1.xml", 65, molgen.Object.Shape.none)
gen=molgen.Generators(30,30,30)
gen.addMolecule(mol1,20)
gen.outPutXml("test")
```

13.2 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 XML 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.
```

13.3 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*, *Object::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.xml", 65, molgen.Object.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).
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).
```

13.4 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.

outPutXml (*string filename*)
switch the function of outputting XML filename.

outPutMol2 (*string filename*)
switch the function of outputting Mol2 files.

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.
gen.setMinimumDistance("A","A", 0.7)
# sets the minimum separated distance between the particle types.
gen.outPutXml("test")
# sets the name of output XML file.
```

CHAPTER 14

GalaTackle

14.1 Usage

The "galaTackle" is a plugin of GALAMOST to analyze some important properties by reading the generated configuration files. The "galaTackle" can be found in the installed "bin" folder. You can use this tool to analyze one or more files by running the following commands.

Examples:

```
galaTackle particle.xml
galaTackle particle0.xml particle1.xml
galaTackle *.xml
```

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 "galaTackle" also can be called and passed parameters in Shell script.

Such as:

```
echo -e "18:gpu=1|rmax=3.0" | galaTackle particles.*.xml
```

The "galaTackle" plugin supports the configuration files with XML and DCD formats. The XML files can be tackled independently. However, a DCD trajectory file has to be tackled along with a XML file for particles attributes and topological information.

Examples:

```
galaTackle particle.xml trajectory.dcd
```

For the help about the parameters, you could input "function number:h" after the function list shown.

Examples:

```
14:h
```

14.2 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 XML conversion
-------------------------------------------------------------
```

14.2.1 1 Rg^2:

Description:

To calculate the mean square of gyration radius and output result to `rg2.log`.

$$
\begin{aligned}
R_g^2 &= \frac{1}{N}\sum_{i=1}^{N}\left(\vec{R}_i - \vec{R}_{cm}\right)^2 \\
\vec{R}_{cm} &= \frac{1}{N}\sum_{i=1}^{N}\vec{R}_i
\end{aligned}
$$

Coefficients:

- $\vec{R}_i$ - monomer position vector

14.2.2 2 Ed^2:

Description:

To calculate the mean square of end to end distance and output result to `ed2.log`.

$$E_d^2 = \left(\vec{R}_0 - \vec{R}_{N-1}\right)^2$$

Coefficients:

- $\vec{R}_i$ - monomer position vector

14.2.3 3 RDF:

Description: To calculate the radial distribution function of all particles and output result to `*.rdf` and `rdf.log`.

Parameters: :maxbin=100|gpu=0|rmax=Lx/2|bondex=false|angleex=false|molex=false

14.2.4 4 bond_distri:

Description:

To calculate the distribution of bond lengths and output result 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

14.2.5 5 angle_distri:

Description:

To calculate the distribution of angle degrees and output result 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

14.2.6 6 dihedral_distri:

Description:

To calculate the distribution of dihedral degrees and output result 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

14.2.7 7 stress tensor:

Description: To calculate the stress tensor by inputing the parameters of force calculation and output result to `stress_tensor.log`.

Parameters: :bondex=true|bodyex=true|diameter=true

14.2.8 8 density:

Description: To calculate the real density (g/cm^3) with basic units [amu] and [nm] and output result to `density.log`.

14.2.9 9 unwrapping:

Description: To unwrap or shift molecules by changing the image information

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|convert_constraint_to_bond=false| remove_bond_cross_box=false

14.2.10 10 MSD:

Description: To compute mean square displacement of all particles and output result to `msd.log`.

Parameters: :direction=XYZ (candidates are X,Y,Z,XY,YZ,XZ,XYZ)

14.2.11 11 RDF-CM:

Description: To calculate the radial distribution function of the mass center of molecules and output result to `rdf_cm.log`.

Parameters: :maxbin=100|gpu=0|rmax=Lx/2

14.2.12 12 MSD-CM:

Description: To compute mean square displacement of the mass center of molecules and output result to `msd_cm.log`.

Parameters: :direction=XYZ (candidates are X,Y,Z,XY,YZ,XZ,XYZ)

14.2.13 13 ents:

Description: To analyze the entanglements of polymers and output result to `ents.log`.

14.2.14 14 strfac:

Description: To calculate the structure factor of particles and output result to `*.strf` and `strf.log`.

Parameters: :qmax=160pi/Lmin|gpu=0|deltaq=2pi/Lmin|direction=XYZ|2D=false

14.2.15 15 domain size:

Description: To calculate the domain size of components in mixtures and output result to `domsize.log`.

Parameters: :kmax=20|qc=0.4690|gpu=0

14.2.16 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.

The results will be output to `dstrf.log`.

Parameters: :kmax=int(L)|q=7.0

Maintainer: Shu-Jia Li

14.2.17 17 config check:

Description: To check the configuration including the minimum distance of particles, and the Maximum and minimum length of bonds, etc. and output result to `config_check.log`.

Parameters: :bondex=true|angleex=true|dihedralex=true|bodyex=true|rcut=2.0

14.2.18 18 RDF between types:

Description: To compute the radial distribution function between types and output result to `*.type.rdf` and `rdf_by_type.log`.

Parameters: :maxbin=100|gpu=0|rmax=Lx/2|bondex=false|angleex=false|molex=false

14.2.19 19 XML conversion:

Description: To convert XML files into other formats

Parameters: :lammps=false|gromacs=false

CHAPTER 15

License

GALAMOST - GPU-Accelerated Large-Scale Molecular Simulation Toolkit

COPYRIGHT

LICENSE

DISCLAIMER

If any results obtained with GALAMOST 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, H. Liu, Z.-W. Li, H.-J. Qian, G. Milano, and Z.-Y. Lu, J. Comput. Chem. 2013, 34, 2197-2211" in their article.

CORRESPONDENCE

Chinese Academy of Sciences, Changchun Institute of Applied Chemistry, State Key Laboratory of Polymer Physics and Chemistry, Changchun 130022, China; Dr. You-Liang Zhu; Email: youliangzhu@ciac.ac.cn.

CHAPTER 16

Original sources

Some methods of GALAMOST are developed partically based on the codes from other packages. To acknowledge the original works, we list the methods of GALAMOST and their original sources.

Methods	Sources
ENUF electrostatics	MDynaMix (CPU codes, GNU License)
Tabulated potential method	IBIsCO (CPU codes, GNU License)
MD-SCF	OCCAM (CPU codes, GNU License)
Gay-Berne model	LAMMPS (CPU codes, GNU License)
Virtual interaction sites	GROMACS (CPU codes, GNU License)
Bond constraint LINCS	GROMACS (CPU codes, GNU License)
Rigid body	LAMMPS (CPU codes, GNU License) and HOOMD (GPU codes, BSD-3 License)
PPPM electrostatics	HOOMD (GPU codes, BSD-3 License)
MTK barostat	HOOMD (GPU codes, BSD-3 License)
Neighbor List on auto parameter (>=4.0.1)	HOOMD (GPU codes, BSD-3 License)

The licenses of source packages that are inluded in GALAMOST codes are listed here for clarity.

HOOMD-blue:

```
BSD 3-Clause License for HOOMD-blue
Copyright (c) 2009-2019 The Regents of the University of Michigan All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are met:


1. Redistributions of source code must retain the above copyright notice,
   this list of conditions and the following disclaimer.
```

(continues on next page)

(continued from previous page)

```
 2. Redistributions in binary form must reproduce the above copyright notice,
    this list of conditions and the following disclaimer in the documentation
    and/or other materials provided with the distribution.


 3. Neither the name of the copyright holder nor the names of its contributors
    may be used to endorse or promote products derived from this software without
    specific prior written permission.


THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

LAMMPS:

```
LAMMPS - Large-scale Atomic/Molecular Massively Parallel Simulator

http://lammps.sandia.gov, Sandia National Laboratories

Steve Plimpton, sjplimp@sandia.gov



Copyright (2003) Sandia Corporation.  Under the terms of Contract

DE-AC04-94AL85000 with Sandia Corporation, the U.S. Government retains

certain rights in this software.  This software is distributed under

the GNU General Public License.
```

MDynaMix:

```
See the README file in the top-level LAMMPS directory.


              Copyright (c) 1996, 1999, 2006 by:
          Alexander Lyubartsev and Aatto Laaksonen,
     Department of Physical Chemistry, Stockholm University

                    All rights reserved
```

GROMACS:

```
GROMACS is free software, distributed under the GNU Lesser General
Public License (LGPL) Version 2.1 or (at your option) any later
version. See section 1 for details. GROMACS includes optional code
covered by several different licences as described below.  The
GROMACS package in its entirety may be copied, modified or
```

(continues on next page)

(continued from previous page)

```
distributed according to the conditions described in section 1.
However, in the interests of clarity and completeness, some
individual parts of GROMACS that can be used under their respective
licenses are also noted here.
```

CHAPTER 17

Indices and tables

- genindex
- modindex
- search

Index

A

AllInfo (*built-in class*), 21
AllInfo.addAngleType() (*built-in function*), 22
AllInfo.addAngleTypeByPairs() (*built-in function*), 22
AllInfo.addBondType() (*built-in function*), 21
AllInfo.addBondTypeByPairs() (*built-in function*), 22
AllInfo.addParticleType() (*built-in function*), 21
AllInfo.setNDimensions() (*built-in function*), 21
AndersenNvt (*built-in class*), 77
AndersenNvt.setT() (*built-in function*), 77
AngleForceCos (*built-in class*), 39
AngleForceCos.setParams() (*built-in function*), 39
AngleForceHarmonic (*built-in class*), 38, 52
AngleForceHarmonic.setParams() (*built-in function*), 38
AngleForceHarmonicCos (*built-in class*), 39, 52
AngleForceHarmonicCos.setParams() (*built-in function*), 39
AngleForceTable (*built-in class*), 44
AngleForceTable.setParams() (*built-in function*), 44
AngleForceTable.setPotential() (*built-in function*), 44
AniForce (*built-in class*), 69
AniForce.setParams() (*built-in function*), 69
AniForce.setPatches() (*built-in function*), 69
Application (*built-in class*), 85
Application.add() (*built-in function*), 85
Application.clear() (*built-in function*), 85
Application.remove() (*built-in function*), 85
Application.run() (*built-in function*), 85
Application.setDt() (*built-in function*), 85
AxialStretching (*built-in class*), 56
AxialStretching.setBoxLength() (*built-in function*), 56

B

BdNvt (*built-in class*), 78
BdNvt.setGamma() (*built-in function*), 78
BdNvt.setT() (*built-in function*), 78
BdNvtRigid (*built-in class*), 79
BdNvtRigid.setGamma() (*built-in function*), 79
BdNvtRigid.setT() (*built-in function*), 79
BerendsenAniNvt (*built-in class*), 68
BerendsenAniNvt.setT() (*built-in function*), 69
BerendsenAniNvt.setTau() (*built-in function*), 69
BerendsenNvt (*built-in class*), 77
BerendsenNvt.setT() (*built-in function*), 77
BinaryDump (*built-in class*), 19
BinaryDump.enableCompression() (*built-in function*), 20
BinaryDump.setOutput() (*built-in function*), 19
BinaryDump.setOutputAll() (*built-in function*), 19
BinaryDump.setOutputForRestart() (*built-in function*), 20
BinaryReader (*built-in class*), 15
BondConstraint (*built-in class*), 53, 60
BondConstraint.setExpansionOrder() (*built-in function*), 53, 60
BondConstraint.setNumIters() (*built-in function*), 53, 60
BondConstraint.setParams() (*built-in function*), 60
BondForceAni (*built-in class*), 67
BondForceAni.setParams() (*built-in function*), 67
BondForceFene (*built-in class*), 36
BondForceFene.setConsiderDiameter() (*built-in function*), 36
BondForceFene.setParams() (*built-in function*), 36
BondForceHarmonic (*built-in class*), 35, 52

BondForceHarmonic.setParams() (built-in function), 35
BondForceMorse (built-in class), 37
BondForceMorse.setParams() (built-in function), 37
BondForcePolynomial (built-in class), 36
BondForcePolynomial.setParams() (built-in function), 36
BondForceTable (built-in class), 43
BondForceTable.setParams() (built-in function), 44
BondForceTable.setPotential() (built-in function), 44
BounceBackConstrain (built-in class), 57
BounceBackConstrain.addCylinder() (built-in function), 58
BounceBackConstrain.addSphere() (built-in function), 58
BounceBackConstrain.addWall() (built-in function), 57

C

CCPMD (built-in class), 59
CCPMD.setParams() (built-in function), 60
CCPMD.setWall() (built-in function), 60
CellList (built-in class), 23
CellList.setDataReproducibility() (built-in function), 23
CellList.setNominalDim() (built-in function), 23
CellList.setNominalWidth() (built-in function), 23
CenterForce (built-in class), 29
CenterForce.setPreNextShift() (built-in function), 30
ComputeInfo (built-in class), 24
ComputeInfo.setNdof() (built-in function), 24

D

DcdDump (built-in class), 19
DcdDump.unpbc() (built-in function), 19
DcdDump.unwrap() (built-in function), 19
DePolymerization (built-in class), 66
DePolymerization.setParams() (built-in function), 66
DePolymerization.setT() (built-in function), 66
DihedralForceAmberCosine (built-in class), 52
DihedralForceHarmonic (built-in class), 40, 41, 53
DihedralForceHarmonic.setCosFactor() (built-in function), 40
DihedralForceHarmonic.setParams() (built-in function), 40, 41
DihedralForceOplsCosine (built-in class), 42
DihedralForceOplsCosine.setParams() (built-in function), 42
DihedralForceTable (built-in class), 44
DihedralForceTable.setParams() (built-in function), 44
DihedralForceTable.setPotential() (built-in function), 44, 45
DPDEwaldForce (built-in class), 47
DPDEwaldForce.setBeta() (built-in function), 47
DPDEwaldForce.setParams() (built-in function), 47
DpdForce (built-in class), 71
DpdForce.setDPDVV() (built-in function), 71
DpdForce.setParams() (built-in function), 71
DpdForce.setT() (built-in function), 71
DpdGwvv (built-in class), 72
DpdGwvv.setLambda() (built-in function), 72
DumpInfo (built-in class), 15
DumpInfo.dumpAnisotropy() (built-in function), 15
DumpInfo.dumpBoxSize() (built-in function), 16
DumpInfo.dumpParticleForce() (built-in function), 16
DumpInfo.dumpParticlePosition() (built-in function), 16
DumpInfo.dumpPotential() (built-in function), 15
DumpInfo.dumpPressTensor() (built-in function), 16
DumpInfo.dumpTypeTemp() (built-in function), 16
DumpInfo.dumpVirial() (built-in function), 15
DumpInfo.dumpVirialMatrix() (built-in function), 16
DumpInfo.setPeriod() (built-in function), 16

E

ENUFForce (built-in class), 48
ENUFForce.setParams() (built-in function), 48
EwaldForce (built-in class), 46
EwaldForce.setParams() (built-in function), 46
ExternalForce (built-in class), 57
ExternalForce.setForce() (built-in function), 57
ExternalForce.setParams() (built-in function), 57

G

GBForce (built-in class), 66
GBForce.setParams() (built-in function), 66
GEMForce (built-in class), 30
GEMForce.setParams() (built-in function), 30
Generators (built-in class), 93
Generators.addMolecule() (built-in function), 94

Generators.outPutMol2() (*built-in function*), 94
Generators.outPutXml() (*built-in function*), 94
Generators.setDimension() (*built-in function*), 94
Generators.setMinimumDistance() (*built-in function*), 94
Generators.setParam() (*built-in function*), 94

L

LjConstrainForce (*built-in class*), 58
LjConstrainForce.addCylinder() (*built-in function*), 58
LjConstrainForce.addSphere() (*built-in function*), 58
LjConstrainForce.addWall() (*built-in function*), 58
LjConstrainForce.setParams() (*built-in function*), 58
LJCoulombShiftForce (*built-in class*), 51
LJEwaldForce (*built-in class*), 31, 51
LJEwaldForce.setDispVirialCorr() (*built-in function*), 31, 51
LJEwaldForce.setEnergy_shift() (*built-in function*), 31, 51
LJEwaldForce.setParams() (*built-in function*), 31
LjForce (*built-in class*), 28
LjForce.setDispVirialCorr() (*built-in function*), 28
LjForce.setEnergy_shift() (*built-in function*), 28
LjForce.setParams() (*built-in function*), 28
LzwForce (*built-in class*), 68
LzwForce.setMethod() (*built-in function*), 68
LzwForce.setParams() (*built-in function*), 68

M

MdScfForce (*built-in class*), 63
MdScfForce.setNewVersion() (*built-in function*), 63
MdScfForce.setParams() (*built-in function*), 63
MdScfForce.setPeriodScf() (*built-in function*), 63
Mol2Dump (*built-in class*), 16
Mol2Dump.deleteBoundaryBond() (*built-in function*), 16
Mol2Dump.setChangeFreeType() (*built-in function*), 16
Molecule (*built-in class*), 90
Molecule.setAngleDegree() (*built-in function*), 90, 91
Molecule.setBody() (*built-in function*), 92
Molecule.setBodyEvacuation() (*built-in function*), 92
Molecule.setBondLength() (*built-in function*), 90
Molecule.setBox() (*built-in function*), 92
Molecule.setCharge() (*built-in function*), 91
Molecule.setCris() (*built-in function*), 92
Molecule.setCylinder() (*built-in function*), 92
Molecule.setDiameter() (*built-in function*), 91
Molecule.setDihedralDegree() (*built-in function*), 91
Molecule.setInert() (*built-in function*), 91
Molecule.setInit() (*built-in function*), 92
Molecule.setIsotactic() (*built-in function*), 90
Molecule.setMass() (*built-in function*), 91
Molecule.setMolecule() (*built-in function*), 92
Molecule.setOrientation() (*built-in function*), 91
Molecule.setParticleTypes() (*built-in function*), 90
Molecule.setQuaternion() (*built-in function*), 91
Molecule.setSphere() (*built-in function*), 92
Molecule.setTopology() (*built-in function*), 90

N

NeighborList (*built-in class*), 22
NeighborList.addExclusionsFromAngles() (*built-in function*), 22
NeighborList.addExclusionsFromBodys() (*built-in function*), 22
NeighborList.addExclusionsFromBonds() (*built-in function*), 22
NeighborList.addExclusionsFromDihedrals() (*built-in function*), 22
NeighborList.setDataReproducibility() (*built-in function*), 22
NeighborList.setFilterDiameters() (*built-in function*), 22
NeighborList.setNsq() (*built-in function*), 22
NeighborList.setRCut() (*built-in function*), 22
NeighborList.setRCutPair() (*built-in function*), 22
NoseHooverNvt (*built-in class*), 76
NoseHooverNvt.setT() (*built-in function*), 77
Npt (*built-in class*), 80
Npt.setP() (*built-in function*), 80
Npt.setT() (*built-in function*), 80
NPTMTK (*built-in class*), 81
NPTMTK.setAnisotropic() (*built-in function*), 82
NPTMTK.setSemiisotropic() (*built-in function*), 82
NPTMTK.setT() (*built-in function*), 82
NPTMTKRigid (*built-in class*), 82
NPTMTKRigid.setAnisotropic() (*built-in function*), 83

NPTMTKRigid.setSemiisotropic() (*built-in function*), 82, 83
NPTMTKRigid.setT() (*built-in function*), 82
NptRigid (*built-in class*), 80
NptRigid.setP() (*built-in function*), 81
NptRigid.setT() (*built-in function*), 81
Nve (*built-in class*), 75
Nve.setZeroForce() (*built-in function*), 75
NveRigid (*built-in class*), 75
NvtRigid (*built-in class*), 79
NvtRigid.setT() (*built-in function*), 79

O

Object (*built-in class*), 93
Object.setRadius() (*built-in function*), 93

P

PairForce (*built-in class*), 34
PairForce.setParams() (*built-in function*), 34
PairForce.setShiftParams() (*built-in function*), 34
PairForceTable (*built-in class*), 43
PairForceTable.setParams() (*built-in function*), 43
PairForceTable.setPotential() (*built-in function*), 43
ParticleSet (*built-in class*), 23
PBGBForce (*built-in class*), 67
PBGBForce.setAspheres() (*built-in function*), 67
PBGBForce.setGUM() (*built-in function*), 67
PBGBForce.setParams() (*built-in function*), 67
PBGBForce.setPatches() (*built-in function*), 67
PerformConfig (*built-in class*), 21
PFMEForce (*built-in class*), 64
PFMEForce.setNewVersion() (*built-in function*), 64
PFMEForce.setPeriodPFME() (*built-in function*), 64
Polymerization (*built-in class*), 64, 65
Polymerization.initExPoint() (*built-in function*), 65
Polymerization.setExchangePr() (*built-in function*), 65
Polymerization.setFuncReactRule() (*built-in function*), 65
Polymerization.setMaxCris() (*built-in function*), 65
Polymerization.setMinDisReactRule() (*built-in function*), 65
Polymerization.setNewBondType() (*built-in function*), 65
Polymerization.setPr() (*built-in function*), 65
Polymerization.setPrFactor() (*built-in function*), 65
PPPMForce (*built-in class*), 48
PPPMForce.setParams() (*built-in function*), 48

R

Reader (*built-in class*), 15

S

SljForce (*built-in class*), 28
SljForce.setEnergy_shift() (*built-in function*), 29
SljForce.setParams() (*built-in function*), 29
Sort (*built-in class*), 24

T

TranRigid (*built-in class*), 76
TranRigid.setRotDimension() (*built-in function*), 76
TranRigid.setTraDimension() (*built-in function*), 76

V

VariantConst (*built-in class*), 55
VariantLinear (*built-in class*), 55
VariantLinear.setPoint() (*built-in function*), 55
VariantSin (*built-in class*), 56
VariantSin.setPoint() (*built-in function*), 56
VariantWell (*built-in class*), 56
VariantWell.setPoint() (*built-in function*), 56
Vsite (*built-in class*), 53, 60
Vsite.setParams() (*built-in function*), 61

X

XmlDump (*built-in class*), 17
XmlDump.setOutput() (*built-in function*), 17
XmlDump.setOutputAngle() (*built-in function*), 18
XmlDump.setOutputBody() (*built-in function*), 17
XmlDump.setOutputBond() (*built-in function*), 18
XmlDump.setOutputCharge() (*built-in function*), 17
XmlDump.setOutputConstraint() (*built-in function*), 18
XmlDump.setOutputCris() (*built-in function*), 18
XmlDump.setOutputDiameter() (*built-in function*), 17
XmlDump.setOutputDihedral() (*built-in function*), 18
XmlDump.setOutputEllipsoid() (*built-in function*), 18
XmlDump.setOutputForce() (*built-in function*), 17
XmlDump.setOutputImage() (*built-in function*), 17

XmlDump.setOutputInert() (*built-in function*), 18
XmlDump.setOutputInit() (*built-in function*), 18
XmlDump.setOutputLocalForce() (*built-in function*), 18
XmlDump.setOutputLocalVirial() (*built-in function*), 18
XmlDump.setOutputLocalVirialMatrix() (*built-in function*), 18
XmlDump.setOutputMass() (*built-in function*), 17
XmlDump.setOutputOrientation() (*built-in function*), 17
XmlDump.setOutputPatch() (*built-in function*), 18
XmlDump.setOutputPosition() (*built-in function*), 17
XmlDump.setOutputQuaternion() (*built-in function*), 18
XmlDump.setOutputRotation() (*built-in function*), 18
XmlDump.setOutputTorque() (*built-in function*), 18
XmlDump.setOutputType() (*built-in function*), 17
XmlDump.setOutputVelocity() (*built-in function*), 17
XmlDump.setOutputVirial() (*built-in function*), 17
XmlDump.setOutputVsite() (*built-in function*), 18
XmlReader (*built-in class*), 15

Z

ZeroMomentum (*built-in class*), 58