ESPResSo++ Documentation

Release latest

Developer team

Mar 03, 2024

CONTENTS

1 Installation **3**

2 Tutorial **5**
- 2.1 Basic System Setup . . . 5
- 2.2 Simple Lennard Jones System . . . 8
- 2.3 Advanced Lennard Jones System . . . 10
- 2.4 Polymer Melt . . . 12
- 2.5 AddNewPotential . . . 14
- 2.6 Appendices . . . 18
- 2.7 Adaptive Resolution Simulations . . . 19
- 2.8 Thermodynamic integration . . . 28

3 User Interface **33**
- 3.1 analysis . . . 33
- 3.2 bc . . . 55
- 3.3 check . . . 57
- 3.4 esutil . . . 57
- 3.5 external . . . 58
- 3.6 integrator . . . 74
- 3.7 interaction . . . 110
- 3.8 io . . . 204
- 3.9 espressopp . . . 210
- 3.10 standard_system . . . 236
- 3.11 storage . . . 238
- 3.12 tools . . . 242
- 3.13 Logging mechanism . . . 255

4 Credits **259**
- 4.1 ESPResSo++ Developers . . . 259
- 4.2 FAQ . . . 259
- 4.3 Getting Help . . . 259

Bibliography **261**

Python Module Index **263**

Index **267**

Welcome to the homepage of the ESPResSo++ project

ESPResSo++ is an extensible, flexible, fast and parallel simulation software for soft matter research. It is a highly versatile software package for the scientific simulation and analysis of coarse-grained atomistic or bead-spring models as they are used in soft matter research.

ESPResSo and ESPResSo++ have common roots and share parts of the developer/user community. However their development is independent and they are different software packages.

ESPResSo++ is free, open-source software published under the GNU General Public License (GPL).

Please cite this, if you used ESPResSo++ in your research

H. V. Guzman, N. Tretyakov, H. Kobayashi, A. C. Fogarty, K. Kreis, J. Krajniak, C. Junghans, K. Kremer, T. Stuehn, "ESPResSo++ 2.0: Advanced methods for multiscale molecular simulation", Computer Physics Communications, 238 (2019), pp. 66-76 DOI: 10.1016/j.cpc.2018.12.017 Online access: https://doi.org/10.1016/j.cpc.2018.12.017

J. D. Halverson, T. Brandes, O. Lenz, A. Arnold, S. Bevc, V. Starchenko, K. Kremer, T. Stuehn, D. Reith, "ESPResSo++: A Modern Multiscale Simulation Package for Soft Matter Systems", Computer Physics Communications, 184 (2013), pp. 1129-1149 DOI: 10.1016/j.cpc.2012.12.004 Online access: http://dx.doi.org/10.1016/j.cpc.2012.12.004

Recent publications where ESPResSo++ was used

CHAPTER

ONE

INSTALLATION

The first step in the installation of ESPResSo++ is to download the latest release from the following location:

https://github.com/espressopp/espressopp/releases

On the command line type:

```
tar -xzf espressopp-latest.tgz
```

This will create a subdirectory espressopp-latest

Enter this subdirectory

```
cd espressopp-latest
```

Create the Makefiles using the cmake command. If you don't have it yet, you have to install it first. It is available for all major Linux distributions and also for Mac OS X. (ubuntu,debian: "apt-get install cmake" or get it from http://www.cmake.org)

```
cmake -B builddir .
```

(the space and dot after *cmake* are necessary)

If cmake doesn't finish successfully (e.g. it didn't find all the libraries) you can tell cmake manually, where to find them by typing:

```
ccmake -B builddir .
```

This will open an interactive page where all configuration information can be specified.

After successfully building all the Makefiles you should build ESPResSo++ with:

```
cmake --build builddir
```

(This will take several minutes)

After successfully building ESPResSo++ add the src directory to your PYTHONPATH:

```
export PYTHONPATH=$PWD/src:${PYTHONPATH}
```

In order to use matplotlib.pyplot for graphical output get the open source code from:

http://sourceforge.net/projects/matplotlib

and follow the installation instructions of your distribution.

CHAPTER

TWO

TUTORIAL

2.1 Basic System Setup

ESPResSo++ is implemented as a python module that has to be imported at the beginning of every script:

```
>>> import espressopp
```

ESPResSo++ uses an object called *System* to store some global variables and is also used to keep the connection between some other important modules. We create it with:

```
>>> system        = espressopp.System()
```

Starting a new simulation with ESPResSo++ we should have an idea about what we want to simulate. E.g. how big should the simulation box be or what is the density of the system or what are the interactions and the interaction ranges between our particles.

Let us start with the size of the simulation box:

```
>>> box           = (10, 10, 10)
```

In many cases you will need a random number generator (e.G. to couple to a temperature bath or to randomly position particles in the simulation box). ESPResSo++ provides its own random number generator (for the experts: see boost/random.hpp) so let's use it:

```
>>> rng           = espressopp.esutil.RNG()
```

Our simulation box needs some boundary conditions. We want to use periodic boundary conditions:

```
>>> bc            = espressopp.bc.OrthorhombicBC(rng, box)
```

We tell our system object about this:

```
>>> system.bc     = bc
>>> system.rng    = rng
```

Now we need to decide which parallelization scheme for the particle storage we want to use. In the current version of ESPResSo++ there is only one storage scheme implemented which is *domain decomposition*. Further parallelized storages (e.g. *atom decomposition* or *force decomposition*) will be implemented in future versions.

The *domain decomposition* storage needs to know how many CPUs (or cores, if there are multicore CPUs) are available for the simulation and how to assign the CPUs to the different domains of our simulation box. Moreover the storage needs to know the maximum interaction range of the particles. In a simple Lennard-Jones fluid this could for example be $r_{cut} = 2^{\frac{1}{6}}$. This value together with the *skin* value determines the minimal size for the so called *linked cells* which are used to speed up Verlet list rebuilds (see Frenkel&Smit or Allen&Tildesley for the details).

```
>>> maxcutoff      = pow(2.0, 1.0/6.0)
>>> skin           = 0.4
```

Tell the system about it:

```
>>> system.skin    = skin
```

In the most simple case, if you want to use only one CPU, the *nodeGrid* and the *cellGrid* could look like this:

```
>>> nodeGrid      = (1,1,1)
>>> cellGrid      = (2,2,2)
```

In general you don't need to take care of that yourself. Just use the corresponding ESPResSo++ routines to calculate a reasonable *nodeGrid* and *cellGrid*:

```
>>> nodeGrid      = espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,
↪maxcutoff,skin)
>>> cellGrid      = espressopp.tools.decomp.cellGrid(box, nodeGrid, maxcutoff, skin)
```

Now we have all the ingredients we need for the *domain decomposition* storage of our system:

```
>>> ddstorage     = espressopp.storage.DomainDecomposition(system, nodeGrid, cellGrid)
```

We initialized the DomainDecomposition object with a pointer to our system. We also have to inform the system about the DomainDecomposition storage:

```
>>> system.storage = ddstorage
```

The next module we need is the *integrator*. This object will do the actual work of integrating Newtons equations of motion. ESPResSo++ implements the well known *velocity Verlet* algorithm (see for example Frenkel&Smit):

```
>>>  integrator    = espressopp.integrator.VelocityVerlet(system)
```

We have to tell the integrator about the basic time step:

```
>>>  dt            = 0.005
>>>  integrator.dt = dt
```

Let's do some math in between:

Note: For 3D vectors like positions, velocities or forces ESPResSo++ provides a so called *Real3D* type, which simplifies handling and arithmetic operations with vectors. 3D coordinates would typically be defined like this:

```
>>> a = espressopp.Real3D(2.0, 5.0, 6.0)
>>> b = espressopp.Real3D(0.1, 0.0, 0.5)
```

Now you could do things like:

```
>>> c = a + b       # c is a Real3D object
>>> d = a * 1.5     # d is a Real3D object
>>> e = a - b       # e is a Real3D object
>>> f = e.sqr()     # f is a scalar
>>> g = e.abs()     # g is a scalar
```

In order to make defining vectors even more simple include the line

```
>>> from espressopp import Real3D
```

just at the beginning of your script. This allows to define vectors as:

```
>>> vec = Real3D(2.0, 1.5, 5.0)
```

Back to our simulation:

The most simple simulation we can do is integrating Newtons equation of motion for one particle without any external forces. So let's simply add one particle to the storage of our system. Every particle in ESPResSo++ has a unique particle id and a position (this is obligatory).

```
>>> pid = 1
>>> pos = Real3D(2.0, 4.0, 6.0)    # remember to add "from espressopp import Real3D"
>>>                                 # at the beginning of your script
>>> system.storage.addParticle(pid, pos)
```

Of course nothing will happen when we integrate this. The particle will stay where it is. Add some initial velocity to the particle by adding the follow line to the script:

```
>>> system.storage.modifyParticle(pid, 'v', Real3D(1.0, 0, 0))
```

After particles have been modified make sure that this information is distributed to all CPUs:

```
>>> system.storage.decompose()
```

Now we can propagate the particle by calling the integrator:

```
>>> integrator.run(100)
```

Check the result with:

```
>>> print "The new particle position is: ", system.storage.getParticle(pid).pos
```

Let's add some more particles at random positions with random velocities and random mass and random type 0 or 1. The boundary condition object knows about how to create random positions within the simulation box. We can add all the particles at once by creating a particle list first:

```
>>> particle_list = []
>>> num_particles = 9
>>> for k in range(num_particles):
>>>   pid  = 2 + k
>>>   pos  = system.bc.getRandomPos()
>>>   v    = Real3D(system.rng(), system.rng(), system.rng())
>>>   mass = system.rng()
>>>   type = system.rng(2)
>>>   part = [pid, pos, type, v, mass]
>>>   particle_list.append(part)
>>> system.storage.addParticles(particle_list, 'id', 'pos', 'type', 'v', 'mass')
>>> # don't forget the decomposition
>>> system.storage.decompose()
```

To have a look at the overall system there are several possibilities. The easiest way to get a nice picture is by writing out a PDB file and looking at the configuration with some visualization programm (e.g. VMD):

```
>>> filename = "myconf.pdb"
>>> espressopp.tools.pdb.pdbwrite(filename, system)
```

or (if *vmd* is in your search PATH) you could directly connect to VMD by:

```
>>> espressopp.tools.vmd.connect(system)
```

or you could print all particle information to the screen:

```
>>> for k in range(10):
>>>   p = system.storage.getParticle(k+1)
>>>   print p.id, p.type, p.mass, p.pos, p.v, p.f, p.q
```

2.2 Simple Lennard Jones System

Lets just copy and paste the beginning from the "System Setup" tutorial:

```
>>> import espressopp
>>> from espressopp import Real3D
>>>
>>> system        = espressopp.System()
>>> box           = (10, 10, 10)
>>> rng           = espressopp.esutil.RNG()
>>> bc            = espressopp.bc.OrthorhombicBC(rng, box)
>>> system.bc     = bc
>>> system.rng    = rng
>>> maxcutoff     = pow(2.0, 1.0/6.0)
>>> skin          = 0.4
>>> system.skin   = skin
>>> nodeGrid      = (1,1,1)
>>> cellGrid      = (1,1,1)
>>> nodeGrid      = espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,
↪maxcutoff,skin)
>>> cellGrid      = espressopp.tools.decomp.cellGrid(box, nodeGrid, maxcutoff, skin)
>>> ddstorage     = espressopp.storage.DomainDecomposition(system, nodeGrid, cellGrid)
>>> system.storage = ddstorage
>>>
>>> integrator    = espressopp.integrator.VelocityVerlet(system)
>>> dt            = 0.005
>>> integrator.dt  = dt
```

And lets add some random particles:

```
>>> num_particles = 20
>>> particle_list = []
>>> for k in range(num_particles):
>>>   pid  = k + 1
>>>   pos  = system.bc.getRandomPos()
>>>   v    = Real3D(0,0,0)
>>>   mass = system.rng()
>>>   type = 0
```

(continues on next page)

(continued from previous page)

```
>>>   part = [pid, pos, type, v, mass]
>>>   particle_list.append(part)
>>> system.storage.addParticles(particle_list, 'id', 'pos', 'type', 'v', 'mass')
>>> system.storage.decompose()
```

All particles should interact via a Lennard Jones potential:

```
>>> LJPot = espressopp.interaction.LennardJones(epsilon=1.0, sigma=1.0, cutoff=maxcutoff, shift=
↪'auto')
```

shift=True means that the potential will be shifted at the cutoff so that potLJ(cutoff)=0 Next we create a VerletList which will than be used in the interaction: (the Verlet List object needs to know from which system to get its particles and which cutoff to use)

```
>>> verletlist = espressopp.VerletList(system, cutoff=maxcutoff)
```

Now create a non bonded interaction object and add the Lennard Jones potential to that:

```
>>> NonBondedInteraction = espressopp.interaction.VerletListLennardJones(verletlist)
>>> NonBondedInteraction.setPotential(type1=0, type2=0, potential=LJPot)
```

Tell the system about the newly created NonBondedInteraction object:

```
>>> system.addInteraction(NonBondedInteraction)
```

We should set the langevin thermostat in the integrator to cool down the random particle system:

```
>>> langevin             = espressopp.integrator.LangevinThermostat(system)
>>> langevin.gamma       = 1.0
>>> langevin.temperature = 1.0
>>> integrator.addExtension(langevin)
```

and finally let the system run and see how it relaxes or explodes:

```
>>> espressopp.tools.analyse.info(system, integrator)
>>> for k in range(100):
>>>   integrator.run(10)
>>>   espressopp.tools.analyse.info(system, integrator)
```

Due to the random particle positions it may happen, that two or more particles are very close to each other and the resulting repulsive force between them are so high that they 'shoot off" in different directions with very high speed. Usually the numbers are then larger than the computer can deal with. A typical error message you get could look like this:

Note: ERROR: particle 5 has moved to outer space (one or more coordinates are nan)

In order to prevent this, systems that are setup in a random way and thus have strong overlaps between particels have to be "warmed up" before they can be equilibrated.

In ESPResSo++ there are several possible ways of warming up a system. As a first approach one could simply constrain the forces in the integrator. For this purpose ESPResSo++ provides an integrator Extension named CapForces. The two parameters of this Extension are the system and the maximum force that a particle can get. The following python code shows how CapForces can be used. Add it to your Lennard-Jones example just after adding the Langevin Extension:

```
>>> print "starting warmup with force capping ..."
>>> force_capping  = espressopp.integrator.CapForce(system, 1000000.0)
>>> integrator.addExtension(force_capping)
>>> # reduce the time step of the integrator to make the integration numerically more stable
>>> integrator.dt = 0.0001
>>> espressopp.tools.analyse.info(system, integrator)
>>> for k in range(10):
>>>   integrator.run(1000)
>>>   espressopp.tools.analyse.info(system, integrator)
```

After the warmup the time step of the integrator can be set to a larger value. The CapForce extension can be disconnected after the warmup to get the original full Lennard-Jones potential back.

```
>>> integrator.dt  = 0.005
>>> integrator.step = 0
>>> force_capping.disconnect()
>>> print "warmup finished - force capping switched off."
```

2.2.1 Task 1:

write a python script that creates a random configuration of 1000 Lennard Jones particles with a number density of 0.85 in a cubic simulation box. Warm up and equilibrate this configuration. Examine the output of the command

```
>>> espressopp.tools.analyse.info(system, integrator)
```

after each integration step. How fast is the energy of the system going down ? How long do you have to warmup ? What are good parameters for dt, force_capping and number of integration steps ?

2.3 Advanced Lennard Jones System

This tutorial needs the matplotlib.pyplot and numpy libraries and also VMD to be installed.

```
>>> import espressopp
```

After importing espressopp we import several other Python packages that we want to use for graphical output of some system parameters (e.g. temperature and energy)

```
>>> import math
>>> import time
>>> import matplotlib
>>> matplotlib.use('TkAgg')
>>> import matplotlib.pyplot as plt
>>> plt.ion()
```

We setup a standard Lennard-Jones system with 1000 particles and a density of 0.85 in a cubic siomulation box. ESPResSo++ provides a “shortcut” to setup such a system:

```
>>> N   = 1000
>>> rho = 0.85
>>> L   = pow(N/rho, 1.0/3)
>>> system, integrator = espressopp.standard_system.LennardJones(N,(L, L, L), dt=0.0001)
```

We also add a Langevin thermostat:

```
>>> langevin = espressopp.integrator.LangevinThermostat(system)
>>> langevin.gamma       = 1.0
>>> langevin.temperature = 1.0
>>> integrator.addExtension(langevin)
```

We do a very short warmup in the beginning to get rid of "extremely" high forces

```
>>> force_capping  = espressopp.integrator.CapForce(system, 1000000.0)
>>> integrator.addExtension(force_capping)
>>> espressopp.tools.analyse.info(system, integrator)
>>> for k in range(10):
>>>   integrator.run(100)
>>>   espressopp.tools.analyse.info(system, integrator)
```

Now let's initialize a graph. So that we can have a realtime-view on what is happening in the simulation:

```
>>> plt.figure()
```

We want to observe temperature and energy of the system:

```
>>> T  = espressopp.analysis.Temperature(system)
>>> E  = espressopp.analysis.EnergyPot(system, per_atom=True)
```

x will be the x-axixs of the graph containg the time. yT and yE will be temperature and energy as y-axes in 2 plots:

```
>>> x  = []
>>> yT  = []
>>> yE  = []
>>> yTmin = 0.0
>>> yEmin = 0.0
>>> x.append(integrator.dt * integrator.step)
>>> yT.append(T.compute())
>>> yE.append(E.compute())
>>> yTmax = max(yT)
>>> yEmax = max(yE)
```

Initialize the two graphs ('ro' means red circles, 'go' means green cirlces, see also pyplot documentation)

```
>>> plt.subplot(211)
>>> gT, = plt.plot(x, yT, 'ro')
>>> plt.subplot(212)
>>> gE, = plt.plot(x, yE, 'go')
```

We also want to observe the configuration with VMD. So we have to connect to vmd. This command will automatically start vmd (vmd has to be found in your PATH environment for this to work)

```
>>> sock = espressopp.tools.vmd.connect(system)
>>> for k in range(200):
>>>   integrator.run(1000)
>>>   espressopp.tools.vmd.imd_positions(system, sock)
```

Update the x-, and y-axes:

```
>>>  x.append(integrator.dt * integrator.step)
>>>  yT.append(T.compute())
>>>  yE.append(E.compute())
>>>  yTmax = max(yT)
>>>  yEmax = max(yE)
```

Plot the temperature graph

```
>>>  plt.subplot(211)
>>>  plt.axis([x[0], x[-1], yTmin, yTmax*1.2 ])
>>>  gT.set_ydata(yT)
>>>  gT.set_xdata(x)
>>>  plt.draw()
```

Plot the energy graph

```
>>>  plt.subplot(212)
>>>  plt.axis([x[0], x[-1], yEmin, yEmax*1.2 ])
>>>  gE.set_ydata(yE)
>>>  gE.set_xdata(x)
>>>  plt.draw()
```

In the end save the equilibrated configurations as .eps and .pdf files

```
>>> plt.savefig('mypyplot.eps')
>>> plt.savefig('mypyplot.pdf')
```

2.4 Polymer Melt

We first import espressopp and then define all the parameters of the simulation:

```
>>> import espressopp
>>> num_chains          = 10
>>> monomers_per_chain  = 10
>>> L                   = 10
>>> box                 = (L, L, L)
>>> bondlen             = 0.97
>>> rc                  = pow(2, 1.0/6.0)
>>> skin                = 0.3
>>> dt                  = 0.005
>>> epsilon             = 1.0
>>> sigma               = 1.0
```

Like in the simple Lennard Jones tutorial we setup the system and the integrator. First the system with the excluded volume interaction (WCA, Lennard Jones type)

```
>>> system        = espressopp.System()
>>> system.rng    = espressopp.esutil.RNG()
>>> system.bc     = espressopp.bc.OrthorhombicBC(system.rng, box)
>>> system.skin   = skin
>>> nodeGrid      = espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,rc,
↪skin)
```

(continues on next page)

(continued from previous page)

```
>>> cellGrid      = espressopp.tools.decomp.cellGrid(box, nodeGrid, rc, skin)
>>> system.storage = espressopp.storage.DomainDecomposition(system, nodeGrid, cellGrid)
>>> interaction    = espressopp.interaction.VerletListLennardJones(espressopp.VerletList(system,␣
↪cutoff=rc))
>>> potLJ         = espressopp.interaction.LennardJones(epsilon, sigma, rc)
>>> interaction.setPotential(type1=0, type2=0, potential=potLJ)
>>> system.addInteraction(interaction)
```

Then the integrator with the Langevin extension

```
>>> integrator    = espressopp.integrator.VelocityVerlet(system)
>>> integrator.dt  = dt
>>> thermostat    = espressopp.integrator.LangevinThermostat(system)
>>> thermostat.gamma  = 1.0
>>> thermostat.temperature = temperature
>>> integrator.addExtension(thermostat)
```

Know we add the particles. Keep in mind that we want to create a polymer melt. This means that particles are "bonded" in chains. We setup each polymer chain as a random walk.

```
>>> props    = ['id', 'type', 'mass', 'pos', 'v']
>>> vel_zero = espressopp.Real3D(0.0, 0.0, 0.0)
```

In providing bonding information for the particles we "setup" the bonded chains. For this we use the FixedPairList object that needs to know where and in which storage the particles can be found:

```
>>> bondlist = espressopp.FixedPairList(system.storage)
>>> pid      = 1
>>> type     = 0
>>> mass     = 1.0
>>> chain    = []
```

ESPResSo++ provides a function that will return position and bond information of a random walk. You have to provide a start ID (particle id) and a starting position which we will get from the random position generator of the boundary condition object:

```
>>> for i in range(num_chains):
>>>   startpos = system.bc.getRandomPos()
>>>   positions, bonds = espressopp.tools.topology.polymerRW(pid, startpos, monomers_per_chain,␣
↪bondlen)
>>>   for k in range(monomers_per_chain):
>>>     part = [pid + k, type, mass, positions[k], vel_zero]
>>>     chain.append(part)
>>>   pid += monomers_per_chain
>>>   type += 1
>>>   system.storage.addParticles(chain, *props)
>>>   system.storage.decompose()
>>>   chain = []
>>>   bondlist.addBonds(bonds)
```

Note: try out the command

```
>>> espressopp.tools.topology.polymerRW(pid, startpos, monomers_per_chain, bondlen)
```

to see what it returns

Don't forget to distribute the particles and the bondlist to the CPUs in the end:

```
>>> system.storage.decompose()
```

Finally add the information about the bonding potential. In this example we are using a FENE-potential between the bonded particles.

```
>>> potFENE    = espressopp.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
>>> interFENE = espressopp.interaction.FixedPairListFENE(system, bondlist, potFENE)
>>> system.addInteraction(interFENE)
```

Start the integrator and observe how the system explodes. Like in the random Lennard Jones system, we have the same problem here: particles can strongly overlap and thus will get very high forces accelerating them to infinite (for the computer) speed.

```
>>> espressopp.tools.analyse.info(system, integrator)
>>> for k in range(nsteps):
>>>   integrator.run(isteps)
>>>   espressopp.tools.analyse.info(system, integrator)
>>>   espressopp.tools.analyse.info(system, integrator)
```

2.4.1 Task 2:

Try to warmup and equilibrate a dense polymer melt (density=0.85) by using the warmup methods that you have learned in the Lennard Jones tutorial.

2.4.2 Hint:

During warmup you can slowly switch on the excluded volume interaction by starting with a small epsilon and increasing it during integration: You can do this by continuously overwriting the interaction potential after some time interval.

```
>>> potLJ          = espressopp.interaction.LennardJones(new_epsilon, sigma, rc)
>>> interaction.setPotential(type1=0, type2=0, potential=potLJ)
```

2.5 AddNewPotential

The aim of the tutorial is to implement a new interaction potential in ESPResSo++. We start with the Gromos fourth-power bond-stretching potential, because its functional form is simple and its implementation is somewhat similar to other potentials already implemented in ESPResSo++. Everything you learn in this tutorial will then be relevent for implementing any other more complicated potential.

Make sure you have a working, compiled version of ESPResSo++ before starting the tutorial.

For those who are not so familiar with C++ or interfacing python and C++, you will find some helpful notes in the appendix.

2.5.1 Steps for adding a new interaction potential

1. Choose the potential and derive the force.
2. Choose the appropriate interaction template from those in $ESPRESSOHOME/src/interaction, e.g. VerletListInteractionTemplate.hpp, FixedTripleListInteractionTemplate.hpp
3. Create the .cpp, .hpp and .py files for your potential, place them in $ESPRESSOHOME/src/interaction and modify $ESPRESSOHOME/src/interaction/bindings.cpp and $ESPRESSOHOME/src/interaction/__init__.py
4. Compile.

These steps are described in more detail below for our tutorial example potential.

2.5.2 Today's tutorial exercise

Step 1

The potential we are implementing today is a two-body bonded potential with the form

$$V(r_{ij}) = \frac{1}{4}k_{ij}(r_{ij}^2 - r_{0,ij}^2)^2$$

noindent where r_{ij} is the distance between particles *i* and *j*. The potential has two input parameters r_0 and k.

Derive the force.

Step 2

This is a 2-body interaction between a predefined (fixed) list of atom pairs. What is the appropriate interaction template to use? Choose one in $ESPRESSOHOME/src/interaction

Open the interaction template file. (When you close the file later, close it without saving, or else later on your compile time will be very long, because of the number of dependencies on the interaction template!) Identify the functions addForces() and computeEnergy(). Many interaction templates also contain functions such as computeVirial(), computeVirialX() (for calculating the virial in slabs along the x-direction) etc.

Find the function calls:

```
potential->_computeForce(force, dist)
```

in addForces() and

```
potential->_computeEnergy(r21)
```

in computeEnergy().

An interaction template can be combined with many different potentials (e.g. harmonic potential, Lennard Jones potential, etc.) Each potential will have its own C++ class containing functions to compute the energy and forces for that particular potential (see e.g. Harmonic.cpp/hpp, LennardJones.cpp/hpp) In turn, each potential can be combined with many different interaction templates.

You don't need to modify the interaction template file today. (Close it without saving!)

Step 3

In this step we create the .cpp, .hpp and .py files for our potential. Let's call the potential FourthPower. The FourthPower.py file will contain the end-user python interface, and in the FourthPower.cpp and FourthPower.hpp files we will create a C++ class for our potential. We will also write a wrapper which will allow the user to call the C++ code from the python interface.

3(a) Interfacing potential class and interaction template

In many cases, it's not necessary to understand the contents of this section in order to implement a new potential. If you like, you can skip directly to Section *3(b) Creating the new potential class*.

Now we need to understand how the interaction template will interface with our new class. This is done via a class template, e.g. in Potential.hpp, AnglePotential.hpp, DihedralPotential.hpp etc.

Still in $ESPRESSOHOME/src/interaction, open the file Potential.hpp. (When you close the file later, close it without saving, or else later on your compile time will be very long, because of the number of dependencies on the file!)

Find the functions _computeForce(Real3D& force, const Real3D& dist) and _computeEnergy(real dist) which you identified in the interaction template. Note that _computeForce(Real3D& force, const Real3D& dist) calls the function _computeForceRaw(force, dist, distSqr) and _computeEnergy(real dist) calls _computeEnergySqr(dist*dist) which calls _computeEnergySqrRaw(distSqr). The functions _computeForceRaw() and _computeEnergySqrRaw() are the new functions we need to write for our new potential. They will be member methods of our new C++ class FourthPower.

You don't need to modify anything in Potential.hpp today. (Close it without saving!)

3(b) Creating the new potential class

An easy way to implement the new C++ class is to identify a previously implemented potential which somewhat resembles your new potential, e.g. here we could take the Harmonic potential, which is also a 2-body potential, and which has also been interfaced with the FixedPairListInteractionTemplate.

Still in $ESPRESSOHOME/src/interaction, copy the files Harmonic.py, Harmonic.cpp and Harmonic.hpp to new files FourthPower.py, FourthPower.cpp and FourthPower.hpp. In the new files, find and replace all occurences of 'Harmonic' with 'FourthPower', and 'HARMONIC' with 'FOURTHPOWER'.

First modify FourthPower.hpp.

Note the #include statement for FixedPairListInteractionTemplate.hpp and Potential.hpp, the files you examined in *Step 2* and Step *3(a) Interfacing potential class and interaction template*.

The Harmonic potential had parameters called K and r0. You can reuse these for the FourthPower potential, along with the setters and getters setK, getK, setR0 and getR0. For better efficiency, you could also create a new variable which contains the square of r0.

Now we need functions _computeForceRaw() and _computeEnergySqrRaw(), as explained in Step *3(a) Interfacing potential class and interaction template*. Modify these functions to use the functional form of the fourth power potential as derived in *Step 1*. Note that Real3D dist, which contains the vector between the two particles, has been defined as $r_{p1} - r_{p2}$ (see addForces() in FixedPairListInteractionTemplate.hpp).

Next open Harmonic.py and FourthPower.py.

Here is an example of an end-user's python script to add an interaction using the harmonic potential:

```
harmonicbondslist = espresso.FixedPairList(system.storage)
harmonicbondslist.addBonds(bond_list) #bond_list is a list of tuples [(particleindex_i,particleindex_j),...
↪]
harmonic_potential = espresso.interaction.Harmonic(K=10.0, r0=1.0, cutoff = 5.0, shift = 0.0)
harmonic_interaction = espresso.interaction.FixedPairListHarmonic(system, harmonicbondslist,
↪potential=harmonic_potential)
system.addInteraction(harmonic_interaction)
```

Compare this to the contents of Harmonic.py to understand the python source code.

Our new potential FourthPower can be called by the end-user in a similar way. Since the Harmonic and FourthPower potentials have similar input parameters (K, r0) and both use the FixedPairListInteractionTemplate, you don't need to make any further modifications to the file FourthPower.py, besides replacing 'Harmonic' with 'FourthPower'.

Next open FourthPower.cpp.

Here you will find the C++/python interface, in the function registerPython(). If you want to understand this function, you will find details in *Exposing a C++ class or struct to python using boost*. You don't need to make any further modifications to this file, besides replacing 'Harmonic' with 'FourthPower'.

3(c) Including the new class in espressopp

Finally, update the files $ESPRESSOHOME/src/interaction/bindings.cpp and $ESPRESSOHOME/src/interaction/__init__.py (for example by copying and modifying all the lines referring to the Harmonic potential so that they now refer to the FourthPower potential). You need to make three modifications: to include the new .hpp file, to call the new registerPython() wrapper, and to import everything in the new python module.

Step 4

Move to the directory $ESPRESSOHOME. Update the makefiles and compile using the commands:

```
cmake -B builddir .
cmake --build builddir
```

2.5.3 Advanced exercise

For an interaction potential of your choosing, follow the steps above to implement it, e.g. a non-bonded two-body interaction, probably using VerletListInteractionTemplate and based on the LennardJones potential, or a bonded three-body interaction, probably using FixedTripleListInteractionTemplate.hpp and based on the AngularHarmonic potential.

You will probably have to write setters and getters for the parameters in your potential in your .hpp file, and make the corresponding modifications to the function registerPython() in the .cpp file and the python user interface in the .py file.

2.6 Appendices

2.6.1 Exposing a C++ class or struct to python using boost

(See http://www.boost.org/doc/libs/1_56_0/libs/python/doc/tutorial/doc/html/python/exposing.html)

Say we have a C++ struct called World:

```
struct World
{
    World(std::string msg): msg(msg) {}              // constructor
    void set(std::string msg) { this->msg = msg; }   // function set
    std::string greet() { return msg; }              // function greet
    std::string msg;                                 // member variable
};
```

Now we write the C++ class wrapper for struct World to expose the constructor and the functions greet and set to python:

```
{
    class_<World>("World", init<std::string>())
        .def("greet", &World::greet)
        .def("set", &World::set)
    ;
}
```

If there are additional constructors we can also expose them using def(), e.g. for an additional constructor which takes two doubles:

```
class_<World>("World", init<std::string>())
    .def(init<double, double>())
    .def("greet", &World::greet)
    .def("set", &World::set)
;
```

We can also expose the data members of the C++ class or struct and the associated access (getter and setter) functions using add_property(), e.g. for the variable myValue with access functions getMyValue and setMyValue:

```
.add_property("myValue",&World::getMyValue,&World::setMyValue)
```

C++ classes and structs may be derived from other classes. Say we have the C++ struct myDerivedStruct which is derived from the struct myBaseStruct:

```
struct myBaseStruct { virtual ~myBaseStruct(); };
struct myDerivedStruct : myBaseStruct {};
```

We can wrap the base class myBaseStruct as explained above:

```
<Base>("Base")
  /*...*/
  ;
```

Now when we want to wrap the class myDerivedStruct, we tell boost that it is derived from the base class myBaseStruct:

```
class_<myDerivedStruct, bases<myBaseStruct> >("myDerivedStruct")
    /*...*/
    ;
```

2.6.2 C++ templates

See http://www.cplusplus.com/doc/oldtutorial/templates/

2.6.3 typedef

typedef declaration allows you to create an alias that can be used anywhere in place of a (possibly complex) type name

```
typedef DataType AliasName;
```

2.6.4 Python notes

Syntax for classes in python

(See also https://docs.python.org/2/tutorial/classes.html)

Here is a python class called DerivedClassName which is derived from two other base classes (BaseClassName1 and BaseClassName1), is initialised with two variables x and y which have default values 1 and 2, and contains a function myFunction.

```
class DerivedClassName(BaseClassName1, BaseClassName2):
    """docstring"""          #a way of providing some documentation for the class
    def __init__(self,x=1,y=2): #takes two variable which have default values 1 and 2
        self.x = x
        self.y = y
    def myFunction(self):
        return self.x * self.y
```

PMI

PMI = parallel method invocation. For more details see the file $ESPRESSOHOME/src/pmi.py

2.7 Adaptive Resolution Simulations

2.7.1 Theory and Background

ESPResSo++ provides functionality to run adaptive resolution simulations using the Adaptive Resolution Simulation Scheme (AdResS). In AdResS molecules in different regions in a simulation box are described by different non-bonded force fields, typically atomistic (AT) and coarse-grained (CG). These different subregions are interfaced and coupled via a hybrid region, where the interaction smoothly changes. Molecules can diffuse between the different regions and change their interaction on the fly.

There are two different AdResS approaches: The force-based scheme, in which forces are interpolated, as well as the energy-based scheme (Hamiltonian AdResS or H-AdResS) which interpolates on the level of potential energies. In

force-based AdResS (see, for example, Praprotnik et al., J. Chem. Phys. 123, 224106 (2005) as well as Annu. Rev. Phys. Chem. 59, 545 (2008)), we have for the net force between the molecules α and β

$$\mathbf{F}_{\alpha|\beta} = \lambda(\mathbf{R}_\alpha)\lambda(\mathbf{R}_\beta)\mathbf{F}^{\mathrm{AT}}_{\alpha|\beta} + (1 - \lambda(\mathbf{R}_\alpha)\lambda(\mathbf{R}_\beta))\,\mathbf{F}^{\mathrm{CG}}_{\alpha|\beta},$$

where $\mathbf{F}^{\mathrm{AT}}_{\alpha|\beta}$ is an AT force field based on the individual atoms belonging to the molecules α and β and λ is a position dependent resolution function smoothly changing from 1 in the AT region to 0 in the CG region via the hybrid buffer region. It is evaluated based on the molecules' center of mass positions $\mathbf{R}_\alpha$. Note that there can of course also be bonded interactions, but these are typically not interpolated, as they are computationally usually much cheaper to evaluate than the non-bonded forces. For the sake of clarity, we omit them here.

In H-AdResS (see Potestio et al., Phys. Rev. Lett. 110, 108301 (2013)), interpolation is performed directly on potential energies in the Hamiltonian as

$$H = \sum_\alpha \sum_{i\in\alpha} \frac{\mathbf{p}^2_{\alpha i}}{2m_{\alpha i}} + \sum_\alpha \left\{\lambda(\mathbf{R}_\alpha)V^{\mathrm{AT}}_\alpha + (1 - \lambda(\mathbf{R}_\alpha))\,V^{\mathrm{CG}}_\alpha\right\},$$

where the first term corresponds to the kinetic energy and we again omitted intramolecular interactions. The forces obtained from this Hamiltonian are

$$\begin{aligned}\mathbf{F}_{\alpha i} = &\sum_{\beta\neq\alpha}\sum_{j\in\beta}\left\{\frac{\lambda_\alpha + \lambda_\beta}{2}\mathbf{F}^{\mathrm{AT}}_{\alpha i|\beta j} + \left(1 - \frac{\lambda_\alpha + \lambda_\beta}{2}\right)\mathbf{F}^{\mathrm{CG}}_{\alpha i|\beta}\right\} \\ &- \left[V^{\mathrm{AT}}_\alpha - V^{\mathrm{CG}}_\alpha\right]\nabla_{\alpha i}\lambda_\alpha.\end{aligned}$$

The last term, the so-called drift force, comes from applying the position gradient on the position-dependent resolution function λ. It acts only in the hybrid region and unphysically pushes molecules from one region to the other. Therefore, it needs to be corrected. On the other hand, force-based AdResS, contrary to H-AdResS, does not allow a Hamiltonian formulation at all.

Usually, the force fields used in the different regions of the adaptive simulation setup have significantly different pressures given the same temperature and particle density. This pressure gradient leads, in addition to the drift force in H-AdResS, to particles being pushed across the hybrid region. Eventually, the system would evolve to an equilibrium state with a inhomogeneous density profile across the simulation box. Therefore, correction forces needs to be applied in the hybrid region to counter these effects. In H-AdResS one can use a so-called free energy correction (FEC), which on average cancels the drift force in the hybrid region (see Potestio et al., Phys. Rev. Lett. 110, 108301 (2013)). The FEC corresponds to the free energy difference between the subsystems and can, for instance, be derived from Kirkwood thermodynamic integration. An alternative approach which is typically used to cancel the pressure gradient in force-based AdResS is the so-called thermodynamic force (see Fritsch et al., Phys. Rev. Lett. 108, 170602 (2012)). It is derived by constructing the correction directly from the distorted density profile which is obtained without any correction and then refined iteratively.

2.7.2 ESPResSo++ code

Several measures had to be taken to implement adaptive resolution simulations in ESPResSo++. On top of the normal particles, which serve as the CG particles in AdResS, another layer of extra AT particles is introduced such that one has access to both atomistic and CG particles throughout the whole system. A mapping between the two defines which atoms belong to which CG bead. The resolution function λ is implemented as a particle property of the CG particles that is updated after each integration step based on the new positions. This happens in an extension to the Velocity Verlet integrator. The actual adaptive resolution scheme is then implemented via new interaction templates that define how forces and energies are computed in force-based and energy-based AdResS. These templates use for particle pairs in the atomistic region the actual atoms, this is the AT particles, for the force and energy computation while in the CG region they use the CG particles. In the hybrid region, both are used, as defined in the equations above. The drift term of H-AdResS is implemented similarly. Furthermore, the AdResS integrator extension makes sure that the atomistic particles in the CG region travel along with the CG particles and that similarly the CG particles in the AT region are

properly updated according to the new atomistic positions after each integration step. The FEC as well as a module to apply the Thermodynamic Force are implemented as integrator extensions.

In the following, we explain the new features step by step (more details about parameters etc. can be found in the documention of the different classes).

Adress Domain Decomposition

When setting up the storage we have to use an appropriate domain decomposition that accomodates storage and proper interprocessor communication of both AT and CG particles.

```
# (H-)AdResS domain decomposition
system.storage = espressopp.storage.DomainDecompositionAdress(system, nodeGrid, cellGrid)
```

Atomistic and Coarse-Grained particles

When adding particles to the storage, we have to define them as atomistic or coarse-grained. This has been implemented as the particle property "adrat". If it is 0, the particle is coarse-grained. If it is 1, it is an atomistic particle.

```
# add particles to system
system.storage.addParticles(allParticles, "id", "pos", "v", "f", "type", "mass", "adrat")
```

When adding the particles as above, it is important that a set of atomistic particles belonging to one CG particle appears in the list of particles allParticles always after the corresponding CG particle.

Next, the FixedTupleListAdress defines which atomistic particles belong to which coarse-grained particles.

```
# create FixedTupleList object and add the tuples
ftpl = espressopp.FixedTupleListAdress(system.storage)
ftpl.addTuples(tuples)
system.storage.setFixedTuplesAdress(ftpl)
```

In this example, tuples is a list of tuples, where each tuple itself is another short list in which the first element is the CG particle and the other elements are the AT particles belonging to it. Note that in ESPResSo++ the CG particle is positioned always in the center of mass of its atoms.

Having set up the FixedTupleList, we can also set up an AdResS fixed pair list that defines bonds between AT particles within individual molecules. This is done in the following way:

```
# add bonds between AT particles
fpl = espressopp.FixedPairListAdress(system.storage, ftpl)
fpl.addBonds(bonds)
```

where bonds is a list of bonds between AT particles within CG molecules. Similarly, triple lists for angles, quadruple lists for dihedrals etc. are set up. Compared to conventional bonds, angles, etc. between different normal CG particles one just adds the suffix Adress to the appropriate list object and provides it also with the FixedTupleList (ftpl in the example). Note that you can define several different such fixed pair lists and you can, for example, also in AdResS simulations still use the normal FixedPairList to define bonds between regular CG particles.

AdResS Verlet List

Next, we construct the AdResS Verlet list object for non-bonded interacting particle pairs:

```
# AdResS Verlet list
vl = espressopp.VerletListAdress(system, cutoff=0.8, adrcut=1.4,
                                 dEx=1.5, dHy=1.0,
                                 adrCenter=[Lx/2, Ly/2, Lz/2], sphereAdr=False)
```

We have to provide the cutoffs of the list as well as the sizes of the atomistic and hybrid regions. The parameter cutoff corresponds to the cutoff used for CG particle pairs with both particles being in the CG region, while adrcut is the cutoff for all other particle pairs (at least one particle of the pair is in the AT or hybrid region). We want to stress that this pair list is build based on the CG particles' positions. Hence, for the AT and hybrid region one needs in some situations to provide a Verlet list cutoff (adrcut) slightly larger than the actual maximum interaction range of the potential, in order to not lose interactions between some atom pairs. Let us clarify this with an example: Thinking of a pair of water molecules, both coarse-grained into single beads, these CG beads could be farther apart than the interaction cutoff. Two hydrogen atoms pointing towards each other, however, could in fact still be in interaction range. Therefore, an appropriate buffer needs to be provided.

The sphereAdr flag decides how to geometrically set up the change in resolution. If it's true, the AT region is a spherical region positioned at adrCenter with radius dEx. If sphereAdr is false, the resolution changes along the x-axis of the system and dEx corresponds to half the width of the AT region. dHy always is the full width of the hybrid region. Instead of providing a 3D position for adrCenter as above, one can also provide a particle ID of a CG particle. In this case, the atomistic region will follow the movement of the particle. This should be only done, however, for force-based AdResS, since it would break the Hamiltonian character of H-AdResS, and also only when using a spherical adaptive geometry. Then, however, it is even possible to provide a list of particle IDs, in which case the AT region corresponds to the overlap of the spherical regions defined by the individual particles provided in the list. It will deform accordingly while these particle move.

Interactions

When adding interactions to the system we have to use the corresponding interaction templates. Here is how to set up a non-bonded interaction in a H-AdResS system:

```
# H-AdResS non-bonded interaction: WCA potential between AT particles
# and tabulated potential between CG particles
interNB = espressopp.interaction.VerletListHadressLennardJones(vl, ftpl)
potWCA  = espressopp.interaction.LennardJones(epsilon=1.0, sigma=1.0, shift='auto',
                                              cutoff=rca)
potCG = espressopp.interaction.Tabulated(itype=3, filename=tabCG, cutoff=rc) # CG
interNB.setPotentialAT(type1=1, type2=1, potential=potWCA) # AT
interNB.setPotentialCG(type1=0, type2=0, potential=potCG) # CG
system.addInteraction(interNB)
```

First, we define the appropriate interaction type, in H-AdResS this is VerletListHadressLennardJones. Next we define the actual potentials. Then we associate them with the H-AdResS interaction and add the interaction to the system. For force-based AdResS the only change required would be to use the VerletListAdressLennardJones interaction.

Note that the here used interaction, VerletListHadressLennardJones, couples only Lennard-Jones-type potentials with tabulated ones. However, there exist more such interaction templates for other potentials and potential combinations.

AdResS Integrator Extension

Finally, we have to set up the AdResS integrator extension:

```
# AdResS integrator extension
adress = espressopp.integrator.Adress(system, verletlist, ftpl, regionupdates = 1)
integrator.addExtension(adress)
```

It takes as arguments the Verlet list and the fixed tuple list. Additionally, for the case of a moving and/or deforming AdResS region based on one or more particles, the parameter regionupdates specifies how regularly we want to update the shape of the AdResS region in number of steps. This is to avoid as much as possible of the additional communication required to inform different processors of the change of the AdResS region. The parameter defaults to 1 and is not used at all for static AdResS regions.

Having set up the AdResS extension, we can distribute all particles in the box and place the CG molecules in the centers of mass of the atoms which they belong to. This can be done conveniently via

```
# distribute atoms and CG molecules according to AdResS domain decomposition,
# place CG molecules in the center of mass
espressopp.tools.AdressDecomp(system, integrator)
```

Free Energy Compensation

When using H-AdResS, we probably want to also employ a FEC. This can be done as follows:

```
# set up FEC
fec = espressopp.integrator.FreeEnergyCompensation(system, center=[Lx/2, Ly/2, Lz/2])
fec.addForce(itype=3, filename="table_fec.tab", type=1)
integrator.addExtension(fec)
```

The FEC takes as arguments the system object as well as the center of the AT region. Then we add the actual force, which needs to be provided in a table (first column: resolution λ, second: energy, third: force). itype defines which type of interpolation should be used for values between the ones provided in the table. 1 corresponds to linear interpolation, 2 to akima splines, 3 to cubic splines. We suggest to use cubic splines. The FEC is applied on CG particles and distributed among the atoms belonging to the CG particle. type specifies the CG particle type for which this correction should be applied. One can, for example, use different FECs for different molecules types.

Thermodynamic Force

When using force-based AdResS, or, alternatively, in addition to the FEC in H-AdResS, we can use the thermodynamic force. It can be set up in the following way, very similar to the FEC before:

```
# set up Thermodynamic Force
thdforce = espressopp.integrator.TDforce(system, verletlist)
thdforce.addForce(itype=3,filename="table_tf.tab",type=1)
integrator.addExtension(thdforce)
```

It works largely as for the FEC with the following differences: The table should not provide resolution values in the first column but actual distance values, this is, the distance from the (closest) AT region center. This allows to extend the application of the thermodynamic force slightly beyond the borders of the hybrid region where the resolution is constant. Furthermore, the Thermodynamic Force needs the verletlist as argument.

It is also possible to define a thermodynamic force, which is suited for an adaptive resolution setup with an AT region that is constructed via the overlap of several spherical regions. In this case, the extension needs more information:

```
# set up Thermodynamic Force
thdforce = espressopp.integrator.TDforce(system, verletlist, startdist = 0.9,
                                         enddist = 2.1, edgeweightmultiplier = 20)
thdforce.addForce(itype=3,filename="table_tf.tab",type=1)
integrator.addExtension(thdforce)
```

It gets three more parameters, startdist, enddist and edgeweightmultiplier. startdist explicitely says at which distance from the center of the closest AT region defining particle the thermodynamic force starts to act and enddist says where it ends. Hence, these value should correspond to what is actually written in the table. edgeweightmultiplier is a parameter that speficies how precisely the thermodynamic force should be applied in the overlap regions of different spheres. For most applications, however, 20 should provide reasonable results (for details, see Kreis et al., J. Chem. Theory Comput. 12, 4067 (2016)). The 3 additional parameters are of course also present with some default values in the basic case, but they are ignored unless we have an AT region that is constructed via the overlap of several spherical regions.

2.7.3 Examples

We have provided several example scripts and setups that are available in the ESPResSo++ source code at examples/adress. Most of them are based on published papers.

The reader is strongly encouraged to play around with them and test what happens when the setups are modified. Possible questions to ask are provided at the end of the following subsections, which explain the individual examples in more detail.

Force-AdResS: Tetrahedral Liquid

Subfolder: fadress_tetraliquid. This example consists of the system that was used in the initial work introducing the force-based adaptive resolution method (see Praprotnik et al., J. Chem. Phys. 123, 224106 (2005) and Phys. Rev. E 73, 066701 (2006)). A liquid composed of artificial tetrahedral molecules, i.e. each molecule consists of 4 bonded atoms arranged in a tetrahedral geometry, is coupled to a CG model which describes the molecules as individual beads.

Questions: The geometry is set in such a way that the resolution changes along the x-axis of the box. Try changing the setup such that the AT region is of spherical shape. You can also try removing the thermostat. Does the system conserve energy? Also vary the size of the atomistic region and see what happens. Can you also make the system all-atomistic or all-CG? You can also try to compare computational times.

Force-AdResS: A Protein in Water

Subfolder: fadress_protein. This system is an aqueous solution of the regulatory protein ubiquitin. The atomistic protein and the atomistic water around it is coupled to a coarse-grained water model, which maps water molecules farther away from the protein to single beads. The CG water interaction was parametrized with iterative Boltzmann inversion (IBI). This system is similar to the setup which was used by Fogarty et al. (J. Chem. Phys. 142, 195101 (2015)) to study the structure and dynamics of a protein hydration shell.

Questions: The setup is significantly more complicated than the previous system. Try to understand the script. You can also have a look into the the actual source code and try to understand, for example, how the gromacs parser works. The example is set up as a fully atomistic simulation by setting the size of the atomistic region to a value larger than the simulation box. Try to change the script such that it is an actual adaptive setup. Do not forget the thermodynamic force! Furthermore, how is the high-resolution region positioned now?

Force-AdResS: Self-Adjusting Adaptive Resolution Simulations

Subfolder: fadress_selfadjusting. This setup demonstrates how force-based adaptive resolution simulations with self-adjusting high-resolution regions can be set up (Kreis et al., J. Chem. Theory Comput. 12, 4067 (2016)). The system is a polyalanine-9 molecule in aqueous solution. A spherical AT region is associated with each atom of the peptide such that the overall AT region formed by the overlap of all these spheres elegantly envelops the peptide. The peptide starts in an extended configuration and as it folds, the AT region surrounding it adjusts itself accordingly. At the outside, we use again a coarse-grained IBI single-bead model for the water molecules.

Questions: Can you change the system such that fewer atoms are associated with AT region, for example, only the heavy atoms? Can you change the update frequency of the shape of the AT region?

H-AdResS: Tetrahedral Liquid

Subfolder: hadress_tetraliquid. This is the system used by Potestio et al. in the paper that proposed the H-AdResS method (Phys. Rev. Lett. 110, 108301 (2013)). It is again a simple system composed of tetrahedral molecules that change their resolution and become individual beads in the CG region. The interpolation occurs along the x-axis. This example has three subfolders.

The first folder hadress_tetraliquid_plain runs a simple H-AdResS simulation without any free energy correction. Hence, the drift force strongly pushes molecules from one region to the other. The script contains analysis routines which measure both a density and a pressure profile along the direction of resolution change while the simulation is running. Gathering enough statistics takes a while, but we have also provided reference profiles which are obtained after a sufficiently long simulation. Have a look at them and try to interpret them.

The second folder hadress_tetraliquid_FEC contains the same setup but with a free energy correction. For this, two tables are provided, table_FEC_Helmholtz.dat and table_FEC_Gibbs.dat. They were derived via Kirkwood thermodynamic integration. The first one is based on the Helmholtz free energy difference per particle between the two subsystem, and the second one corresponds to the Gibbs free energy difference per particle. Two density and pressure profiles obtained while applying these correction are also shown. Try to interpret them.

The third folder hadress_tetraliquid_KTI contains a simple implementation of Kirkwood thermodynamic integration (KTI) which could in principle, when run for long enough, be used to derive the FEC. This is not an adaptive resolution simulation. Instead, we tell the AdResS integrator extension that we want to run KTI. Then, the extension does not modify the resolution values associated with the different molecules and we can change them by hand during the simulation. In this way, we can set up a simulation in which we change the resolution of all molecules in the system every few steps and slowly proceed from a complete CG system to an all-atom one. Have a look and try to understand what is going on.

There are many more interesting things you can try out: Are the H-AdResS simulations energy conserving? Add the commented Langevin thermostat and compare. Also vary the timestep. Additionally, you can change the size of the hybrid region. What happens if it becomes smaller or larger? Furthermore, what happens if you change the system from H-AdResS to force-based AdResS?

H-AdResS: Water

Subfolder: hadress_water. This is a slightly more advanced H-AdResS system in which an atomistic model is coupled to a coarse-grained one, mapping the three water atoms onto single beads.

Questions: Feel free to play around with the system. You could also try to figure out, how the gromacs parsers sets up the interactions and chooses the right H-AdResS interactions.

2.7.4 Adaptive Resolution Simulations with Multiple Time Stepping

Coarse-grained (CG) potentials are typically significantly softer than atomistic (AT) force fields and the corresponding equations of motions can be solved using a larger time step. This suggests the use of multiple time stepping (MTS) techniques in adaptive resolution simulations, in which both AT and CG potentials are present simultaneously. For simulations in which the CG region is much larger than the AT one, this promises a significant speed-up compared to calculations in which a single short time step is used for the whole system. ESPResSo++ provides a RESPA-based MTS scheme (J. Chem. Phys. 97, 1990 (1992)), in which the CG interactions are integrated on a slow timescale and all AT interactions (bonded and non-bonded) on a faster timescale. The scheme can be used both with force-based AdResS and energy-based H-AdResS.

Note that MTS within AdResS simulations can be interpreted as spatially adaptive MTS, in which the integration time scales of the different forces within and between molecules depend on its positions in the simulation box. Large time steps are used in one domain while short time steps are used in another domain of the box. This is in contrast to usual MTS applications, in which the same multiple time stepping is applied everywhere in the system and the separation is usually just between bonded and non-bonded interactions, but without any spatial dependency.

AdResS with Multiple Time Stepping: Implementation

The AdResS-MTS scheme is implemented in ESPResSo++ by two modifications. On the one hand, a new interaction type (*NonbondedSlow*) and a new integrator that implements the RESPA scheme are provided. The integrator calculates all interactions of type *NonbondedSlow* on the long time scale, while all other interactions are treated on the fast timescale. On the other hand, a set of new interaction templates for adaptive resolution interactions are provided. The fast AT and the slow CG adaptive resolution interactions are now implemented in separate interaction templates that have different types (*Nonbonded* and *NonbondedSlow*), which can be exploited by the MTS integrator. Furthermore, the user can specify whether the Thermodynamic Force and the Free Energy Compensation are applied on the slow or fast time scale. Code examples are below:

```
# set up the atomistic part of a force-based adaptive resolution interaction. This interaction template␣
↪incorporates both a Lennard-Jones
# term and a Reaction Field term for the force to compute both the Van der Waals and electrostatic␣
↪forces during one loop over
# the atomistic- and hybrid-region particle pairs
non_bonded_interaction_at = espressopp.interaction.
↪VerletListAdressATLenJonesReacFieldGen(verletlist, ftpl)
potLJ = espressopp.interaction.LennardJones(epsilon=epsilon, sigma=sigma, shift='auto',␣
↪cutoff=interaction_cutoff_at)
potQQ = espressopp.interaction.ReactionFieldGeneralized(prefactor=138.935485, kappa=0.0, epsilon1=1.
↪0, epsilon2=80.0, cutoff=interaction_cutoff_at, shift="auto")
non_bonded_interaction_at.setPotential1(type1=1, type2=1, potential=potLJ)
non_bonded_interaction_at.setPotential2(type1=1, type2=1, potential=potQQ)
non_bonded_interaction_at.setPotential2(type1=1, type2=0, potential=potQQ)
non_bonded_interaction_at.setPotential2(type1=0, type2=0, potential=potQQ)
system.addInteraction(non_bonded_interaction_at)
```

```
# set up the coarse-grained part of a force-based adaptive resolution interaction
non_bonded_interaction_cg = espressopp.interaction.VerletListAdressCGTabulated(verletlist, ftpl)
potCG = espressopp.interaction.Tabulated(itype=3, filename="table_ibi.dat", cutoff=interaction_cutoff_
↪cg)
non_bonded_interaction_cg.setPotential(type1=typeCG, type2=typeCG, potential=potCG)
system.addInteraction(non_bonded_interaction_cg)
```

```
# set up the RESPA VelocityVerlet Integrator (timestep is the short time step,
# and multistep is an integer multiplier to construct the long time step as the
# product of the short time step with the multiplier)
integrator = espressopp.integrator.VelocityVerletRESPA(system)
integrator.dt = timestep
integrator.multistep = multistep
```

```
# add AdResS extension. It also needs to know about the multiple time stepping (multistep parameter)
adress = espressopp.integrator.Adress(system, verletlist, ftpl, multistep=multistep)
integrator.addExtension(adress)
```

```
# add Thermodynamic Force and specify whether the force is applied
# together with the slow (slow=True) or fast (slow=False) forces
thdforce = espressopp.integrator.TDforce(system, verletlist, slow=False)
thdforce.addForce(itype=3, filename="table_tf.xvg", type=typeCG)
integrator.addExtension(thdforce)
```

AdResS with Multiple Time Stepping: Examples

Subfolders within the AdResS examples folder: multiple_time_stepping_fadress for a force-based AdResS MTS simulation and multiple_time_stepping_hadress for an H-AdResS MTS simulation of liquid water. In both examples, the system is a box of liquid water in which the resolution changes along the x-axis. The atomistic model is the SPC/Fw force field with a Reaction Field approach to treat electrostatics. The force-based AdResS example uses a tabulated iterative Boltzmann inversion-based potential in the CG region, while the H-AdResS example employs a simple truncated Harmonic potential to describe the CG interactions.

Have a look at the examples and modify the different time steps. For example, you can test the effect of different time step configurations on the energy conservation in H-AdResS or you investigate whether it makes a difference to apply the corrections on the slow or fast time scale.

2.7.5 Path Integral-AdResS

The path integral (PI) formalism can be used in molecular simulations to account for the quantum mechanical delocalization of light nuclei. It is frequently used, for example, when modeling hydrogen-rich chemical and biological systems, such as proteins or DNA. In the PI methodology, quantum particles are mapped onto classical ring polymers, which represent delocalized wave functions. This renders the PI approach computationally highly expensive (for a detailed introduction see, for example, *M. E. Tuckerman, Statistical Mechanics: Theory and Molecular Simulation*). However, in practice the quantum mechanical description is often only necessary in a small subregion of the overall simulation.

Recently, a PI-based adaptive resolution scheme was developed that allows to include the PI description only locally and to use efficient classical Newtonian mechanics in the rest of the system (*J. Chem. Phys.* **147**, 244104 (2017) and *J. Chem. Theory Comput.* **12**, 3030 (2016)). In this approach the ring polymers are forced to collapse to classical, point-like particles in the classical region. This is achieved by introducing a position-dependent and adaptively changing particle mass which controls the spring constants between the ring polymer beads. Note that this does not necessarily affect the separate "kinietic" masses which are typically introduced in Path Integral Molecular Dyamics.

The method is based on an overall Hamiltonian description and it is consistent with a bottom-up PI quantization procedure. It allows for the calculation of both quantum statistical as well as approximate quantum dynamical quantities in the quantum subregion using ring polymer or centroid molecular dynamics. The methodology is implemented in the ESPResSo++ package and it also makes use of multiple time stepping. For technical details, please see the original publications.

PI-AdResS Implementation

PI-AdResS is implemented in ESPResSo++ by the addition of further particle properties, this is, a variable mass parameter to control the ring polymers' spring constants and a path integral bead (pib) number indicating which imaginary time slice or Trotter number a particle corresponds to. A system is typically set up in such a way that the physical atoms correspond to the coarse-grained ESPResSo++ particles, while ESPResSo++'s atoms, which are linked to the coarse-grained particles using the fixed tuple list, are the actual beads of the ring polymer. A coarse-grained ESPResSo++ particle then correspond to the ring polymer centroid, which are therefore used for the construction of the Verlet list, and control whether a ring polymer is send to another CPU in parallel simulations.

Furthermore, new interaction templates were implemented to accommodate the calculation of interactions between atoms in a path integral-based manner and a new multiple time stepping integrator was developed (see the user manual for detailed documention of the classes and the example for usage in practice).

```
# path integral-based adaptive resolution interaction that employs a tabulated potential for the potential␣
↪in the
# path integral region and a Lennard Jones potential in the classical region (where the ring polymers are␣
↪collapsed)
non_bonded_interaction = espressopp.interaction.VerletListPIadressTabulatedLJ(verletlist, fixedtuplelist,
↪ TrotterNumber, speedup_in_CL_region)
```

```
# path integral-based adaptive resolution multiple time stepping integrator
integrator = espressopp.integrator.PIAdressIntegrator(system, verletlist, timestep_short, multiplier_
↪short_to_medium, multiplier_medium_to_long, nTrotter, realkinmass, constkinmass, temperature,␣
↪gamma, centroidThermostat, CMDparameter, PILE, PILElambda, CLmassmultiplier, freezeCLrings,␣
↪KTI)
```

PI-AdRresS Example: Liquid Water

Subfolder: piadress_water within the AdResS example folder. This system is a box of liquid water and the resolution changes along the x-axis. In the center of the box, the molecules behave quantum mechanically with extended ring polymers, while elsewhere the ring polymers collapse to pointlike particles, making them behave classically and allowing for an efficient force computation. In the PI region, a tabulated potential is used, which was specifically developed for PI-based simulations, while in the classical region a simple WCA potential is employed. The setup is similar to those used in *J. Chem. Phys.* **147**, 244104 (2017).

Have a look at the example and try to understand and play around with the many available options for the PI-based adaptive resolution setup. Use the user manual and the original publication as reference.

2.8 Thermodynamic integration

2.8.1 Theoretical explanation

Thermodynamic integration (TI) is a method used to calculate the free energy difference between two states A and B. For the theoretical background, see e.g. http://www.alchemistry.org. In this tutorial, we show how to perform TI calculations with ESPResSo++. We calculate the free energy of solvation of methanol in water. The complete python script is available in the ESPResSo+ source code under examples/thd_integration_solvation

To do TI, we define states A and B, with potentials U^A and U^B. We then construct a pathway of intermediate states between A and B by defining a parameter λ that takes values between 0 and 1 and writing the system potential U as a

function of λ, U^A and U^B. The free energy difference between the states A and B is then given by

$$\Delta A = \int_0^1 \left\langle \frac{dU(\lambda)}{d\lambda} \right\rangle_\lambda \mathrm{d}\lambda$$

In practise, we discretise λ and perform a series of MD simulations with different λ values between 0 and 1, sampling $\frac{dU(\lambda)}{d\lambda}$ in each simulation.

To calculate the solvation free energy of methanol in water, we use a box of water containing one methanol molecule. We simulate desolvation via two separate TI calculations. (Note that the procedure described here is decoupling, and solute-solute interactions will be treated differently if you're doing annihilation instead of decoupling, see Note 1.)

Step 1: free energy change for switching off the Coulombic interactions

State A: methanol has full non-bonded (Coulomb and Lennard Jones) interactions with the solvent

State B: methanol has only Lennard Jones interactions with the solvent

---

Step 2: free energy change for switching off the Lennard Jones interactions

State A: methanol has only Lennard Jones interactions with the solvent

State B: methanol has no interaction with the solvent

Step 1 can be done using a linear function of λ:

$$U(\lambda_C) = (1 - \lambda_C)U_C^A + U_{unaffected}$$

where U_C^A is the solute-solvent Coulombic interaction in state A. In ESPResSo++ the charges used for state A are the particle charges contained in the particle property charge. The charges in state B are zero, so $U_C^B(q)$ does not appear in the expression. (The case where A and B both have non-zero charges is not implemented in ESPResSo++). The term $U_{unaffected}$ is all other parts of the potential that don't change with λ_C including all bonded interactions, any solute-solute Coulombic interactions, solvent-solvent Coulombic interactions and all Lennard-Jones interactions. The parameter λ_C goes from 0 to 1 in Step 1.

Step 2 must be done using a softcore potential because of the singularity in the Lennard-Jones potential at $r_{ij} = 0$.

$$\begin{aligned} U(\lambda_L) &= \sum_{i,j} U_L(r_{ij}, \lambda_L) + U_{unaffected} \\ U_L(r_{ij}, \lambda_L) &= (1 - \lambda_L)U_H^A(r_A) + \lambda_L U_H^B(r_B) \\ r_A &= (\alpha\sigma_A^6\lambda^p + r_{ij}^6)^{1/6} \\ r_B &= (\alpha\sigma_B^6(1 - \lambda)^p + r_{ij}^6)^{1/6} \end{aligned}$$

The terms $U_H^A(r_A)$ and $U_H^B(r_B)$ are the normal Lennard-Jones 12-6 hardcore potentials:

$$U_H^A(r_A) = 4.0\epsilon_A\left(\frac{\sigma_A}{r_A}^{12} - \frac{\sigma_A}{r_A}^{6}\right)$$

The sum $\sum_{i,j} U_L(r_{ij}, \lambda_L)$ is over all solute-solvent interactions. The term $U_{unaffected}$ is all other parts of the potential that don't change with λ_L including any solute-solute Lennard-Jones interactions and solvent-solvent Lennard-Jones interactions, which are treated using standard hardcore Lennard-Jones. (In this particular example of methanol, there are no solute-solute Lennard-Jones interactions). Finally α and p are adjustable parameters of the softcore potential.

The ESPResSo++ C++ code allows for different values of ϵ_A, ϵ_B, σ_A and σ_B for every pair of atomtypes interacting via this potential. In this example, we will set ϵ_B to 0 (we are switching off the Lennard-Jones interaction). The parameter λ_L goes from 0 to 1 in Step 2.

2.8.2 ESPResSo++ code

We must perform many separate simulations, each with a different λ value. It is convenient to define a list of λ values in the python script and use an index to access a different element of the list in each separate simulation. The script for the first simulation contains these lines:

```
# Parameters for Thermodynamic Integration
stateBIndices = [1,2,3,4,5,6] #indices of the methanol atoms
lambdaVectorCoul = [0.00, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50,
                    0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.85, 0.90, 0.95, 1.00, 1.000,
                    1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000,
                    1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000,
                    1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000,
                    1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000, 1.000,
                    1.000, 1.000, 1.000]
lambdaVectorVdwl = [0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00,
                    0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.00, 0.025,
                    0.050, 0.075, 0.100, 0.125, 0.150, 0.175, 0.200, 0.225, 0.250,
                    0.275, 0.300, 0.325, 0.350, 0.375, 0.400, 0.425, 0.450, 0.475,
                    0.500, 0.525, 0.550, 0.575, 0.600, 0.625, 0.650, 0.675, 0.700,
                    0.725, 0.750, 0.775, 0.800, 0.825, 0.850, 0.875, 0.900, 0.925,
                    0.950, 0.975, 1.000]
lambdaIndex = 0
lambdaTICoul = lambdaVectorCoul[lambdaIndex]
lambdaTIVdwl = lambdaVectorVdwl[lambdaIndex]
```

The list lambdaVectorCoul contains the values of λ_C and the list lambdaVectorVdwl contains the values of λ_L. The total number of simulations to do Step 1 and Step 2 will be len(lambdaVectorCoul) or len(lambdaVectorVdwl). We must make a copy of the python script for each simulation, changing each time the value of lambdaIndex.

Next we set up the Coulombic interactions, assuming we already have created a system and a verletlist. The electrostatics method used is generalised reaction field.

```
#atTypes - list of all atomtypes (integers) used in the pairs interacting via this potential
#epsilon1,epsilon2,kappa - reaction field parameters
#annihilate=False means decoupling is used (see Note 1)
#ftpl - a FixedTupleListAdResS object (see AdResS tutorial)
#for non-AdResS simulations, simply set adress=False, and the parameter ftpl is not needed
qq_adres_interaction = gromacs.setCoulombInteractionsTI(system, verletlist, nbCutoff,
                                        atTypes, epsilon1=1, epsilon2=80,
                                        kappa=0, lambdaTI=lambdaTICoul,
                                        pidlist=stateBIndices,
                                        annihilate=False, adress=True, ftpl=ftpl)
```

Now we set up the softcore Lennard Jones interaction.

```
#atomtypeparameters - dictionary of format {atomtype: {'eps': epsilon, 'sig': sigma}}
#                     where atomtype is integer and epsilon and sigma are real
#defaults - dictionary containing a key 'combinationrule' with value 1 if the contents
#          of atomtypeparameters need to be converted from c6,c12 format to
#          epsilon,sigma format; can also be an empty dictionary if no conversion needed
#sigmaSC, alphaSC, powerSC - parameters of the softcore potential
alphaSC = 0.5
powerSC = 1.0
```

(continues on next page)

(continued from previous page)

```
epsilonB = 0.0
sigmaSC = 0.3
lj_adres_interaction = gromacs.setLennardJonesInteractionsTI(system, defaults,
                                    atomtypeparameters, verletlist, nbCutoff,
                                    epsilonB=epsilonB, sigmaSC=sigmaSC, alphaSC=alphaSC,
                                    powerSC=powerSC, lambdaTI=lambdaTIVdwl,
                                    pidlist=stateBIndices, annihilate=False,
                                    adress=True, ftpl=ftpl)
```

We open an output file. In the first line we write the values of λ_C and λ_L for this simulation.

```
dhdlF = open("dhdl.xvg","a")
dhdlF.write("#(coul-lambda, vdw-lambda) = ("+str(lambdaTICoul)+", "+str(lambdaTIVdwl)+")\n")
```

During the MD run, every x number of MD steps, we return to the python level and calculate the derivatives of the energies with respect to λ.

```
dhdlCoul = qq_adres_interaction.computeEnergyDeriv()
dhdlVdwl = lj_adres_interaction.computeEnergyDeriv()
dhdlF.write(str(time)+" "+str(dhdlCoul)+" "+str(dhdlVdwl)+"\n")
```

After all simulations, we can now average $\frac{dU(\lambda)}{d\lambda}$ for each value of λ_C or λ_L, integrate over λ_C and λ_L, add the values ΔA_C and ΔA_L, and take the negative (because the procedure described here is desolvation and we want the free energy of solvation).

2.8.3 Some notes

1. This example given here uses decoupling (solute-solvent interactions are a function of λ, solute-solute interactions are not affected by changes in λ). In ESPResSo++ it is also possible to do annihilation, where both solute-solvent and solute-solute interactions are a function of λ, by setting annihilate=True when creating the non-bonded interactions.
2. The procedure described here is desolvation. To get the free energy of solvation, we take the negative of the value obtained after integration.
3. The example Python code snippets here use the helper functions gromacs.setLennardJonesInteractionsTI and gromacs.setCoulombInteractionsTI contained in $ESPRESSOHOME/src/tools/convert/gromacs.py, but this is not necessary. You can do TI with ESPResSo++ without the Gromacs parser by directly calling espresso.interaction.LennardJonesSoftcoreTI and espresso.interaction.ReactionFieldGeneralizedTI. See the documentation of these two classes.

CHAPTER

THREE

USER INTERFACE

3.1 analysis

3.1.1 espressopp.analysis.AllParticlePos

espressopp.analysis.AllParticlePos.gatherAllPositions()

Return type

3.1.2 espressopp.analysis.AnalysisBase

Overview

List of classes based on AnalysisBase:

espressopp.analysis.LBOutput

Overview

espressopp.analysis.LBOutputScreen(*args, **kwds)	
espressopp.analysis.LBOutputVzInTime(*args, ...)	
espressopp.analysis.LBOutputVzOfX(*args, **kwds)	

Details

Abstract output class for LB simulations. The implemented realisations are:

- espressopp.analysis.LBOutputScreen to output simulation progress and control flux conservation when using MD to LB coupling.
- espressopp.analysis.LBOutputVzInTime to output velocity component v_z on the lattice site with an index $(0.25 * N_i, 0, 0)$ in time.
- espressopp.analysis.LBOutputVzOfX to output local density ρ and v_z component of the velocity as a function of the coordinate x.

Note: all derived output classes have to be called from class espressopp.integrator.ExtAnalyze with specified periodicity of invokation and after this added to the integrator. See examples.

espressopp.analysis.LBOutputScreen

Computes and outputs to the screen the simulation progress (finished step) and controls mass flux conservation when using MD-to-LB coupling. Ideally, the sum of mass fluxes should be 0, i.e. $j_{LB} + j_{MD} = 0$.

class espressopp.analysis.LBOutputScreen(*system*, *lb*)

Parameters

- system (std::shared_ptr) – system object defined earlier in the python-script
- lb (lb_object) – lattice boltzmann object defined earlier in the python-script

Example:

```
>>> # initialise output to the screen
>>> outputScreen = espressopp.analysis.LBOutputScreen(system,lb)
>>>
>>> # initialise external analysis object with previously created output object
>>> # and periodicity of invocation (steps):
>>> extAnalysis = espressopp.integrator.ExtAnalyze(outputScreen,100)
>>>
>>> # add the external analysis object as an extension to the integrator
>>> integrator.addExtension( extAnalysis )
```

espressopp.analysis.LBOutputVzInTime

Computes and outputs the velocity component v_z in time on the lattice site with an index $(0.25 * N_i, 0, 0)$.

class espressopp.analysis.LBOutputVzInTime(*system*, *lb*)

Parameters

- system (std::shared_ptr) – system object defined earlier in the python-script
- lb (lb_object) – lattice boltzmann object defined earlier in the python-script

Example:

```
>>> # initialise output of the Vz as a function of time
>>> outputVzInTime = espressopp.analysis.LBOutputVzInTime(system,lb)
>>>
>>> # initialise external analysis object with previously created output object
>>> # and periodicity of invocation (steps):
>>> extAnalysis = espressopp.integrator.ExtAnalyze(outputVzInTime,100)
>>>
>>> # add the external analysis object as an extension to the integrator
>>> integrator.addExtension( extAnalysis )
```

espressopp.analysis.LBOutputVzOfX

Computes and outputs simulation progress (finished step) and controls flux conservation when using MD to LB coupling.

class espressopp.analysis.LBOutputVzOfX(*system*, *lb*)

Parameters

- system (std::shared_ptr) – system object defined earlier in the python-script
- lb (lb_object) – lattice boltzmann object defined earlier in the python-script

Example:

```
>>> # initialise output of the Vz as a function of x-coordinate
>>> outputVzOfX = espressopp.analysis.LBOutputVzOfX(system,lb)
>>>
>>> # initialise external analysis object with previously created output object
>>> # and periodicity of invocation (steps):
>>> extAnalysis = espressopp.integrator.ExtAnalyze(outputVzOfX,100)
>>>
>>> # add the external analysis object as an extension to the integrator
>>> integrator.addExtension( extAnalysis )
```

espressopp.analysis.OrderParameter

espressopp.analysis.OrderParameter(*system*, *cutoff*, *angular_momentum*, *do_cluster_analysis*, *include_surface_particles*, *ql_low*, *ql_high*)

Parameters

- system –
- cutoff –
- angular_momentum (int) – (default: 6)
- do_cluster_analysis – (default: False)
- include_surface_particles – (default: False)
- ql_low – (default: -1.0)
- ql_high (real) – (default: 1.0)

espressopp.analysis.ParticleRadiusDistribution

espressopp.analysis.ParticleRadiusDistribution(*system*)

Parameters

system –

espressopp.analysis.PressureTensor

This class computes the pressure tensor of the system. It can be used as standalone class in python as well as in combination with the integrator extension ExtAnalyze.

Example of standalone Usage:

```
>>> pt = espressopp.analysis.PressureTensor(system)
>>> print "pressure tensor of current configuration = ", pt.compute()
```

or

```
>>> pt = espressopp.analysis.PressureTensor(system)
>>> for k in xrange(100):
>>>     integrator.run(100)
>>>     pt.performMeasurement()
>>> print "average pressure tensor = ", pt.getAverageValue()
```

Example of usage in integrator with ExtAnalyze:

```
>>> pt           = espressopp.analysis.PressureTensor(system)
>>> extension_pt = espressopp.integrator.ExtAnalyze(pt , interval=100)
>>> integrator.addExtension(extension_pt)
>>> integrator.run(10000)
>>> pt_ave = pt.getAverageValue()
>>> print "average Pressure Tensor = ", pt_ave[:6]
>>> print "          std deviation = ", pt_ave[6:]
>>> print "number of measurements  = ", pt.getNumberOfMeasurements()
```

The following methods are supported:

- **performMeasurement()**
  computes the pressure tensor and updates average and standard deviation
- **reset()**
  resets average and standard deviation to 0
- **compute()**
  computes the instant pressure tensor, return value: [xx, yy, zz, xy, xz, yz]
- **getAverageValue()**
  returns the average pressure tensor and the standard deviation, return value: [xx, yy, zz, xy, xz, yz, +-xx, +-yy, +-zz, +-xy, +-xz, +-yz]
- **getNumberOfMeasurements()**
  counts the number of measurements that have been computed (standalone or in integrator) does _not_ include measurements that have been done using "compute()"

espressopp.analysis.PressureTensor(*system*)

Parameters
system –

espressopp.analysis.PressureTensorLayer

This class computes the pressure tensor of the system in layer h0. It can be used as standalone class in python as well as in combination with the integrator extension ExtAnalyze.

Example of standalone Usage:

```
>>> pt = espressopp.analysis.PressureTensorLayer(system, h0, dh)
>>> print "pressure tensor of current configuration = ", pt.compute()
```

or

```
>>> pt = espressopp.analysis.PressureTensorLayer(system)
>>> for k in xrange(100):
>>>     integrator.run(100)
>>>     pt.performMeasurement()
>>> print "average pressure tensor = ", pt.getAverageValue()
```

Example of usage in integrator with ExtAnalyze:

```
>>> pt           = espressopp.analysis.PressureTensorLayer(system)
>>> extension_pt = espressopp.integrator.ExtAnalyze(pt , interval=100)
>>> integrator.addExtension(extension_pt)
>>> integrator.run(10000)
>>> pt_ave = pt.getAverageValue()
>>> print "average Pressure Tensor = ", pt_ave[:6]
>>> print "          std deviation = ", pt_ave[6:]
>>> print "number of measurements  = ", pt.getNumberOfMeasurements()
```

The following methods are supported:

- **performMeasurement()**
  computes the pressure tensor and updates average and standard deviation
- **reset()**
  resets average and standard deviation to 0
- **compute()**
  computes the instant pressure tensor in layer h0, return value: [xx, yy, zz, xy, xz, yz]
- **getAverageValue()**
  returns the average pressure tensor and the standard deviation, return value: [xx, yy, zz, xy, xz, yz, +-xx, +-yy, +-zz, +-xy, +-xz, +-yz]
- **getNumberOfMeasurements()**
  counts the number of measurements that have been computed (standalone or in integrator) does _not_ include measurements that have been done using "compute()"

espressopp.analysis.PressureTensorLayer(*system*, *h0*, *dh*)

Parameters

- system –
- h0 –
- dh –

espressopp.analysis.PressureTensorMultiLayer

This class computes the pressure tensor of the system in *n* layers. Layers are perpendicular to Z direction and are equidistant(distance is Lz/n). It can be used as standalone class in python as well as in combination with the integrator extension ExtAnalyze.

Example of standalone Usage:

```
>>> pt = espressopp.analysis.PressureTensorMultiLayer(system, n, dh)
>>> for i in xrange(n):
>>>     print "pressure tensor in layer %d: %s" % ( i, pt.compute())
```

or

```
>>> pt = espressopp.analysis.PressureTensorMultiLayer(system, n, dh)
>>> for k in xrange(100):
>>>     integrator.run(100)
>>>     pt.performMeasurement()
>>> for i in xrange(n):
>>>     print "average pressure tensor in layer %d: %s" % ( i, pt.compute())
```

Example of usage in integrator with ExtAnalyze:

```
>>> pt           = espressopp.analysis.PressureTensorMultiLayer(system, n, dh)
>>> extension_pt = espressopp.integrator.ExtAnalyze(pt , interval=100)
>>> integrator.addExtension(extension_pt)
>>> integrator.run(10000)
>>> pt_ave = pt.getAverageValue()
>>> for i in xrange(n):
>>>   print "average Pressure Tensor = ", pt_ave[i][:6]
>>>   print "           std deviation = ", pt_ave[i][6:]
>>> print "number of measurements  = ", pt.getNumberOfMeasurements()
```

The following methods are supported:

- **performMeasurement()**
  computes the pressure tensor and updates average and standard deviation
- **reset()**
  resets average and standard deviation to 0
- **compute()**
  computes the instant pressure tensor in *n* layers, return value: [xx, yy, zz, xy, xz, yz]
- **getAverageValue()**
  returns the average pressure tensor and the standard deviation, return value: [xx, yy, zz, xy, xz, yz, +-xx, +-yy, +-zz, +-xy, +-xz, +-yz]
- **getNumberOfMeasurements()**
  counts the number of measurements that have been computed (standalone or in integrator) does _not_ include measurements that have been done using "compute()"

espressopp.analysis.PressureTensorMultiLayer(*system*, *n*, *dh*)

Parameters

- system –
- n –

- dh –

espressopp.analysis.Temperature

Calculate the temperature of the system (in k_BT units).

espressopp.analysis.Temperature(*system*)

Parameters
system (std::shared_ptr) – system object

Returns
temperature

Return type
real

Temperature of the system of N particles is calculated as:

$$T = \frac{1}{N_f} \sum_{i=1}^{N} m_i v_i^2,$$

where m_i and v_i are the mass and velocity of a particle i.

$N_f = 3N$ is the number of the system's degrees of freedom.

Example:

```
>>> # declare an object, e.g., T:
>>> T   = espressopp.analysis.Temperature(system)
>>>
>>> # later in your script compute temperature and print it:
>>> print T.compute()
```

espressopp.analysis.Test

Test class for any analysis tool.

espressopp.analysis.Test(*system*)

Parameters
system –

Details

This abstract base class provides the interface and some basic functionality for classes that do analysis or observable measurements

It provides the following methods:

espressopp.analysis.AnalysisBase.compute()

Computes the instant value of the observable.

Return type
a python list or a scalar

espressopp.analysis.AnalysisBase.getAverageValue()

Returns the average value for the observable and the standard deviation.

Return type
a python list

espressopp.analysis.AnalysisBase.getNumberOfMeasurements()

counts the number of measurements that have been performed (standalone or in integrator) does _not_ include measurements that have been done using "compute()"

Return type

espressopp.analysis.AnalysisBase.performMeasurement()

Computes the observable and updates average and standard deviation

Return type

espressopp.analysis.AnalysisBase.reset()

Resets average and standard deviation

Return type

3.1.3 espressopp.analysis.Autocorrelation

espressopp.analysis.Autocorrelation(*system*)

Parameters
system –

espressopp.analysis.Autocorrelation.clear()

Return type

espressopp.analysis.Autocorrelation.compute()

Return type

espressopp.analysis.Autocorrelation.gather(*value*)

Parameters
value –

Return type

3.1.4 espressopp.analysis.CMVelocity

Compute and reset (set to zero) the center-of-mass (CM) velocity of the system.

class espressopp.analysis.CMVelocity(*system*)

Parameters
system (espressopp.System) – system object

Note: CMVelocity can be attached to the *integrator*. In this case the reset() method is called, so you will reset the CM-velocity every *n-th* steps.

Methods

compute()

Compute the CM-velocity of the system

Return type
Real3D

reset()

Reset (set to zero) the CM-velocity of the system. Done by computing the CM-velocity of the system and subtracting it then from every particle.

Return type
void

Example of resetting velocity

```
>>> total_velocity = espressopp.analysis.CMVelocity(system)
>>> total_velocity.reset()
```

Example of attaching to integrator

```
>>> # This extension can be attached to integrator
>>> # and run `reset()` every `n-th` steps.
>>> total_velocity = espressopp.analysis.CMVelocity(system)
>>> ext_remove_com = espressopp.integrator.ExtAnalyze(total_velocity, 10)
>>> integrator.addExtension(ext_remove_com)
```

3.1.5 espressopp.analysis.ConfigsParticleDecomp

espressopp.analysis.ConfigsParticleDecomp(*system*)

Parameters
system –

espressopp.analysis.ConfigsParticleDecomp(*system*, *chainlength*, *start_pid*)

Parameters

- system –
- chainlength –
- start_pid –

espressopp.analysis.ConfigsParticleDecomp.clear()

Return type

espressopp.analysis.ConfigsParticleDecomp.compute()

Return type

espressopp.analysis.ConfigsParticleDecomp.gather()

Return type

espressopp.analysis.ConfigsParticleDecomp.gatherFromFile(*filename*)

Parameters
filename –

Return type

3.1.6 espressopp.analysis.Configurations

- *gather()* add configuration to trajectory
- *clear()* clear trajectory
- *back()* get last configuration of trajectory
- *capacity* maximum number of configurations in trajectory further adding (*gather()*) configurations results in erasing oldest configuration before adding new one capacity=0 means: infinite capacity (until memory is full)
- *size* number of stored configurations

usage:

storing trajectory

```
>>> configurations = espressopp.Configurations(system)
>>> configurations.gather()
>>> for k in xrange(100):
>>>   integrator.run(100)
>>>   configurations.gather()
```

accessing trajectory data:

iterate over all stored configurations:

```
>>> for conf in configurations:
```

iterate over all particles stored in configuration:

```
>>>   for pid in conf
>>>     particle_coords = conf[pid]
>>>     print pid, particle_coords
```

access particle with id <pid> of stored configuration <n>:

```
>>> print "particle coord: ",configurations[n][pid]
```

espressopp.analysis.Configurations(*system*)

Parameters
system –

espressopp.analysis.Configurations.back()

Return type

espressopp.analysis.Configurations.clear()

Return type

espressopp.analysis.Configurations.gather()

Return type

3.1.7 espressopp.analysis.ConfigurationsExt

- *gather()* add configuration to trajectory
- *clear()* clear trajectory
- *back()* get last configuration of trajectory
- *capacity* maximum number of configurations in trajectory further adding (*gather()*) configurations results in erasing oldest configuration before adding new one capacity=0 means: infinite capacity (until memory is full)
- *size* number of stored configurations

usage:

storing trajectory

```
>>> configurations = espressopp.ConfigurationsExt(system)
>>> configurations.gather()
>>> for k in xrange(100):
>>>   integrator.run(100)
>>>   configurations.gather()
```

accessing trajectory data:

iterate over all stored configurations:

```
>>> for conf in configurations:
```

iterate over all particles stored in configuration:

```
>>>   for pid in conf
>>>     particle_coords = conf[pid]
>>>     print pid, particle_coords
```

access particle with id <pid> of stored configuration <n>:

```
>>> print "particle coord: ",configurations[n][pid]
```

espressopp.analysis.ConfigurationsExt(*system*)

Parameters
system –

espressopp.analysis.ConfigurationsExt.back()

Return type

espressopp.analysis.ConfigurationsExt.clear()

Return type

espressopp.analysis.ConfigurationsExt.gather()

Return type

3.1.8 espressopp.analysis.Energy

espressopp.analysis.EnergyPot(*system*, *per_atom*)

Parameters

- system –
- per_atom – (default: False)

espressopp.analysis.EnergyPot.compute()

Return type

espressopp.analysis.EnergyKin(*system*, *per_atom*)

Parameters

- system –
- per_atom – (default: False)

espressopp.analysis.EnergyKin.compute()

Return type

espressopp.analysis.EnergyTot(*system*, *per_atom*)

Parameters

- system –
- per_atom – (default: False)

espressopp.analysis.EnergyTot.compute()

Return type

3.1.9 espressopp.analysis.IntraChainDistSq

espressopp.analysis.IntraChainDistSq(*system*, *fpl*)

Parameters

- system –
- fpl –

espressopp.analysis.IntraChainDistSq.compute()

Return type

3.1.10 espressopp.analysis.MeanSquareDispl

espressopp.analysis.MeanSquareDispl(*system*, *chainlength*)

Parameters

- system –
- chainlength – (default: None)
- start_pid – (default: 0)

espressopp.analysis.MeanSquareDispl.computeG2()

Return type

espressopp.analysis.MeanSquareDispl.computeG3()

Return type

espressopp.analysis.MeanSquareDispl.strange()

Return type

3.1.11 espressopp.analysis.MeanSquareInternalDist

espressopp.analysis.MeanSquareInternalDist(*system*, *chainlength*, *start_pid*)

Parameters

- system –
- chainlength –
- start_pid –

3.1.12 espressopp.analysis.NPartSubregion

Class to compute the number of (coarse-grained) particles in a subregion of the simulation box.

Examples:

```
>>> subregionparticles_instance = espressopp.analysis.NPartSubregion(system, parttype=1, span=0.75,
↪geometry=1, center=[Lx/2, Ly/2, Lz/2])
>>> # creates instance of the class for calculating number of particles of type 1 in a subregion centered
↪in the simulation box and bounded within +-0.75 in x-direction from the center
```

```
>>> number_of_particles = subregionparticles_instance.compute()
>>> # computes the number of particles in subregion of the simulation box
```

espressopp.analysis.NPartSubregion(*system*, *parttype*, *span*, *geometry*, *center*)

Constructs the NPartSubregion object.

Parameters

- system (std::shared_ptr<System>) – system object
- parttype (int) – particle type to be considered for particle number calculation
- span (real) – radius of the subregion to be considered

- geometry (str in ['spherical', 'bounded-x', 'bounded-y', 'bounded-z']) – geometry of the subregion. Can only be in ['spherical', 'bounded-x', 'bounded-y', 'bounded-z']
- center (list of 3 reals (x,y,z coordinates of center)) – center of the subregion

espressopp.analysis.NPartSubregion.compute():

Calculates the number of particles in defined subregion.

Return type
int

class espressopp.analysis.NPartSubregion.NPartSubregionLocal(*system*, *parttype*, *span*, *geometry*, *center*)

The (local) class for computing the number of particles in a subregion of the system.

3.1.13 espressopp.analysis.Observable

Overview

List of classes based on Observable:

espressopp.analysis.AdressDensity

Class to compute radial density profiles in adaptive resolution simulations based on distance to closest AdResS center. Works also for multiple overlapping AdResS regions.

Examples:

```
>>> densityprofile = espressopp.analysis.AdressDensity(system, verletlist)
>>> # creates the class
```

```
>>> densityprofile.addExclusions([1,2,3])
>>> # defines particle to be excluded from the calculation based on list of particle ids
```

```
>>> densityprofile.compute(100)
>>> # computes the densityprofile using 100 bins
```

espressopp.analysis.AdressDensity(*system*, *verletlist*)

Parameters

- system (std::shared_ptr<System>) – system object
- verletlist (std::shared_ptr<VerletListAdress>) – verletlist object

espressopp.analysis.AdressDensity.compute(*bins*)

Parameters
bins (int) – number of bins

Return type
list of reals

espressopp.analysis.AdressDensity.addExclusions(*pidlist*)

Parameters
pidlist (list of ints) – list of ids of particles to be excluded from the calculation

espressopp.analysis.CenterOfMass

espressopp.analysis.CenterOfMass(*system*)

Parameters
system –

espressopp.analysis.MaxPID

espressopp.analysis.MaxPID(*system*)

Parameters
system –

espressopp.analysis.NeighborFluctuation

espressopp.analysis.NeighborFluctuation(*system*, *radius*)

Parameters

- system –
- radius –

espressopp.analysis.NPart

espressopp.analysis.NPart(*system*)

Parameters
system –

espressopp.analysis.PotentialEnergy

The object that computes potential energy of different interactions.

espressopp.analysis.PotentialEnergy(*system*, *potential*, *compute_method=None*)

Parameters

- system (espressopp.System) – The system object
- interaction (espressopp.interaction.Interaction) – The interaction object.
- compute_method (str) – If set to *ALL* (default) then compute total potential energies, if set to *CG* then compute only coarse-grained part (if feasible), if set to *AT* then compute only atomitic part of potential energy.

espressopp.analysis.Pressure

espressopp.analysis.Pressure(*system*)

Parameters
system –

espressopp.analysis.RadialDistrF

espressopp.analysis.RadialDistrF(*system*)

Parameters
system –

espressopp.analysis.RadialDistrF.compute(*rdfN*)

Parameters
rdfN –

Return type

espressopp.analysis.RDFatomistic

Class to compute radial distribution functions in adaptive resolution simulations in subregions of the box. Can be used for regular atomistic/coarse-grained (AT/CG) adaptive resolution simulations as well as path integral-based adaptive resolution simulations. The two functions (compute, computePathIntegral) exhibit different behavior.

The regular compute function is used for regular AT/CG simulations and there are two options:

Option 1 (spanbased = True): the RDF can be calculated in a cuboid region in the center of the box (periodic in y,z, limited in x). In this case, particle pairs are considered for which at least one of them is in the defined cuboid region. This can be useful when the high resolution region has a slab geometry. No further normalization should be required.

Option 2 (spanbased = False): the routine can also calculate unnormalized RDFs using particle pairs with both particles being in the high resolution region (based on the resolution value lambda, the span parameter is not used then). This can be useful when atomistic region has complicated or spherical geometries.

In any case, only pairs of atomistic particles belonging to two different coarse-grained particles are considered. Furthermore, note that the routine uses L_y / half (L_y is the box length in y-direction) as the maximum distance for the RDF calculation, which is then binned according to rdfN during the computation. Hence, L_y should be the shortest box side (or, equally short as L_x and/or L_z).

The computePathIntegral function is used for path integral-based adaptive resolution functions. It calculates the radial distribution functions over pairs of particles between different atoms or coarse-grained beads. Note, however, that in these types of quantum/classical adaptive resolution simulations, regular coarse-grained espressopp particles are associated with each atom and the additional "AdResS" atomistic particles correspond to the different Trotter beads. This means that the routine will, for molecules consisting of multiple atoms, calculate intramolecular rdfs, averaging over the Trotter bead pairs of the ring polymers, which represent the atoms. In doing so, it considers only particles pair with matching Trotter number and with the correct atomistic types. The results are averaged over all Trotter beads. Also in this case L_y / half (L_y is the box length in y-direction) is used as the maximum distance for the RDF calculation, which is then binned according to rdfN during the computation. Furthermore, the calculation is always "spanbased" in x direction (the function ignores the spanbased flag), but in such a fashion that BOTH particles need to be in the defined cuboid region. Normalization is performed as derived in R. Potestio et al., Phys. Rev. Lett. 111, 060601 (2013), Supp. Info. This means that, considering only particles with matching Trotter numbers, the computePathIntegral function

calculates the RDF between particles of type A and B within a region bounded in x-direction by X_{min} and X_{max} as

$$g^{ab}_{slab}(r^{AB}) = \sum_{a \in N^A} \sum_{b \in N^B} \frac{1}{N^A N^B} \frac{\delta_\Delta(|\mathbf{r}_a - \mathbf{r}_b| - r)}{v(\mathbf{r}_a)/V_{slab}}$$

$$\delta_\Delta(r) = \begin{cases} 1 & \text{for} \quad r < \Delta \\ 0 & \text{otherwise} \end{cases}$$

$$v(\mathbf{r}_a) = 2\pi\Delta\ r_a(2r_a - h(\mathbf{r}_a))$$

$$h(\mathbf{r}_a) = (r_a - X^+)\theta(r_a - X^+) - (r_a - X^-)\theta(r_a - X^-)$$

$$X^+ = X_{max} - x_a$$

$$X^- = x_a - X_{min}$$

$$\theta(r) = \begin{cases} 1 & \text{for} \quad r > 0 \\ 0 & \text{otherwise} \end{cases}$$

where N^A and N^B are the number of particles of type A and B in the relevant subregion for the RDF calculation and V_{slab} is the total volume of this subregion. Furthermore, r_a denotes the radius of the spherical shell for the RDF calculation around particle a, Δ is the thickness of the shell, and x_a is the x coordinate of particle a. The final result is an average over all imaginary time slices (Trotter numbers).

Examples:

```
>>> rdf_0_1 = espressopp.analysis.RDFatomistic(system = system, type1 = 0, type2 = 1, spanbased
↪= True, span = 1.5)
>>> # creates the class for calculating the RDF between atomistic particles of type 1 and 0 between
↪different molecules,
>>> # At least one of these particles has to be within plus/minus 1.5 from the center of the box in x-
↪direction
```

```
>>> rdf_0_1.compute(100)
>>> # computes the rdf using 100 bins over a distance corresponding to L_y / 2.0
```

espressopp.analysis.RDFatomistic(*system*, *type1*, *type2*, *spanbased*, *span*)

Constructs the RDFatomistic object.

Parameters

- system (std::shared_ptr<System>) – system object
- type1 (int) – type of atom 1
- type2 (int) – type of atom 2
- spanbased (bool) – (default: True) If True, calculates RDFs in a cuboid region of radius span from the center (limited in x, periodic in y,z). If False, calculates RDFs with both particles being in the high resolution region (using lambda resolution values and ignoring span parameter).
- span (real) – (default: 1.0) +/- distance from centre of box in x-direction of the cuboid region used for RDF calculation if spanbased == True. If spanbased == False, this parameter is not used.

espressopp.analysis.RDFatomistic.compute(*rdfN*)

Calculates the atomistic RDF assuming a regular atomistic/coarse-grained adaptive resolution setup.

Parameters

rdfN (int) – number of bins

Return type
list of reals

espressopp.analysis.RDFatomistic.computePathIntegral(*rdfN*)

Calculates the path integral-based RDF averaging over all Trotter bead pairs with the same Trotter bead number between different ring polymers assuming a path integral-based quantum/classical adaptive resolution setup.

Parameters
rdfN (int) – number of bins

Return type
list of reals

espressopp.analysis.StaticStructF

espressopp.analysis.StaticStructF(*system*)

Parameters
system –

espressopp.analysis.StaticStructF.compute(*nqx*, *nqy*, *nqz*, *bin_factor*, *ofile*)

Parameters

- nqx –
- nqy –
- nqz –
- bin_factor –
- ofile – (default: None)

Return type

espressopp.analysis.StaticStructF.computeSingleChain(*nqx*, *nqy*, *nqz*, *bin_factor*, *chainlength*, *ofile*)

Parameters

- nqx –
- nqy –
- nqz –
- bin_factor –
- chainlength –
- ofile – (default: None)

Return type

espressopp.analysis.XDensity

espressopp.analysis.XDensity(*system*)

Parameters
system –

espressopp.analysis.XDensity.compute(*rdfN*)

Parameters
rdfN –

Return type

espressopp.analysis.XPressure

espressopp.analysis.XPressure(*system*)

Parameters
system –

espressopp.analysis.XPressure.compute(*N*)

Parameters
N –

Return type

espressopp.analysis.XTemperature

espressopp.analysis.XTemperature(*system*)

Parameters
system –

espressopp.analysis.XTemperature.compute(*N*)

Parameters
N –

Return type

Details

espressopp.analysis.Observable.compute()

Return type

3.1.14 espressopp.analysis.RadGyrXProfilePI

Class to compute the radius of gyration profile in adaptive path integral-based simulations along slabs in the x-direction of the system for specified particle type.

Examples:

```
>>> gyrationprofile_instance = espressopp.analysis.RadGyrXProfilePI(system=system)
>>> # creates instance of the class for calculating the radius of gyration profile
```

```
>>> gyrationprofile_list = gyrationprofile_instance.compute(bins=100, ntrotter=32, ptype=2)
>>> # computes the radius of gyration profile for particles of type 2 using 100 bins. The system uses 32
↪Trotter beads.
```

espressopp.analysis.RadGyrXProfilePI(*system*)

Constructs the RadGyrXProfilePI object.

Parameters
system (std::shared_ptr<System>) – system object

espressopp.analysis.RadGyrXProfilePI.compute(bins, ntrotter, ptype):

Calculates the radius of gyration profile in x-direction.

Parameters

- bins (int) – number of bins
- ntrotter (int) – number of Trotter beads
- ptype (int) – particle type

Return type
list of reals

class espressopp.analysis.RadGyrXProfilePI.RadGyrXProfilePILocal(*system*)

The (local) class for computing the radius of gyration profile in x-direction of path integral ring polymers.

compute(*(analysis_RadGyrXProfilePI)arg1*, *(object)arg2*, *(object)arg3*, *(object)arg4*) → list :

C++ signature :
boost::python::list compute(espressopp::analysis::RadGyrXProfilePI {lvalue},int,int,int)

3.1.15 espressopp.analysis.SubregionTracking

Class to compute the number of (coarse-grained) particles that belong to a specified particle list and that reside in a specified subregion of the simulation box (when specifying a list of particles that reside in a certain subregion at the beginning of the simulation, the routine can be used, for example, to track how many of these particles still stay in the same region after some simulation time).

Examples:

```
>>> subregiontracking_instance = espressopp.analysis.SubregionTracking(system, span=0.75,
↪geometry=1, pidlist=tracklist, center=[Lx/2, Ly/2, Lz/2])
>>> # creates instance of the class for calculating number of particles that belong to particle id list
↪tracklist and reside in a subregion which is centered in the simulation box and bounded within +-0.75
↪in x-direction from the center
```

```
>>> number_of_particles = subregiontracking_instance.compute()
>>> # computes the number of particles belonging to specified particle id list in specified subregion of
↪the simulation box
```

espressopp.analysis.SubregionTracking(*self*, *system*, *span*, *geometry*, *center*, *pidlist*)

Constructs the SubregionTracking object.

Parameters

- system (std::shared_ptr<System>) – system object
- span (real) – radius of the subregion to be considered
- geometry (str in ['spherical', 'bounded-x', 'bounded-y', 'bounded-z']) – geometry of the subregion. Can only be in ['spherical', 'bounded-x', 'bounded-y', 'bounded-z']
- center (list of 3 reals (x,y,z coordinates of center)) – center of the subregion
- pidlist (list of ints) – list of particle ids of coarse-grained particles that are counted in the specified subregion

espressopp.analysis.SubregionTracking.compute():

Calculates the number of particles that are present in specified subregion and that belong to specified particle id list.

Return type
real

class espressopp.analysis.SubregionTracking.SubregionTrackingLocal(*system*, *span*, *geometry*, *center*, *pidlist*)

The (local) class for computing the number of particles that are present in a specified subregion of the system and that belong to a specified group of particles.

3.1.16 espressopp.analysis.SystemMonitor

SystemMonitor prints and logs to file values obtained from Observables like temperature, pressure or potential energy.

espressopp.analysis.SystemMonitor(*system*, *integrator*, *output*)

Parameters

- system (espressopp.System) – The system object.
- integrator (espressopp.integrator.MDIntegrator) – The MD integrator.
- output (espressopp.analysis.SystemMonitorOutputCSV) – The output object.

espressopp.analysis.SystemMonitor.add_observable(*name*, *observable*, *is_visible*)

The function adds new observable to SystemMonitor.

Parameters

- name (str) – The name of observable
- observable – The observable, eg. espressopp.analysis.PotentialEnergy
- is_visible (bool) – If set to True then values will be print on console.

espressopp.analysis.SystemMonitor.info()

The method print out on console the values of observables.

CSV Output

The output of SystemMonitor to CSV files.

espressopp.analysis.SystemMonitorOutputCSV(*file_name*, *delimiter*)

Parameters

- file_name (str) – The name of CSV file.
- delimiter (str) – The field delimiter, by default it is tabulator.

Example

```
>>> interaction = espressopp.interaction.VerletListLennardJones(verletlist)
>>> interaction.setPotential(type1=0, type2=0,
                             potential=espressopp.interaction.LennardJones(epsilon=1.0, sigma=1.0,
                                                                           cutoff=2.0))
>>> system_monitor_csv = espressopp.analysis.SystemMonitorOutputCSV('out.csv')
>>> system_monitor = espressopp.analysis.SystemMonitor(
    system, integrator, espressopp.analysis.SystemMonitorOutputCSV('out.csv'))
>>> system_monitor.add_observable('pot', espressopp.analysis.PotentialEnergy(system, interaction))
>>> ext_analysis = espressopp.integrator.ExtAnalyze(system_monitor, 10)
>>> integrator.addExtension(ext_analysis)
```

3.1.17 espressopp.analysis.Velocities

espressopp.analysis.Velocities(*system*)

Parameters

system –

espressopp.analysis.Velocities.clear()

Return type

espressopp.analysis.Velocities.gather()

Return type

3.1.18 espressopp.analysis.VelocityAutocorrelation

espressopp.analysis.VelocityAutocorrelation(*system*)

Parameters

system –

3.1.19 espressopp.analysis.Viscosity

espressopp.analysis.Viscosity(*system*)

Parameters
system –

espressopp.analysis.Viscosity.compute(*t0*, *dt*, *T*)

Parameters

- t0 –
- dt –
- T –

Return type

espressopp.analysis.Viscosity.gather()

Return type

3.2 bc

3.2.1 espressopp.bc.BC

Overview

espressopp.bc.OrthorhombicBC(*args, **kwds)
espressopp.bc.SlabBC(*args, **kwds)

Details

This is the abstract base class for all boundary condition objects. It cannot be used directly. All derived classes implement at least the following methods:

class espressopp.bc.BC

getFoldedPosition(*pos*, *imageBox*)

Parameters

- pos –
- imageBox – (default: None)

Return type

getMinimumImageVector(*pos1*, *pos2*)

Parameters

- pos1 –
- pos2 –

Return type

getRandomPos()

Return type

getUnfoldedPosition(*pos*, *imageBox*)

Parameters

- pos –
- imageBox –

Return type

pos, *pos1* and *pos2* are particle coordinates (type: (*float*, *float*, *float*)). *imageBox* (type: (*int*, *int*, *int*)) specifies the

espressopp.bc.OrthorombicBC

Like all boundary condition objects, this class implements all the methods of the base class **BC** , which are described in detail in the documentation of the abstract class **BC**.

The OrthorhombicBC class is responsible for the orthorhombic boundary condition. Currently only periodic boundary conditions are supported.

Example:

```
>>> boxsize = (Lx, Ly, Lz)
>>> bc = espressopp.bc.OrthorhombicBC(rng, boxsize)
```

espressopp.bc.OrthorhombicBC(*rng*, *boxL*)

Parameters

- rng –
- boxL (real) – (default: 1.0)

espressopp.bc.OrthorhombicBC.setBoxL(*boxL*)

Parameters

boxL –

espressopp.bc.SlabBC

Like all boundary condition objects, this class implements all the methods of the base class **BC** , which are described in detail in the documentation of the abstract class **BC**.

The SlabBC class is responsible for a cuboid boundary condition that is periodic in all but the "dir" dimension. Currently, dir is set arbirtrarily to "0" (the x-direction).

Example:

```
>>> boxsize = (Lx, Ly, Lz)
>>> bc = espressopp.bc.SlabBC(rng, boxsize)
```

espressopp.bc.SlabBC(*rng*, *boxL*)

Parameters

- rng –
- boxL (real) – (default: 1.0)

espressopp.bc.SlabBC.setBoxL(*boxL*)

Parameters
boxL –

3.3 check

3.3.1 espressopp.check.System

3.4 esutil

3.4.1 espressopp.esutil.Collectives

locate the node with here=True (e.g. indicating that data of a distributed storage is on the local node). This is a collective SPMD function.

here is a boolean value, which should be True on at most one node. Returns on the controller the number of the node with here=True, or an KeyError exception if no node had the item, i.e. had here=True.

espressopp.esutil.locateItem(*here*)

Parameters
here –

3.4.2 espressopp.esutil.GammaVariate

espressopp.esutil.GammaVariate(*alpha*, *beta*)

Parameters

- alpha –
- beta –

class espressopp.esutil.GammaVariate.GammaVariate(**args*, ***kwds*)
A random gamma variate.

3.4.3 espressopp.esutil.Grid

class espressopp.esutil.Grid.Grid(**args*, ***kwds*)

Grid class

3.4.4 espressopp.esutil.NormalVariate

espressopp.esutil.NormalVariate(*mean*, *sigma*)

Parameters

- mean (real) – (default: 0.0)
- sigma (real) – (default: 1.0)

class espressopp.esutil.NormalVariate.NormalVariate(**args*, ***kwds*)

A random normal variate.

3.4.5 espressopp.esutil.RNG

class espressopp.esutil.RNG.RNG(**args*, ***kwds*)

Random number generator.

3.4.6 espressopp.esutil.UniformOnSphere

class espressopp.esutil.UniformOnSphere.UniformOnSphere(**args*, ***kwds*)

A random variate that generates 3D vectors that are uniformly distributed on a sphere.

3.5 external

3.5.1 espressopp.external.transformations

Homogeneous Transformation Matrices and Quaternions.

A library for calculating 4x4 matrices for translating, rotating, reflecting, scaling, shearing, projecting, orthogonalizing, and superimposing arrays of 3D homogeneous coordinates as well as for converting between rotation matrices, Euler angles, and quaternions. Also includes an Arcball control object and functions to decompose transformation matrices.

Authors

Christoph Gohlke, Laboratory for Fluorescence Dynamics, University of California, Irvine

Version

2011.01.25

Requirements

- Python 2.6 or 3.1
- Numpy 1.5
- transformations.c 2010.04.10 (optional implementation of some functions in C)

Notes

The API is not stable yet and is expected to change between revisions.

This Python code is not optimized for speed. Refer to the transformations.c module for a faster implementation of some functions.

Documentation in HTML format can be generated with epydoc.

Matrices (M) can be inverted using numpy.linalg.inv(M), concatenated using numpy.dot(M0, M1), or used to transform homogeneous coordinates (v) using numpy.dot(M, v) for shape (4, *) "point of arrays", respectively numpy.dot(v, M.T) for shape (*, 4) "array of points".

Use the transpose of transformation matrices for OpenGL glMultMatrixd().

Calculations are carried out with numpy.float64 precision.

Vector, point, quaternion, and matrix function arguments are expected to be "array like", i.e. tuple, list, or numpy arrays.

Return types are numpy arrays unless specified otherwise.

Angles are in radians unless specified otherwise.

Quaternions w+ix+jy+kz are represented as [w, x, y, z].

A triple of Euler angles can be applied/interpreted in 24 ways, which can be specified using a 4 character string or encoded 4-tuple:

Axes 4-string: e.g. 'sxyz' or 'ryxy'

- first character : rotations are applied to 's'tatic or 'r'otating frame
- remaining characters : successive rotation axis 'x', 'y', or 'z'

Axes 4-tuple: e.g. (0, 0, 0, 0) or (1, 1, 1, 1)

- inner axis: code of axis ('x':0, 'y':1, 'z':2) of rightmost matrix.
- parity : even (0) if inner axis 'x' is followed by 'y', 'y' is followed by 'z', or 'z' is followed by 'x'. Otherwise odd (1).
- repetition : first and last axis are same (1) or different (0).
- frame : rotations are applied to static (0) or rotating (1) frame.

References

(1) Matrices and transformations. Ronald Goldman. In "Graphics Gems I", pp 472-475. Morgan Kaufmann, 1990.

(2) More matrices and transformations: shear and pseudo-perspective. Ronald Goldman. In "Graphics Gems II", pp 320-323. Morgan Kaufmann, 1991.

(3) Decomposing a matrix into simple transformations. Spencer Thomas. In "Graphics Gems II", pp 320-323. Morgan Kaufmann, 1991.

(4) Recovering the data from the transformation matrix. Ronald Goldman. In "Graphics Gems II", pp 324-331. Morgan Kaufmann, 1991.

(5) Euler angle conversion. Ken Shoemake. In "Graphics Gems IV", pp 222-229. Morgan Kaufmann, 1994.

(6) Arcball rotation control. Ken Shoemake. In "Graphics Gems IV", pp 175-192. Morgan Kaufmann, 1994.

(7) Representing attitude: Euler angles, unit quaternions, and rotation vectors. James Diebel. 2006.

(8) A discussion of the solution for the best rotation to relate two sets of vectors. W Kabsch. Acta Cryst. 1978. A34, 827-828.

(9) Closed-form solution of absolute orientation using unit quaternions. BKP Horn. J Opt Soc Am A. 1987. 4(4):629-642.

(10) Quaternions. Ken Shoemake. http://www.sfu.ca/~jwa3/cmpt461/files/quatut.pdf

(11) From quaternion to matrix and back. JMP van Waveren. 2005. http://www.intel.com/cd/ids/developer/asmo-na/eng/293748.htm

(12) Uniform random rotations. Ken Shoemake. In "Graphics Gems III", pp 124-132. Morgan Kaufmann, 1992.

(13) Quaternion in molecular modeling. CFF Karney. J Mol Graph Mod, 25(5):595-604

(14) New method for extracting the quaternion from a rotation matrix. Itzhack Y Bar-Itzhack, J Guid Contr Dynam. 2000. 23(6): 1085-1087.

Examples

```
>>> alpha, beta, gamma = 0.123, -1.234, 2.345
>>> origin, xaxis, yaxis, zaxis = (0, 0, 0), (1, 0, 0), (0, 1, 0), (0, 0, 1)
>>> I = identity_matrix()
>>> Rx = rotation_matrix(alpha, xaxis)
>>> Ry = rotation_matrix(beta, yaxis)
>>> Rz = rotation_matrix(gamma, zaxis)
>>> R = concatenate_matrices(Rx, Ry, Rz)
>>> euler = euler_from_matrix(R, 'rxyz')
>>> numpy.allclose([alpha, beta, gamma], euler)
True
>>> Re = euler_matrix(alpha, beta, gamma, 'rxyz')
>>> is_same_transform(R, Re)
True
>>> al, be, ga = euler_from_matrix(Re, 'rxyz')
>>> is_same_transform(Re, euler_matrix(al, be, ga, 'rxyz'))
True
>>> qx = quaternion_about_axis(alpha, xaxis)
>>> qy = quaternion_about_axis(beta, yaxis)
>>> qz = quaternion_about_axis(gamma, zaxis)
>>> q = quaternion_multiply(qx, qy)
>>> q = quaternion_multiply(q, qz)
>>> Rq = quaternion_matrix(q)
>>> is_same_transform(R, Rq)
True
>>> S = scale_matrix(1.23, origin)
>>> T = translation_matrix((1, 2, 3))
>>> Z = shear_matrix(beta, xaxis, origin, zaxis)
>>> R = random_rotation_matrix(numpy.random.rand(3))
>>> M = concatenate_matrices(T, R, Z, S)
>>> scale, shear, angles, trans, persp = decompose_matrix(M)
>>> numpy.allclose(scale, 1.23)
True
>>> numpy.allclose(trans, (1, 2, 3))
True
>>> numpy.allclose(shear, (0, math.tan(beta), 0))
True
>>> is_same_transform(R, euler_matrix(axes='sxyz', *angles))
True
>>> M1 = compose_matrix(scale, shear, angles, trans, persp)
```

(continues on next page)

(continued from previous page)

```
>>> is_same_transform(M, M1)
True
>>> v0, v1 = random_vector(3), random_vector(3)
>>> M = rotation_matrix(angle_between_vectors(v0, v1), vector_product(v0, v1))
>>> v2 = numpy.dot(v0, M[:3,:3].T)
>>> numpy.allclose(unit_vector(v1), unit_vector(v2))
True
```

class espressopp.external.transformations.Arcball(*initial=None*)

Virtual Trackball Control.

```
>>> ball = Arcball()
>>> ball = Arcball(initial=numpy.identity(4))
>>> ball.place([320, 320], 320)
>>> ball.down([500, 250])
>>> ball.drag([475, 275])
>>> R = ball.matrix()
>>> numpy.allclose(numpy.sum(R), 3.90583455)
True
>>> ball = Arcball(initial=[1, 0, 0, 0])
>>> ball.place([320, 320], 320)
>>> ball.setaxes([1,1,0], [-1, 1, 0])
>>> ball.setconstrain(True)
>>> ball.down([400, 200])
>>> ball.drag([200, 400])
>>> R = ball.matrix()
>>> numpy.allclose(numpy.sum(R), 0.2055924)
True
>>> ball.next()
```

down(*point*)

Set initial cursor window coordinates and pick constrain-axis.

drag(*point*)

Update current cursor window coordinates.

getconstrain()

Return state of constrain to axis mode.

matrix()

Return homogeneous rotation matrix.

next(*acceleration=0.0*)

Continue rotation in direction of last drag.

place(*center*, *radius*)

Place Arcball, e.g. when window size changes.

center

[sequence[2]] Window coordinates of trackball center.

radius

[float] Radius of trackball in window coordinates.

setaxes(**axes*)

Set axes to constrain rotations.

setconstrain(*constrain*)

Set state of constrain to axis mode.

espressopp.external.transformations.angle_between_vectors(*v0*, *v1*, *directed=True*, *axis=0*)

Return angle between vectors.

If directed is False, the input vectors are interpreted as undirected axes, i.e. the maximum angle is pi/2.

```
>>> a = angle_between_vectors([1, -2, 3], [-1, 2, -3])
>>> numpy.allclose(a, math.pi)
True
>>> a = angle_between_vectors([1, -2, 3], [-1, 2, -3], directed=False)
>>> numpy.allclose(a, 0)
True
>>> v0 = [[2, 0, 0, 2], [0, 2, 0, 2], [0, 0, 2, 2]]
>>> v1 = [[3], [0], [0]]
>>> a = angle_between_vectors(v0, v1)
>>> numpy.allclose(a, [0., 1.5708, 1.5708, 0.95532])
True
>>> v0 = [[2, 0, 0], [2, 0, 0], [0, 2, 0], [2, 0, 0]]
>>> v1 = [[0, 3, 0], [0, 0, 3], [0, 0, 3], [3, 3, 3]]
>>> a = angle_between_vectors(v0, v1, axis=1)
>>> numpy.allclose(a, [1.5708, 1.5708, 1.5708, 0.95532])
True
```

espressopp.external.transformations.arcball_constrain_to_axis(*point*, *axis*)

Return sphere point perpendicular to axis.

espressopp.external.transformations.arcball_map_to_sphere(*point*, *center*, *radius*)

Return unit sphere coordinates from window coordinates.

espressopp.external.transformations.arcball_nearest_axis(*point*, *axes*)

Return axis, which arc is nearest to point.

espressopp.external.transformations.clip_matrix(*left*, *right*, *bottom*, *top*, *near*, *far*, *perspective=False*)

Return matrix to obtain normalized device coordinates from frustrum.

The frustrum bounds are axis-aligned along x (left, right), y (bottom, top) and z (near, far).

Normalized device coordinates are in range [-1, 1] if coordinates are inside the frustrum.

If perspective is True the frustrum is a truncated pyramid with the perspective point at origin and direction along z axis, otherwise an orthographic canonical view volume (a box).

Homogeneous coordinates transformed by the perspective clip matrix need to be dehomogenized (devided by w coordinate).

```
>>> frustrum = numpy.random.rand(6)
>>> frustrum[1] += frustrum[0]
>>> frustrum[3] += frustrum[2]
>>> frustrum[5] += frustrum[4]
>>> M = clip_matrix(perspective=False, *frustrum)
>>> numpy.dot(M, [frustrum[0], frustrum[2], frustrum[4], 1.0])
array([-1., -1., -1.,  1.])
```

(continues on next page)

(continued from previous page)

```
>>> numpy.dot(M, [frustrum[1], frustrum[3], frustrum[5], 1.0])
array([ 1.,  1.,  1.,  1.])
>>> M = clip_matrix(perspective=True, *frustrum)
>>> v = numpy.dot(M, [frustrum[0], frustrum[2], frustrum[4], 1.0])
>>> v / v[3]
array([-1., -1., -1.,  1.])
>>> v = numpy.dot(M, [frustrum[1], frustrum[3], frustrum[4], 1.0])
>>> v / v[3]
array([ 1.,  1., -1.,  1.])
```

espressopp.external.transformations.compose_matrix(*scale=None*, *shear=None*, *angles=None*, *translate=None*, *perspective=None*)

Return transformation matrix from sequence of transformations.

This is the inverse of the decompose_matrix function.

Sequence of transformations:

scale : vector of 3 scaling factors shear : list of shear factors for x-y, x-z, y-z axes angles : list of Euler angles about static x, y, z axes translate : translation vector along x, y, z axes perspective : perspective partition of matrix

```
>>> scale = numpy.random.random(3) - 0.5
>>> shear = numpy.random.random(3) - 0.5
>>> angles = (numpy.random.random(3) - 0.5) * (2*math.pi)
>>> trans = numpy.random.random(3) - 0.5
>>> persp = numpy.random.random(4) - 0.5
>>> M0 = compose_matrix(scale, shear, angles, trans, persp)
>>> result = decompose_matrix(M0)
>>> M1 = compose_matrix(*result)
>>> is_same_transform(M0, M1)
True
```

espressopp.external.transformations.concatenate_matrices(**matrices*)

Return concatenation of series of transformation matrices.

```
>>> M = numpy.random.rand(16).reshape((4, 4)) - 0.5
>>> numpy.allclose(M, concatenate_matrices(M))
True
>>> numpy.allclose(numpy.dot(M, M.T), concatenate_matrices(M, M.T))
True
```

espressopp.external.transformations.decompose_matrix(*matrix*)

Return sequence of transformations from transformation matrix.

matrix

[array_like] Non-degenerative homogeneous transformation matrix

Return tuple of:

scale : vector of 3 scaling factors shear : list of shear factors for x-y, x-z, y-z axes angles : list of Euler angles about static x, y, z axes translate : translation vector along x, y, z axes perspective : perspective partition of matrix

Raise ValueError if matrix is of wrong type or degenerative.

```
>>> T0 = translation_matrix((1, 2, 3))
>>> scale, shear, angles, trans, persp = decompose_matrix(T0)
>>> T1 = translation_matrix(trans)
>>> numpy.allclose(T0, T1)
True
>>> S = scale_matrix(0.123)
>>> scale, shear, angles, trans, persp = decompose_matrix(S)
>>> scale[0]
0.123
>>> R0 = euler_matrix(1, 2, 3)
>>> scale, shear, angles, trans, persp = decompose_matrix(R0)
>>> R1 = euler_matrix(*angles)
>>> numpy.allclose(R0, R1)
True
```

espressopp.external.transformations.euler_from_matrix(*matrix*, *axes='sxyz'*)

Return Euler angles from rotation matrix for specified axis sequence.

axes : One of 24 axis sequences as string or encoded tuple

Note that many Euler angle triplets can describe one matrix.

```
>>> R0 = euler_matrix(1, 2, 3, 'syxz')
>>> al, be, ga = euler_from_matrix(R0, 'syxz')
>>> R1 = euler_matrix(al, be, ga, 'syxz')
>>> numpy.allclose(R0, R1)
True
>>> angles = (4.0*math.pi) * (numpy.random.random(3) - 0.5)
>>> for axes in _AXES2TUPLE.keys():
...    R0 = euler_matrix(axes=axes, *angles)
...    R1 = euler_matrix(axes=axes, *euler_from_matrix(R0, axes))
...    if not numpy.allclose(R0, R1): print(axes, "failed")
```

espressopp.external.transformations.euler_from_quaternion(*quaternion*, *axes='sxyz'*)

Return Euler angles from quaternion for specified axis sequence.

```
>>> angles = euler_from_quaternion([0.99810947, 0.06146124, 0, 0])
>>> numpy.allclose(angles, [0.123, 0, 0])
True
```

espressopp.external.transformations.euler_matrix(*ai*, *aj*, *ak*, *axes='sxyz'*)

Return homogeneous rotation matrix from Euler angles and axis sequence.

ai, aj, ak : Euler's roll, pitch and yaw angles axes : One of 24 axis sequences as string or encoded tuple

```
>>> R = euler_matrix(1, 2, 3, 'syxz')
>>> numpy.allclose(numpy.sum(R[0]), -1.34786452)
True
>>> R = euler_matrix(1, 2, 3, (0, 1, 0, 1))
>>> numpy.allclose(numpy.sum(R[0]), -0.383436184)
True
>>> ai, aj, ak = (4.0*math.pi) * (numpy.random.random(3) - 0.5)
>>> for axes in _AXES2TUPLE.keys():
...    R = euler_matrix(ai, aj, ak, axes)
```

(continues on next page)

(continued from previous page)

```
>>> for axes in _TUPLE2AXES.keys():
...     R = euler_matrix(ai, aj, ak, axes)
```

espressopp.external.transformations.identity_matrix()

Return 4x4 identity/unit matrix.

```
>>> I = identity_matrix()
>>> numpy.allclose(I, numpy.dot(I, I))
True
>>> numpy.sum(I), numpy.trace(I)
(4.0, 4.0)
>>> numpy.allclose(I, numpy.identity(4, dtype=numpy.float64))
True
```

espressopp.external.transformations.inverse_matrix(*matrix*)

Return inverse of square transformation matrix.

```
>>> M0 = random_rotation_matrix()
>>> M1 = inverse_matrix(M0.T)
>>> numpy.allclose(M1, numpy.linalg.inv(M0.T))
True
>>> for size in xrange(1, 7):
...     M0 = numpy.random.rand(size, size)
...     M1 = inverse_matrix(M0)
...     if not numpy.allclose(M1, numpy.linalg.inv(M0)): print(size)
```

espressopp.external.transformations.is_same_transform(*matrix0*, *matrix1*)

Return True if two matrices perform same transformation.

```
>>> is_same_transform(numpy.identity(4), numpy.identity(4))
True
>>> is_same_transform(numpy.identity(4), random_rotation_matrix())
False
```

espressopp.external.transformations.orthogonalization_matrix(*lengths*, *angles*)

Return orthogonalization matrix for crystallographic cell coordinates.

Angles are expected in degrees.

The de-orthogonalization matrix is the inverse.

```
>>> O = orthogonalization_matrix((10., 10., 10.), (90., 90., 90.))
>>> numpy.allclose(O[:3, :3], numpy.identity(3, float) * 10)
True
>>> O = orthogonalization_matrix([9.8, 12.0, 15.5], [87.2, 80.7, 69.7])
>>> numpy.allclose(numpy.sum(O), 43.063229)
True
```

espressopp.external.transformations.projection_from_matrix(*matrix*, *pseudo=False*)

Return projection plane and perspective point from projection matrix.

Return values are same as arguments for projection_matrix function: point, normal, direction, perspective, and pseudo.

```
>>> point = numpy.random.random(3) - 0.5
>>> normal = numpy.random.random(3) - 0.5
>>> direct = numpy.random.random(3) - 0.5
>>> persp = numpy.random.random(3) - 0.5
>>> P0 = projection_matrix(point, normal)
>>> result = projection_from_matrix(P0)
>>> P1 = projection_matrix(*result)
>>> is_same_transform(P0, P1)
True
>>> P0 = projection_matrix(point, normal, direct)
>>> result = projection_from_matrix(P0)
>>> P1 = projection_matrix(*result)
>>> is_same_transform(P0, P1)
True
>>> P0 = projection_matrix(point, normal, perspective=persp, pseudo=False)
>>> result = projection_from_matrix(P0, pseudo=False)
>>> P1 = projection_matrix(*result)
>>> is_same_transform(P0, P1)
True
>>> P0 = projection_matrix(point, normal, perspective=persp, pseudo=True)
>>> result = projection_from_matrix(P0, pseudo=True)
>>> P1 = projection_matrix(*result)
>>> is_same_transform(P0, P1)
True
```

espressopp.external.transformations.projection_matrix(*point*, *normal*, *direction=None*, *perspective=None*, *pseudo=False*)

Return matrix to project onto plane defined by point and normal.

Using either perspective point, projection direction, or none of both.

If pseudo is True, perspective projections will preserve relative depth such that Perspective = dot(Orthogonal, PseudoPerspective).

```
>>> P = projection_matrix((0, 0, 0), (1, 0, 0))
>>> numpy.allclose(P[1:, 1:], numpy.identity(4)[1:, 1:])
True
>>> point = numpy.random.random(3) - 0.5
>>> normal = numpy.random.random(3) - 0.5
>>> direct = numpy.random.random(3) - 0.5
>>> persp = numpy.random.random(3) - 0.5
>>> P0 = projection_matrix(point, normal)
>>> P1 = projection_matrix(point, normal, direction=direct)
>>> P2 = projection_matrix(point, normal, perspective=persp)
>>> P3 = projection_matrix(point, normal, perspective=persp, pseudo=True)
>>> is_same_transform(P2, numpy.dot(P0, P3))
True
>>> P = projection_matrix((3, 0, 0), (1, 1, 0), (1, 0, 0))
>>> v0 = (numpy.random.rand(4, 5) - 0.5) * 20.0
>>> v0[3] = 1.0
>>> v1 = numpy.dot(P, v0)
>>> numpy.allclose(v1[1], v0[1])
True
```

(continues on next page)

(continued from previous page)

```
>>> numpy.allclose(v1[0], 3.0-v1[1])
True
```

espressopp.external.transformations.quaternion_about_axis(*angle*, *axis*)

Return quaternion for rotation about axis.

```
>>> q = quaternion_about_axis(0.123, (1, 0, 0))
>>> numpy.allclose(q, [0.99810947, 0.06146124, 0, 0])
True
```

espressopp.external.transformations.quaternion_conjugate(*quaternion*)

Return conjugate of quaternion.

```
>>> q0 = random_quaternion()
>>> q1 = quaternion_conjugate(q0)
>>> q1[0] == q0[0] and all(q1[1:] == -q0[1:])
True
```

espressopp.external.transformations.quaternion_from_euler(*ai*, *aj*, *ak*, *axes='sxyz'*)

Return quaternion from Euler angles and axis sequence.

ai, aj, ak : Euler's roll, pitch and yaw angles axes : One of 24 axis sequences as string or encoded tuple

```
>>> q = quaternion_from_euler(1, 2, 3, 'ryxz')
>>> numpy.allclose(q, [0.435953, 0.310622, -0.718287, 0.444435])
True
```

espressopp.external.transformations.quaternion_from_matrix(*matrix*, *isprecise=False*)

Return quaternion from rotation matrix.

If isprecise=True, the input matrix is assumed to be a precise rotation matrix and a faster algorithm is used.

```
>>> q = quaternion_from_matrix(identity_matrix(), True)
>>> numpy.allclose(q, [1., 0., 0., 0.])
True
>>> q = quaternion_from_matrix(numpy.diag([1., -1., -1., 1.]))
>>> numpy.allclose(q, [0, 1, 0, 0]) or numpy.allclose(q, [0, -1, 0, 0])
True
>>> R = rotation_matrix(0.123, (1, 2, 3))
>>> q = quaternion_from_matrix(R, True)
>>> numpy.allclose(q, [0.9981095, 0.0164262, 0.0328524, 0.0492786])
True
>>> R = [[-0.545, 0.797, 0.260, 0], [0.733, 0.603, -0.313, 0],
...      [-0.407, 0.021, -0.913, 0], [0, 0, 0, 1]]
>>> q = quaternion_from_matrix(R)
>>> numpy.allclose(q, [0.19069, 0.43736, 0.87485, -0.083611])
True
>>> R = [[0.395, 0.362, 0.843, 0], [-0.626, 0.796, -0.056, 0],
...      [-0.677, -0.498, 0.529, 0], [0, 0, 0, 1]]
>>> q = quaternion_from_matrix(R)
>>> numpy.allclose(q, [0.82336615, -0.13610694, 0.46344705, -0.29792603])
True
>>> R = random_rotation_matrix()
```

(continues on next page)

(continued from previous page)

```
>>> q = quaternion_from_matrix(R)
>>> is_same_transform(R, quaternion_matrix(q))
True
```

espressopp.external.transformations.quaternion_imag(*quaternion*)

Return imaginary part of quaternion.

```
>>> quaternion_imag([3.0, 0.0, 1.0, 2.0])
[0.0, 1.0, 2.0]
```

espressopp.external.transformations.quaternion_inverse(*quaternion*)

Return inverse of quaternion.

```
>>> q0 = random_quaternion()
>>> q1 = quaternion_inverse(q0)
>>> numpy.allclose(quaternion_multiply(q0, q1), [1, 0, 0, 0])
True
```

espressopp.external.transformations.quaternion_matrix(*quaternion*)

Return homogeneous rotation matrix from quaternion.

```
>>> M = quaternion_matrix([0.99810947, 0.06146124, 0, 0])
>>> numpy.allclose(M, rotation_matrix(0.123, (1, 0, 0)))
True
>>> M = quaternion_matrix([1, 0, 0, 0])
>>> numpy.allclose(M, identity_matrix())
True
>>> M = quaternion_matrix([0, 1, 0, 0])
>>> numpy.allclose(M, numpy.diag([1, -1, -1, 1]))
True
```

espressopp.external.transformations.quaternion_multiply(*quaternion1*, *quaternion0*)

Return multiplication of two quaternions.

```
>>> q = quaternion_multiply([4, 1, -2, 3], [8, -5, 6, 7])
>>> numpy.allclose(q, [28, -44, -14, 48])
True
```

espressopp.external.transformations.quaternion_real(*quaternion*)

Return real part of quaternion.

```
>>> quaternion_real([3.0, 0.0, 1.0, 2.0])
3.0
```

espressopp.external.transformations.quaternion_slerp(*quat0*, *quat1*, *fraction*, *spin=0*, *shortestpath=True*)

Return spherical linear interpolation between two quaternions.

```
>>> q0 = random_quaternion()
>>> q1 = random_quaternion()
>>> q = quaternion_slerp(q0, q1, 0.0)
>>> numpy.allclose(q, q0)
True
```

(continues on next page)

(continued from previous page)

```
>>> q = quaternion_slerp(q0, q1, 1.0, 1)
>>> numpy.allclose(q, q1)
True
>>> q = quaternion_slerp(q0, q1, 0.5)
>>> angle = math.acos(numpy.dot(q0, q))
>>> numpy.allclose(2.0, math.acos(numpy.dot(q0, q1)) / angle) or        numpy.allclose(2.0, math.
↪acos(-numpy.dot(q0, q1)) / angle)
True
```

espressopp.external.transformations.random_quaternion(*rand=None*)

Return uniform random unit quaternion.

rand: array like or None
Three independent random variables that are uniformly distributed between 0 and 1.

```
>>> q = random_quaternion()
>>> numpy.allclose(1.0, vector_norm(q))
True
>>> q = random_quaternion(numpy.random.random(3))
>>> len(q.shape), q.shape[0]==4
(1, True)
```

espressopp.external.transformations.random_rotation_matrix(*rand=None*)

Return uniform random rotation matrix.

rnd: array like
Three independent random variables that are uniformly distributed between 0 and 1 for each returned quaternion.

```
>>> R = random_rotation_matrix()
>>> numpy.allclose(numpy.dot(R.T, R), numpy.identity(4))
True
```

espressopp.external.transformations.random_vector(*size*)

Return array of random doubles in the half-open interval [0.0, 1.0).

```
>>> v = random_vector(10000)
>>> numpy.all(v >= 0.0) and numpy.all(v < 1.0)
True
>>> v0 = random_vector(10)
>>> v1 = random_vector(10)
>>> numpy.any(v0 == v1)
False
```

espressopp.external.transformations.reflection_from_matrix(*matrix*)

Return mirror plane point and normal vector from reflection matrix.

```
>>> v0 = numpy.random.random(3) - 0.5
>>> v1 = numpy.random.random(3) - 0.5
>>> M0 = reflection_matrix(v0, v1)
>>> point, normal = reflection_from_matrix(M0)
>>> M1 = reflection_matrix(point, normal)
```

(continues on next page)

(continued from previous page)

```
>>> is_same_transform(M0, M1)
True
```

espressopp.external.transformations.reflection_matrix(*point*, *normal*)

Return matrix to mirror at plane defined by point and normal vector.

```
>>> v0 = numpy.random.random(4) - 0.5
>>> v0[3] = 1.0
>>> v1 = numpy.random.random(3) - 0.5
>>> R = reflection_matrix(v0, v1)
>>> numpy.allclose(2., numpy.trace(R))
True
>>> numpy.allclose(v0, numpy.dot(R, v0))
True
>>> v2 = v0.copy()
>>> v2[:3] += v1
>>> v3 = v0.copy()
>>> v2[:3] -= v1
>>> numpy.allclose(v2, numpy.dot(R, v3))
True
```

espressopp.external.transformations.rotation_from_matrix(*matrix*)

Return rotation angle and axis from rotation matrix.

```
>>> angle = (random.random() - 0.5) * (2*math.pi)
>>> direc = numpy.random.random(3) - 0.5
>>> point = numpy.random.random(3) - 0.5
>>> R0 = rotation_matrix(angle, direc, point)
>>> angle, direc, point = rotation_from_matrix(R0)
>>> R1 = rotation_matrix(angle, direc, point)
>>> is_same_transform(R0, R1)
True
```

espressopp.external.transformations.rotation_matrix(*angle*, *direction*, *point=None*)

Return matrix to rotate about axis defined by point and direction.

```
>>> R = rotation_matrix(math.pi/2.0, [0, 0, 1], [1, 0, 0])
>>> numpy.allclose(numpy.dot(R, [0, 0, 0, 1]), [ 1., -1.,  0.,  1.])
True
>>> angle = (random.random() - 0.5) * (2*math.pi)
>>> direc = numpy.random.random(3) - 0.5
>>> point = numpy.random.random(3) - 0.5
>>> R0 = rotation_matrix(angle, direc, point)
>>> R1 = rotation_matrix(angle-2*math.pi, direc, point)
>>> is_same_transform(R0, R1)
True
>>> R0 = rotation_matrix(angle, direc, point)
>>> R1 = rotation_matrix(-angle, -direc, point)
>>> is_same_transform(R0, R1)
True
>>> I = numpy.identity(4, numpy.float64)
>>> numpy.allclose(I, rotation_matrix(math.pi*2, direc))
```

(continues on next page)

(continued from previous page)

```
True
>>> numpy.allclose(2., numpy.trace(rotation_matrix(math.pi/2,
...                                                direc, point)))
True
```

espressopp.external.transformations.scale_from_matrix(*matrix*)

Return scaling factor, origin and direction from scaling matrix.

```
>>> factor = random.random() * 10 - 5
>>> origin = numpy.random.random(3) - 0.5
>>> direct = numpy.random.random(3) - 0.5
>>> S0 = scale_matrix(factor, origin)
>>> factor, origin, direction = scale_from_matrix(S0)
>>> S1 = scale_matrix(factor, origin, direction)
>>> is_same_transform(S0, S1)
True
>>> S0 = scale_matrix(factor, origin, direct)
>>> factor, origin, direction = scale_from_matrix(S0)
>>> S1 = scale_matrix(factor, origin, direction)
>>> is_same_transform(S0, S1)
True
```

espressopp.external.transformations.scale_matrix(*factor*, *origin=None*, *direction=None*)

Return matrix to scale by factor around origin in direction.

Use factor -1 for point symmetry.

```
>>> v = (numpy.random.rand(4, 5) - 0.5) * 20.0
>>> v[3] = 1.0
>>> S = scale_matrix(-1.234)
>>> numpy.allclose(numpy.dot(S, v)[:3], -1.234*v[:3])
True
>>> factor = random.random() * 10 - 5
>>> origin = numpy.random.random(3) - 0.5
>>> direct = numpy.random.random(3) - 0.5
>>> S = scale_matrix(factor, origin)
>>> S = scale_matrix(factor, origin, direct)
```

espressopp.external.transformations.shear_from_matrix(*matrix*)

Return shear angle, direction and plane from shear matrix.

```
>>> angle = (random.random() - 0.5) * 4*math.pi
>>> direct = numpy.random.random(3) - 0.5
>>> point = numpy.random.random(3) - 0.5
>>> normal = numpy.cross(direct, numpy.random.random(3))
>>> S0 = shear_matrix(angle, direct, point, normal)
>>> angle, direct, point, normal = shear_from_matrix(S0)
>>> S1 = shear_matrix(angle, direct, point, normal)
>>> is_same_transform(S0, S1)
True
```

espressopp.external.transformations.shear_matrix(*angle*, *direction*, *point*, *normal*)

Return matrix to shear by angle along direction vector on shear plane.

The shear plane is defined by a point and normal vector. The direction vector must be orthogonal to the plane's normal vector.

A point P is transformed by the shear matrix into P" such that the vector P-P" is parallel to the direction vector and its extent is given by the angle of P-P'-P", where P' is the orthogonal projection of P onto the shear plane.

```
>>> angle = (random.random() - 0.5) * 4*math.pi
>>> direct = numpy.random.random(3) - 0.5
>>> point = numpy.random.random(3) - 0.5
>>> normal = numpy.cross(direct, numpy.random.random(3))
>>> S = shear_matrix(angle, direct, point, normal)
>>> numpy.allclose(1.0, numpy.linalg.det(S))
True
```

espressopp.external.transformations.superimposition_matrix(*v0*, *v1*, *scaling=False*, *usesvd=True*)

Return matrix to transform given vector set into second vector set.

v0 and v1 are shape (3, *) or (4, *) arrays of at least 3 vectors.

If usesvd is True, the weighted sum of squared deviations (RMSD) is minimized according to the algorithm by W. Kabsch [8]. Otherwise the quaternion based algorithm by B. Horn [9] is used (slower when using this Python implementation).

The returned matrix performs rotation, translation and uniform scaling (if specified).

```
>>> v0 = numpy.random.rand(3, 10)
>>> M = superimposition_matrix(v0, v0)
>>> numpy.allclose(M, numpy.identity(4))
True
>>> R = random_rotation_matrix(numpy.random.random(3))
>>> v0 = ((1,0,0), (0,1,0), (0,0,1), (1,1,1))
>>> v1 = numpy.dot(R, v0)
>>> M = superimposition_matrix(v0, v1)
>>> numpy.allclose(v1, numpy.dot(M, v0))
True
>>> v0 = (numpy.random.rand(4, 100) - 0.5) * 20.0
>>> v0[3] = 1.0
>>> v1 = numpy.dot(R, v0)
>>> M = superimposition_matrix(v0, v1)
>>> numpy.allclose(v1, numpy.dot(M, v0))
True
>>> S = scale_matrix(random.random())
>>> T = translation_matrix(numpy.random.random(3)-0.5)
>>> M = concatenate_matrices(T, R, S)
>>> v1 = numpy.dot(M, v0)
>>> v0[:3] += numpy.random.normal(0.0, 1e-9, 300).reshape(3, -1)
>>> M = superimposition_matrix(v0, v1, scaling=True)
>>> numpy.allclose(v1, numpy.dot(M, v0))
True
>>> M = superimposition_matrix(v0, v1, scaling=True, usesvd=False)
>>> numpy.allclose(v1, numpy.dot(M, v0))
True
>>> v = numpy.empty((4, 100, 3), dtype=numpy.float64)
>>> v[:, :, 0] = v0
>>> M = superimposition_matrix(v0, v1, scaling=True, usesvd=False)
```

(continues on next page)

(continued from previous page)

```
>>> numpy.allclose(v1, numpy.dot(M, v[:, :, 0]))
True
```

espressopp.external.transformations.translation_from_matrix(*matrix*)

Return translation vector from translation matrix.

```
>>> v0 = numpy.random.random(3) - 0.5
>>> v1 = translation_from_matrix(translation_matrix(v0))
>>> numpy.allclose(v0, v1)
True
```

espressopp.external.transformations.translation_matrix(*direction*)

Return matrix to translate by direction vector.

```
>>> v = numpy.random.random(3) - 0.5
>>> numpy.allclose(v, translation_matrix(v)[:3, 3])
True
```

espressopp.external.transformations.unit_vector(*data*, *axis=None*, *out=None*)

Return ndarray normalized by length, i.e. eucledian norm, along axis.

```
>>> v0 = numpy.random.random(3)
>>> v1 = unit_vector(v0)
>>> numpy.allclose(v1, v0 / numpy.linalg.norm(v0))
True
>>> v0 = numpy.random.rand(5, 4, 3)
>>> v1 = unit_vector(v0, axis=-1)
>>> v2 = v0 / numpy.expand_dims(numpy.sqrt(numpy.sum(v0*v0, axis=2)), 2)
>>> numpy.allclose(v1, v2)
True
>>> v1 = unit_vector(v0, axis=1)
>>> v2 = v0 / numpy.expand_dims(numpy.sqrt(numpy.sum(v0*v0, axis=1)), 1)
>>> numpy.allclose(v1, v2)
True
>>> v1 = numpy.empty((5, 4, 3), dtype=numpy.float64)
>>> unit_vector(v0, axis=1, out=v1)
>>> numpy.allclose(v1, v2)
True
>>> list(unit_vector([]))
[]
>>> list(unit_vector([1.0]))
[1.0]
```

espressopp.external.transformations.vector_norm(*data*, *axis=None*, *out=None*)

Return length, i.e. eucledian norm, of ndarray along axis.

```
>>> v = numpy.random.random(3)
>>> n = vector_norm(v)
>>> numpy.allclose(n, numpy.linalg.norm(v))
True
>>> v = numpy.random.rand(6, 5, 3)
>>> n = vector_norm(v, axis=-1)
```

(continues on next page)

(continued from previous page)

```
>>> numpy.allclose(n, numpy.sqrt(numpy.sum(v*v, axis=2)))
True
>>> n = vector_norm(v, axis=1)
>>> numpy.allclose(n, numpy.sqrt(numpy.sum(v*v, axis=1)))
True
>>> v = numpy.random.rand(5, 4, 3)
>>> n = numpy.empty((5, 3), dtype=numpy.float64)
>>> vector_norm(v, axis=1, out=n)
>>> numpy.allclose(n, numpy.sqrt(numpy.sum(v*v, axis=1)))
True
>>> vector_norm([])
0.0
>>> vector_norm([1.0])
1.0
```

espressopp.external.transformations.vector_product(*v0*, *v1*, *axis=0*)

Return vector perpendicular to vectors.

```
>>> v = vector_product([2, 0, 0], [0, 3, 0])
>>> numpy.allclose(v, [0, 0, 6])
True
>>> v0 = [[2, 0, 0, 2], [0, 2, 0, 2], [0, 0, 2, 2]]
>>> v1 = [[3], [0], [0]]
>>> v = vector_product(v0, v1)
>>> numpy.allclose(v, [[0, 0, 0, 0], [0, 0, 6, 6], [0, -6, 0, -6]])
True
>>> v0 = [[2, 0, 0], [2, 0, 0], [0, 2, 0], [2, 0, 0]]
>>> v1 = [[0, 3, 0], [0, 0, 3], [0, 0, 3], [3, 3, 3]]
>>> v = vector_product(v0, v1, axis=1)
>>> numpy.allclose(v, [[0, 0, 6], [0, -6, 0], [6, 0, 0], [0, -6, 6]])
True
```

3.6 integrator

3.6.1 espressopp.integrator.Adress

The AdResS object is an extension to the integrator. It makes sure that the integrator also processes the atomistic particles and not only the CG particles. Hence, this object is of course only used when performing AdResS or H-AdResS simulations.

In detail the AdResS extension makes sure:

- that also the forces on the atomistic particles are initialized and set to by Adress::initForces
- that also the atomistic particles are integrated and propagated by Adress::integrate1 and Adress::integrate2

Example - how to turn on the AdResS integrator extension:

```
>>> adress = espressopp.integrator.Adress(system, verletlist, fixedtuplelist)
>>> integrator.addExtension(adress)
```

If KTI is set to True, then the resolution parameters are not updated. This can be used for example for Kirkwood thermodynamic integration, during which one manually sets the whole system on different resolution parameters. KTI

= True then prevents overwriting these manually set values. Furthermore, when having moving AdResS regions based on particles, regionupdates specifies the update frequency of the AdResS region in number of steps (or, to be more precise, calls of communicateAdrPositions()). Note that there is a tradeoff: The more frequently the AdResS region is updated, the more gradually and accurately the AdResS region changes and adapts it shape. This could allow for a smaller overall AdResS region and possibly a smoother simulation. However, when having many AdResS region defining particles, these frequent updates can become computationally significant and cost additional simulation time. The optimum is highly system and application dependent.

Finally, when making use of a RESPA Velocity Verlet integrator, then the multistep parameter defines after how many steps of the inner integration loop the slow forces are updated. It should be set consistently with the same parameter in VelocityVerletRESPA.

class espressopp.integrator.Adress(*_system*, *_verletlist*, *_fixedtuplelist*, *KTI*, *regionupdates*, *multistep*)

Parameters

- _system (std::shared_ptr<System>) – system object
- _verletlist (std::shared_ptr<VerletListAdress>) – verletlist object
- _fixedtuplelist (std::shared_ptr<FixedTupleListAdress>) – fixedtuplelist object
- KTI (bool) – (default: False) update resolution parameter? (Yes: set False, No: set True)
- regionupdates (int) – (default: 1) after how many steps does the AdResS region needs to be updated?
- multistep (int) – (default: 1) when used with VelocityVerletRESPA (otherwise, ignored), after how many steps of the inner integration loop do we update the slow forces? This parameter should be set consistently with multistep in VelocityVerletRESPA.

3.6.2 espressopp.integrator.AssociationReaction

3.6.3 espressopp.integrator.BerendsenBarostat

This is the Berendsen barostat implementation according to the original paper [Berendsen84]. If Berendsen barostat is defined (as a property of integrator) then at the each run the system size and the particle coordinates will be scaled by scaling parameter μ according to the formula:

$$\mu = [1 - \Delta t/\tau(P_0 - P)]^{1/3}$$

where Δt - integration timestep, τ - time parameter (coupling parameter), P_0 - external pressure and P - instantaneous pressure.

Example:

```
>>> berendsenP = espressopp.integrator.BerendsenBarostat(system)
>>> berendsenP.tau = 0.1
>>> berendsenP.pressure = 1.0
>>> integrator.addExtension(berendsenP)
```

!IMPORTANT In order to run *npt* simulation one should separately define thermostat as well (e.g. BerendsenThermostat).

Definition:

In order to define the Berendsen barostat

```
>>> berendsenP = espressopp.integrator.BerendsenBarostat(system)
```

one should have the System defined.

Properties:

- *berendsenP.tau*

  The property 'tau' defines the time parameter τ.

- *berendsenP.pressure*

  The property 'pressure' defines the external pressure P_0.

Setting the integration property:

```
>>> integrator.addExtension(berendsenP)
```

It will define Berendsen barostat as a property of integrator.

One more example:

```
>>> berendsen_barostat = espressopp.integrator.BerendsenBarostat(system)
>>> berendsen_barostat.tau = 10.0
>>> berendsen_barostat.pressure = 3.5
>>> integrator.addExtension(berendsen_barostat)
```

Canceling the barostat:

If one do not need the pressure regulation in system anymore or need to switch the ensamble or whatever :)

```
>>> # define barostat with parameters
>>> berendsen = espressopp.integrator.BerendsenBarostat(system)
>>> berendsen.tau = 0.8
>>> berendsen.pressure = 15.0
>>> integrator.addExtension(berendsen)
>>> ...
>>> # some runs
>>> ...
>>> # disconnect Berendsen barostat
>>> berendsen.disconnect()
>>> # the next runs will not include the system size and particle coordinates␣
↪scaling
```

Connecting the barostat back after the disconnection

```
>>> berendsen.connect()
```

References:

espressopp.integrator.BerendsenBarostat(*system*)

Parameters

system –

3.6.4 espressopp.integrator.BerendsenBarostatAnisotropic

#TODO fix these comments This is the Berendsen barostat implementation according to the original paper [Berendsen84]. If Berendsen barostat is defined (as a property of integrator) then at the each run the system size and the particle coordinates will be scaled by scaling parameter μ according to the formula:

$$\mu = [1 - \Delta t/\tau(P_0 - P)]^{1/3}$$

where Δt - integration timestep, τ - time parameter (coupling parameter), P_0 - external pressure and P - instantaneous pressure.

Example:

```
>>> berendsenP = espressopp.integrator.BerendsenBarostatAnisotropic(system)
>>> berendsenP.tau = 0.1
>>> berendsenP.pressure = 1.0
>>> integrator.addExtension(berendsenP)
```

!IMPORTANT In order to run *npt* simulation one should separately define thermostat as well (e.g. BerendsenThermostat).

Definition:

In order to define the Berendsen barostat

```
>>> berendsenP = espressopp.integrator.BerendsenBarostatAnisotropic(system)
```

one should have the System defined.

Properties:

- *berendsenP.tau*

  The property 'tau' defines the time parameter τ.

- *berendsenP.pressure*

  The property 'pressure' defines the external pressure P_0.

Setting the integration property:

```
>>> integrator.addExtension(berendsenP)
```

It will define Berendsen barostat as a property of integrator.

One more example:

```
>>> berendsen_barostat = espressopp.integrator.BerendsenBarostatAnisotropic(system)
>>> berendsen_barostat.tau = 10.0
>>> berendsen_barostat.pressure = 3.5
>>> integrator.addExtension(berendsen_barostat)
```

Canceling the barostat:

If one do not need the pressure regulation in system anymore or need to switch the ensamble or whatever :)

```
>>> # define barostat with parameters
>>> berendsen = espressopp.integrator.BerendsenBarostatAnisotropic(system)
>>> berendsen.tau = 0.8
```

(continues on next page)

(continued from previous page)

```
>>> berendsen.pressure = 15.0
>>> integrator.addExtension(berendsen)
>>> ...
>>> # some runs
>>> ...
>>> # disconnect Berendsen barostat
>>> berendsen.disconnect()
>>> # the next runs will not include the system size and particle coordinates␣
↪scaling
```

Connecting the barostat back after the disconnection

```
>>> berendsen.connect()
```

espressopp.integrator.BerendsenBarostatAnisotropic(*system*)

Parameters
system –

3.6.5 espressopp.integrator.BerendsenThermostat

This is the Berendsen thermostat implementation according to the original paper [Berendsen84]. If Berendsen thermostat is defined (as a property of integrator) then at the each run the system size and the particle coordinates will be scaled by scaling parameter λ according to the formula:

$$\lambda = [1 + \Delta t/\tau_T(T_0/T - 1)]^{1/2}$$

where Δt - integration timestep, τ_T - time parameter (coupling parameter), T_0 - external temperature and T - instantaneous temperature.

Example:

```
>>> berendsenT = espressopp.integrator.BerendsenThermostat(system)
>>> berendsenT.tau = 1.0
>>> berendsenT.temperature = 1.0
>>> integrator.addExtension(berendsenT)
```

Definition:

In order to define the Berendsen thermostat

```
>>> berendsenT = espressopp.integrator.BerendsenThermostat(system)
```

one should have the System defined.

Properties:

- *berendsenT.tau*

  The property 'tau' defines the time parameter τ_T.

- *berendsenT.temperature*

  The property 'temperature' defines the external temperature T_0.

Setting the integration property:

```
>>> integrator.addExtension(berendsenT)
```

It will define Berendsen thermostat as a property of integrator.

One more example:

```
>>> berendsen_thermostat = espressopp.integrator.BerendsenThermostat(system)
>>> berendsen_thermostat.tau = 0.1
>>> berendsen_thermostat.temperature = 3.2
>>> integrator.addExtension(berendsen_thermostat)
```

Canceling the thermostat:

```
>>> # define thermostat with parameters
>>> berendsen = espressopp.integrator.BerendsenThermostat(system)
>>> berendsen.tau = 2.0
>>> berendsen.temperature = 5.0
>>> integrator.addExtension(berendsen)
>>> ...
>>> # some runs
>>> ...
>>> # disconnect Berendsen thermostat
>>> berendsen.disconnect()
```

Connecting the thermostat back after the disconnection

```
>>> berendsen.connect()
```

espressopp.integrator.BerendsenThermostat(*system*)

Parameters
system –

3.6.6 espressopp.integrator.CapForce

This class can be used to forcecap all particles or a group of particles. Force capping means that the force vector of a particle is rescaled so that the length of the force vector is <= capforce

Example Usage:

```
>>> capforce     = espressopp.integrator.CapForce(system, 1000.0)
>>> integrator.addExtension(capForce)
```

CapForce can also be used to forcecap only a group of particles:

```
>>> particle_group = [45, 67, 89, 103]
>>> capforce       = espressopp.integrator.CapForce(system, 1000.0, particle_group)
>>> integrator.addExtension(capForce)
```

espressopp.integrator.CapForce(*system*, *capForce*, *particleGroup*)

Parameters

- system –
- capForce –

- particleGroup – (default: None)

3.6.7 espressopp.integrator.DPDThermostat

espressopp.integrator.DPDThermostat(*system*, *vl*)

Parameters

- system –
- vl –

3.6.8 espressopp.integrator.EmptyExtension

espressopp.integrator.EmptyExtension(*system*)

Parameters
system –

3.6.9 espressopp.integrator.ExtAnalyze

This class can be used to execute nearly all analysis objects within the main integration loop which allows to automatically accumulate time averages (with standard deviation error bars).

Example Usage:

```
>>> pt            = espressopp.analysis.PressureTensor(system)
>>> extension_pt = espressopp.integrator.ExtAnalyze(pt , interval=100)
>>> integrator.addExtension(extension_pt)
>>> integrator.run(10000)
>>>
>>> pt_ave = pt.getAverageValue()
>>> print "average Pressure Tensor = ", pt_ave[:6]
>>> print "            std deviation = ", pt_ave[6:]
>>> print "number of measurements  = ", pt.getNumberOfMeasurements()
```

espressopp.integrator.ExtAnalyze(*action_obj*, *interval*)

Parameters

- action_obj –
- interval (int) – (default: 1)

3.6.10 espressopp.integrator.Extension

espressopp.integrator.Extension.connect()

Return type

espressopp.integrator.Extension.disconnect()

Return type

3.6.11 espressopp.integrator.ExtForce

espressopp.integrator.ExtForce(*system*, *extForce*, *particleGroup*)

Parameters

- system –
- extForce –
- particleGroup – (default: None)

3.6.12 espressopp.integrator.FixPositions

espressopp.integrator.FixPositions(*system*, *particleGroup*, *fixMask*)

Parameters

- system –
- particleGroup –
- fixMask –

3.6.13 espressopp.integrator.FreeEnergyCompensation

Free Energy Compensation used in Hamiltonian Adaptive Resolution Simulations (H-AdResS) or Path Integral Adaptive Resolution Simulations (PI-AdResS). This works for spherical or slab adaptive resolution geometries. However, it only works for fixed, non-moving atomistic region (otherwise, H-AdResS is not properly defined).

Example:

```
>>> fec = espressopp.integrator.FreeEnergyCompensation(system, center=[Lx/2, Ly/2, Lz/2])
>>> # set up the fec module with the center in the center of the box
>>> fec.addForce(itype=3,filename="tablefec.xvg",type=typeCG)
>>> # set up the actual force
>>> integrator.addExtension(fec)
>>> # add to previously defined integrator
```

espressopp.integrator.FreeEnergyCompensation(*system*, *center*, *sphereAdr*, *ntrotter*, *slow*)

Parameters

- system (std::shared_ptr<System>) – system object
- center (list of reals) – (default: [], corresponds to (0.0, 0.0, 0.0) position) center of high resolution region
- sphereAdr (bool) – (default: False) Spherical AdResS region (True) vs. slab geometry with resolution change in x-direction (False)
- ntrotter (int) – (default: 1) Trotter number when used in Path Integral AdResS. Default leads to normal non-PI-AdResS behaviour.
- slow (bool) – (default: False) When used with RESPA Velocity Verlet, this flag decides whether the Free Energy Compensation is applied together with the slow, less frequently updated forces (slow=True) or with the fast, more frequently updated (slow=False) forces.

espressopp.integrator.FreeEnergyCompensation.addForce(*itype*, *filename*, *type*)

Parameters

- itype (int) – interpolation type 1: linear, 2: Akima, 3: Cubic
- filename (string) – filename for TD force file
- type (int) – particle type on which the TD force needs to be applied

espressopp.integrator.FreeEnergyCompensation.computeCompEnergy()

Return type
real

3.6.14 espressopp.integrator.GeneralizedLangevinThermostat

espressopp.integrator.GeneralizedLangevinThermostat(*system*)

Parameters
system –

espressopp.integrator.GeneralizedLangevinThermostat.addCoeffs(*itype*, *filename*, *type*)

Parameters

- itype –
- filename –
- type –

Return type

3.6.15 espressopp.integrator.Isokinetic

espressopp.integrator.Isokinetic(*system*)

Parameters
system –

3.6.16 espressopp.integrator.LangevinBarostat

This is the barostat implementation to perform Langevin dynamics in a Hoover style extended system according to the paper [Quigley04]. It includes corrections of Hoover approach which were introduced by Martyna et al [Martyna94]. If LangevinBarostat is defined (as a property of integrator) the integration equations will be modified. The volume of system V is introduced as a dynamical variable:

$$\dot{\boldsymbol{r}}_i = \frac{\boldsymbol{p}_i}{m_i} + \frac{p_\epsilon}{W}\boldsymbol{r}_i$$

$$\dot{\boldsymbol{p}}_i = -\bigtriangledown_{\boldsymbol{r}_i}\Phi - (1+\frac{n}{N_f})\frac{p_\epsilon}{W}\boldsymbol{p}_i - \gamma\boldsymbol{p}_i + \boldsymbol{R}_i$$

$$\dot{V} = dVp_\epsilon/W$$

$$\dot{p}_\epsilon = nV(X - P_{ext}) + \frac{n}{N_f}\sum_{i=1}^{N}\frac{\boldsymbol{p}_i^2}{m_i} - \gamma_p p_\epsilon + R_p$$

where volume has a fictitious mass W and associated momentum p_ϵ, γ_p - friction coefficient, P_{ext} - external pressure and X - instantaneous pressure without white noise contribution from thermostat, n - dimension, N_f - degrees of freedom (if there are no constrains and N is the number of particles in system $N_f = nN$). R_p - values which are drawn from Gaussian distribution of zero mean and unit variance scaled by

$$\sqrt{\frac{2k_BTW\gamma_p}{\Delta t}}$$

!IMPORTANT Terms $-\gamma \boldsymbol{p}_i + \boldsymbol{R}_i$ correspond to the termostat. They are not included here and will not be calculated if the Langevin Thermostat is not defined.

Example:

```
>>> rng = espressopp.esutil.RNG()
>>> langevinP = espressopp.integrator.LangevinBarostat(system, rng, desiredTemperature)
>>> langevinP.gammaP = 0.05
>>> langevinP.pressure = 1.0
>>> langevinP.mass = pow(10.0, 4)
>>> integrator.addExtension(langevinP)
```

!IMPORTANT This barostat is supposed to be run in a couple with thermostat in order to simulate the *npt* ensamble, because the term R_p needs the temperature as a parameter.

Definition:

In order to define the Langevin-Hoover barostat

```
>>> langevinP = espressopp.integrator.LangevinBarostat(system, rng, desiredTemperature)
```

one should have the System and *RNG* defined and know the desired temperature.

Properties:

- *langevinP.gammaP*

  The property 'gammaP' defines the friction coefficient γ_p.

- *langevinP.pressure*

  The property 'pressure' defines the external pressure P_{ext}.

- *langevinP.mass*

  The property 'mass' defines the fictitious mass W.

Methods:

- *setMassByFrequency(frequency)*

  Set the proper *langevinP.mass* using expression $W = dNk_bT/\omega_b^2$, where frequency, ω_b, is the frequency of required volume fluctuations. The value of ω_b should be less then the lowest frequency which appears in the NVT temperature spectrum [Quigley04] in order to match the canonical distribution. d - dimensions, N - number of particles, k_b - Boltzmann constant, T - desired temperature.

NOTE The *langevinP.mass* can be set both directly and using the (*setMassByFrequency(frequency)*)

Adding to the integration:

```
>>> integrator.addExtension(langevinP)
```

It will define Langevin-Hoover barostat as a property of integrator.

One more example:

```
>>> rngBaro = espressopp.esutil.RNG()
>>> lP = espressopp.integrator.LangevinBarostat(system, rngBaro, desiredTemperature)
>>> lP.gammaP = .5
>>> lP.pressure = 1.0
>>> lP.mass = pow(10.0, 5)
>>> integrator.addExtension(lP)
```

Canceling the barostat:

If one do not need the pressure regulation in system anymore or need to switch the ensamble or whatever :)

```
>>> # define barostat with parameters
>>> rngBaro = espressopp.esutil.RNG()
>>> lP = espressopp.integrator.LangevinBarostat(system, rngBaro,␣
↪desiredTemperature)
>>> lP.gammaP = .5
>>> lP.pressure = 1.0
>>> lP.mass = pow(10.0, 5)
>>> integrator.langevinBarostat = lP
>>> ...
>>> # some runs
>>> ...
>>> # disconnect barostat
>>> langevinBarostat.disconnect()
>>> # the next runs will not include the modification of integration equations
```

Connecting the barostat back after the disconnection

```
>>> langevinBarostat.connect()
```

References:

espressopp.integrator.LangevinBarostat(*system*, *rng*, *temperature*)

Parameters

- system –
- rng –
- temperature –

3.6.17 espressopp.integrator.LangevinThermostat

Langevin Thermostat

Example:

```
>>> langevin = espressopp.integrator.LangevinThermostat(system)
>>> # set up the thermostat
>>> langevin.gamma = gamma
>>> # set friction coefficient gamma
>>> langevin.temperature = temp
>>> # set temperature
>>> langevin.adress = True
```

(continues on next page)

(continued from previous page)

```
>>> # set adress (default is False)
>>> integrator.addExtension(langevin)
>>> # add extensions to a previously defined integrator
```

espressopp.integrator.LangevinThermostat(*system*)

Parameters
system (std::shared_ptr<System>) – system object

espressopp.integrator.LangevinThermostat.addExclusions(*pidlist*)

Parameters
pidlist (list of ints) – list of particle ids to be excluded from thermostating. In adaptive (AdResS) simulations, add ids of atomistic particles to be excluded (thermostats acts in this case on atomistic level). For normal simulations, add normal or coarse-grained particle ids.

3.6.18 espressopp.integrator.LangevinThermostat1D

espressopp.integrator.LangevinThermostat1D(*system*)

Parameters
system –

3.6.19 espressopp.integrator.LangevinThermostatHybrid

As LangevinThermostat, but for use in AdResS systems, to allow the application of different thermostat friction constants (γ) to different AdResS regions. Uses three values of γ, one for the atomistic region, one for the hybrid region, and one for the coarse-grained region.

```
>>> # create FixedTupleList object
>>> ftpl = espressopp.FixedTupleListAdress(system.storage)
>>> ftpl.addTuples(tuples)
>>> system.storage.setFixedTuplesAdress(ftpl)
>>>
>>> system.storage.decompose()
>>>
>>> # create Langevin thermostat
>>> thermostat             = espressopp.integrator.LangevinThermostatHybrid(system,ftpl)
>>>
>>> # set Langevin friction constants
>>> thermostat.gamma       = 0.0 # units = 1/timeunit
>>> print "# gamma for atomistic region for langevin thermostat = ",thermostat.gamma
>>> thermostat.gammahy     = 10.0 # units = 1/timeunit
>>> print "# gamma for hybrid region for langevin thermostat = ",thermostat.gammahy
>>> thermostat.gammacg     = 10.0 # units = 1/timeunit
>>> print "# gamma for coarse-grained region for langevin thermostat = ",thermostat.gammacg
>>>
>>> # set temperature of thermostat
>>> thermostat.temperature = kBT
>>> # kBT is a float with the value of temperature in reduced units, i.e. temperature * Boltzmann's␣
↪constant in appropriate units
```

No need to include the line

```
>>> thermostat.adress = True
```

as is necessary in the case of the basic LangevinThermostat, because LangevinThermostatHybrid is always only used in AdResS systems

3.6.20 espressopp.integrator.LangevinThermostatOnGroup

Thermalize particles in the ParticleGroup only.

espressopp.integrator.LangevinThermostatOnGroup(*system*, *particle_group*)

Parameters

- system (espressopp.System) – The system object.
- particle_group (espressopp.ParticleGroup) – The particle group.

Example

```
>>> pg = espressopp.ParticleGroup(system.storage)
>>> for pid in range(10):
>>>     pg.add(pid)
>>> thermostat = espressopp.integrator.LangevinThermostatOnGroup(system, pg)
>>> thermostat.temperature = 1.0
>>> thermostat.gamma = 1.0
>>> integrator.addExtension(thermostat)
```

3.6.21 espressopp.integrator.LangevinThermostatOnRadius

Langevin Thermostat for Radii of Particles

Example:

```
>>> radius_mass = mass
>>> # set virtual mass for dynamics of radius
>>> langevin = espressopp.integrator.LangevinThermostatOnRadius(system, radius_mass)
>>> # set up the thermostat
>>> langevin.gamma = gamma
>>> # set friction coefficient gamma
>>> langevin.temperature = temp
>>> # set temperature
>>> integrator.addExtension(langevin)
>>> # add extensions to a previously defined integrator
```

espressopp.integrator.LangevinThermostatOnRadius(*system*, *dampingmass*)

Parameters

- system –
- _dampingmass –

espressopp.integrator.LangevinThermostatOnRadius.addExclusions(*pidlist*)

Parameters
pidlist (list of ints) – list of particle ids to be excluded from thermostating.

3.6.22 espressopp.integrator.LatticeBoltzmann

Overview

espressopp.integrator.LatticeBoltzmann(...)

Details

The LatticeBoltzmann (LB) class controls fluid hydrodynamics and allows for hybrid LB/MD simulations. It is implemented as espressopp.integrator.Extension in ESPResSo++.

class espressopp.integrator.LatticeBoltzmann(*system*, *nodeGrid*, *a=1.*, *tau=1.*, *numDims=3*, *numVels=19*)

Parameters

- system (std::shared_ptr) – system object
- nodeGrid (Int3D) – arrangement of CPUs in space
- a (real) – lattice spacing (in lattice units).
- tau (real) – time discretization (in lattice units)
- numDims (int) – dimensionality of the LB model
- numVels (int) – number of velocity vectors in the LB model

Returns
lb object

The LB-fluid in ESPResSo++ is aiming at simulations of complex soft matter systems. They consist of MD particles (colloids, composite nanoparticles or polymer chains) that are solved in the LB-fluid preserving hydrodynamic interactions.

The default lattice model is D3Q19 ($\mathrm{numDims} = 3$, $\mathrm{numVels} = 19$) and both lattice spacing a and timestep tau are set to 1. If some other lattice model is needed feel free to modify the code: adding 3D ones is straightforward, for 2D cases one has to make more thouroughs changes.

The parameters of the LB-fluid are expected in Lennard-Jones (LJ) units. This strategy helps users with MD-background think of LB-fluid in term of LJ liquid. One only has to specify its properties such as liquid density, ρ, temperature, T, and viscosity, η.

Note: Standard LJ fluid can be characterized by $\rho \sim 1[\sigma^{-3}]$, $T \sim 1[\epsilon]$, and $\eta \sim 5[\epsilon\tau/\sigma^3]$

Example

```
>>> L = 20
>>>
>>> # create cubic box
>>> box = (L, L, L)
```

(continues on next page)

(continued from previous page)

```
>>> # The rc+skin= lattice_size
>>> rc = 0.9
>>> skin= 0.1
>>>
>>> # initialize empty default system with the created cubic box.
>>> system, integrator = espressopp.standard_system.Default(box)
>>>
>>> # nodeGrid is determined based on the number of CPUs used for simulation among others
>>> nodeGrid=espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,rc,
↪skin)
>>>
>>> # initialize lb object. The dimensions of the lattice are obtained from the
>>> # system's box dimensions employing lattice spacing 1.
>>> lb = espressopp.integrator.LatticeBoltzmann(system, nodeGrid)
```

Methods

getLBMom(*node*, *moment*)

Get hydrodynamic moment from a specific node

Parameters

- node (Int3D) – node index
- moment (int) – hydrodynamic moment to get

Use 0 to get density ρ and 1-3 for mass flux components j_x, j_y and j_z, correspondingly.

setLBMom(*node*, *moment*, *value*)

Set hydrodynamic moment for a specific node

Parameters

- node (Int3D) – node index
- moment (int) – hydrodynamic moment to set
- value (real) – value to set

saveLBConf()

Dumps LB configuration with separate files for coupling forces, LB-fluid moments and populations (the last one is a bit overkill). The dump files are written for every CPU separately and are put in the *dump* folder

keepLBDump()

Sets a flag to keep previously dumped LB configuration. Normally the previous dump is deleted after a new one is made.

Example

```
>>> # set bulk viscosity
>>> for k in range (10):
>>>     integrator.run(50000)
>>>
>>>     # output LB configuration
>>>     lb.keepLBDump()         # flag to keep previously saved LB state
>>>     lb.saveLBConf()         # saves current state of the LB fluid
```

Properties

Int3D nodeGrid

Array of CPUs in space

Example

```
>>> # it is advised to set nodeGrid by internal ESPResSo++ function
>>> # based on the number of CPUs
>>> nodeGrid=espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,
↪rc,skin)
```

real a = 1.

Lattice spacing (*lattice units*)

real tau = 1.

Lattice time step (*lattice units*)

int numDims = 3

Number of dimensions of the LB model (*D3Q19*)

int numVels = 19

Number of velocity vectors of the LB model (*D3Q19*)

real visc_b

Bulk viscosity (*LJ units*), affects gamma_b.

Example

```
>>> # set bulk viscosity
>>> lb.visc_b = 5.
```

real visc_s

Shear viscosity (*LJ units*), affects gamma_s.

Example

```
>>> # set shear viscosity
>>> lb.visc_s = 5.
```

real gamma_b = 0.

Bulk gamma (for experienced LB users)

real gamma_s = 0.

Shear gamma (for experienced LB users)

real gamma_odd = 0.

Odd gamma (for experienced LB users)

real gamma_even = 0.

Even gamma (for experienced LB users)

real lbTemp = 0.

Temperature of the LB fluid (*LJ units*)

Example

```
>>> L = 20
>>> T = 1.
>>> N = 200
>>>
>>> # create cubic box
>>> box = (L, L, L)
>>> rc=0.9
>>> skin=0.1
>>>
>>> # initialize Lennard Jones system with the created cubic box and given temperature.
>>> system, integrator = espressopp.standard_system.LennardJones(N, box, temperature=T)
>>>
>>> # nodeGrid is determined based on the number of CPUs used for simulation
>>> nodeGrid=espressopp.tools.decomp.nodeGrid(espressopp.MPI.COMM_WORLD.size,box,
↪rc,skin)
>>>
>>> # initialize lb object. The dimensions of the lattice are obtained from the
>>> # system's box dimensions employing lattice spacing 1.
>>> lb = espressopp.integrator.LatticeBoltzmann(system, nodeGrid)
>>>
>>> # set LB temperature to T
>>> lb.lbTemp = T
```

real fricCoeff=5.

Friction coefficient of the coupling (*LJ units*)

Example

```
>>> # set friction coefficient of the coupling
>>> lb.fricCoeff = 20.
```

int nSteps = 1

Timescale contrast (ratio) between LB and MD

Example

```
>>> # set time step contrast between LB and MD
>>> lb.nSteps = 10
```

int profStep = 10000

Frequency of time profiling

Example

```
>>> # set profiling frequency
>>> lb.profStep = 5000
```

Int3D getMyNi

Number of real and halo nodes for the CPU

3.6.23 espressopp.integrator.LBInit

Overview

espressopp.integrator.LBInitPopUniform(...)
espressopp.integrator.LBInitPopWave(*args, ...)
espressopp.integrator.LBInitConstForce(...)
espressopp.integrator.LBInitPeriodicForce(...)

Details

This abstract class provides the interface to (re-)initialize populations and handle external forces.

class espressopp.integrator.LBInit

createDenVel(*rho0*, *u0*)

to set initial density and velocity of the LB-fluid.

Parameters

- rho0 (real) – density
- u0 (Real3D) – velocity

The following options for LB-fluid initialization are supported:

- espressopp.integrator.LBInitPopUniform A typical choice. It initializes uniformly distributed density and velocity: On every lattice site the density is rho0 and velocity is u0
- espressopp.integrator.LBInitPopWave for uniform density at every lattice site, but harmonic velocity $v_z(x)$ with the period of lattice sites in *x*-direction

setForce(*value*)

to set an external force onto LB-fluid.

Parameters

value (Real3D) – value of the force

addForce(*force*)

to add a new external force to the existing one.

Parameters

force (Real3D) – value of the force

Two main external force types are implemented:

- espressopp.integrator.LBInitConstForce to manage constant (gravity-like) forces acting on every lattice site and
- espressopp.integrator.LBInitPeriodicForce to manage periodic (sin-like) forces

espressopp.integrator.LBInitPopUniform

This class creates LB-fluid with uniform density rho0 and velocity u0 (*lattice units*).

Example:

```
>>> # set initial density and velocity
>>> initDen = 1.
>>> initVel = Real3D( 0. )
>>>
>>> # create initPop object and initialize populations
>>> initPop = espressopp.integrator.LBInitPopUniform(system,lb)
>>> initPop.createDenVel( initDen, initVel )
```

espressopp.integrator.LBInitPopWave

This class creates LB-fluid with uniform density and harmonic velocity (*lattice units*):

$v_x = 0$, $v_y = 0$, $v_z(i) = A \cdot sin(2\pi \cdot i\ /N_x)$, where A is the amplitude of the velocity wave, N_x is the number of lattice nodes in x-direction and i is the index of the node the velocity is calculated for.

This may be used to test the system: total moment is zero and the liquid tends to equilibrium, i.e. relaxes to a uniform zero velocity.

Example:

```
>>> # set initial density
>>> initDen = 1.
>>>
>>> # set initial velocity
>>> Vx = Vy = 0.
>>> ampVz = 0.0005
>>> initVel = Real3D( Vx, Vy, ampVz )
>>>
>>> # create initPop object and initialize populations
>>> initPop = espressopp.integrator.LBInitPopWave(system,lb)
>>> initPop.createDenVel( initDen, initVel )
```

espressopp.integrator.LBInitConstForce

This class allows to set or add constant (gravity-like) external forces (*lattice units*) to the LB-fluid. At first, one has to create a force object and then set or add this force to the system.

Example to set extenal force:

```
>>> extForceToSet = Real3D(0., 0., 0.0005)
>>> lbforce = espressopp.integrator.LBInitConstForce(system,lb)
>>> lbforce.setForce( extForceToSet )
```

Example to add extenal force to the existing forces:

```
>>> extForceToAdd = Real3D(0.0001, 0., 0.)
>>> lbforce = espressopp.integrator.LBInitConstForce(system,lb)
>>> lbforce.addForce( extForceToAdd )
```

espressopp.integrator.LBInitPeriodicForce

This class allows to set or add external periodic forces (*lattice units*) to the LB-fluid. At first, one has to create a force object and then set or add this force to the system.

Note: Please note, that a periodic (sin-like) force acts in z-direction as a function of x. The z-component of the force provides therefore the amplitude of the sin-modulation. The x- and y-components of the specified force interpreted as body forces in corresponding directions.

Example to set external sin-like force.

```
>>> ampFz = 0.0001
>>> Fx = Fy = 0.
>>> extForceToSet = Real3D( Fx, Fy, ampFz )
>>> lbforceSin = espressopp.integrator.LBInitConstForce(system,lb)
>>> lbforceSin.setForce( extForceToSet )
```

Example to add external sin-like force.

```
>>> ampFz = 0.0005
>>> Fx = Fy = 0.
>>> extForceToAdd = Real3D( Fx, Fy, ampFz )
>>> lbforceSin = espressopp.integrator.LBInitConstForce(system,lb)
>>> lbforceSin.addForce( extForceToAdd )
```

3.6.24 espressopp.integrator.MDIntegrator

espressopp.integrator.MDIntegrator.addExtension(*extension*)

Parameters
extension –

Return type

espressopp.integrator.MDIntegrator.getExtension(*k*)

Parameters
k –

Return type

espressopp.integrator.MDIntegrator.getNumberOfExtensions()

Return type

espressopp.integrator.MDIntegrator.run(*niter*)

Parameters
niter –

Return type

3.6.25 espressopp.integrator.MinimizeEnergy

This is a very simple approach to perform energy minimization of the system. The module uses a steepest descent method. The position of particles is updated following the equation:

$$p_{i+1} = p_i + min(\gamma F_i, d_{max})$$

where p_{i+} is a new position, p_i is a position at current step with corresponding force F_i. The parameters γ and d_{max} are set by user and control the relaxation of the energy and the maximum update of the coordinates per step.

Additionaly, a variable γ step is also implemented. In this case, the position of particles is updated following the equation:

$$p_{i+1} = p_i + d_{max}/f_{max}F_i$$

where f_{max} is a maximum force in a single step of steepest descent method. $\gamma = d_{max}/f_{max}$ is automatically adjusted to a force magnitude.

In both cases, the routine runs until the maximum force is bigger than f_{max} or for at most *n* steps.

Please note This module does not support any integrator extensions.

Example

```
>>> em = espressopp.integrator.MinimizeEnergy(system, gamma=0.001, ftol=0.01, max_
↪displacement=0.0001)
>>> em.run(10000)
```

Example

```
>>> em = espressopp.integrator.MinimizeEnergy(system, gamma=0.01, ftol=0.01, max_displacement=0.
↪01, variable_step_flag=True)
>>> em.run(10000)
```

API

espressopp.integrator.MinimizeEnergy(*system*, *gamma*, *ftol*, *max_displacement*, *variable_step_flag*)

Parameters

- system (espressopp.System) – The espressopp system object.
- gamma (float) – The gamma value.
- ftol (float) – The force tolerance
- max_displacement (float) – The maximum displacement.
- variable_step_flag (bool) – The flag of adjusting gamma to the force strength.

espressopp.integrator.MinimizeEnergy.run(*max_steps*, *verbose*)

Parameters

- max_steps (int) – The maximum number of steps to run.
- verbose (bool) – If set to True then display information about maximum force during the iterations.

Returns

The true if the maximum force in the system is lower than ftol otherwise false.

Return type
bool

espressopp.integrator.MinimizeEnergy.f_max

The maximum force in the system.

espressopp.integrator.MinimizeEnergy.displacement

The maximum displacement used during the run of MinimizeEnergy

espressopp.integrator.MinimizeEnergy.step

The current iteration step.

3.6.26 espressopp.integrator.OnTheFlyFEC

espressopp.integrator.OnTheFlyFEC(*system*, *center*)

Parameters

- system –
- center – (default: [])

espressopp.integrator.OnTheFlyFEC.getBins()

Return type

espressopp.integrator.OnTheFlyFEC.getGap()

Return type

espressopp.integrator.OnTheFlyFEC.getSteps()

Return type

espressopp.integrator.OnTheFlyFEC.makeArrays()

Return type

espressopp.integrator.OnTheFlyFEC.resetCounter()

Return type

espressopp.integrator.OnTheFlyFEC.writeFEC()

Return type

3.6.27 espressopp.integrator.PIAdressIntegrator

The PIAdressIntegrator implements the integration method for Hamiltonian Adaptive Resolution Path Integral Simulations proposed in J. Chem. Phys 147, 244104 (2017) (PI-AdResS). It can be used to run path integral molecular dynamics as well as ring polymer and centroid molecular dynamics in a quantum-classical adaptive resolution fashion, using different empirical force fields. To facilitate an efficient integration, the integrator uses a 3-layer RESPA (J. Chem. Phys. 97, 1990 (1992)) multiple timestepping scheme (inner level: intraatomic spring forces between the Trotter beads. medium level: interatomic bonded forces. outer level: interatomic non-bonded forces). Importantly, the integrator should only be used in combination with PI-AdResS interactions. Furthermore, the integrator has its own thermostat (Langevin), and the only extensions that should be used with it are the Free Energy Compensation (FreeEnergyCompensation) and the Thermodynamic Force (TDforce).

Example:

```
>>> integrator = espressopp.integrator.PIAdressIntegrator(system, verletlist, timestep_short, timesteps_
↪outerlevel, timesteps_centrallevel, nTrotter, realkinmass, constkinmass, temperature, gamma,␣
↪centroidthermostat, CMDparameter, PILE, PILElambda, clmassmultiplier, speedupFreezeRings, KTI)
>>> ...
>>> integrator.run(nsteps)
```

espressopp.integrator.PIAdressIntegrator(*system*, *verletlist*, *timestep*, *sSteps*, *mSteps*, *nTrotter*, *realKinMass*, *constKinMass*, *temperature*, *gamma*, *centroidThermostat*, *CMDparameter*, *PILE*, *PILElambda*, *CLmassmultiplier*, *speedup*, *KTI*)

Constructs the PIAdressIntegrator object. Note that all parameters can also be set and fetched via setter and getter functions. Additionally, all parameters except the system and the Verletlist are implemented as class variables that can be directly accessed and modified.

Parameters

- system (std::shared_ptr<System>) – system object
- verletlist (std::shared_ptr<VerletListAdress>) – Verletlist object. Should be an AdResS Verletlist
- timestep (real) – (default: 0.0) the inner (shortest) timestep for the calculation of the intraatomic spring forces between the Trotter beads
- sSteps (int) – (default: 1) multiplier to construct medium timestep (interatomic bonded forces) as mediumstep = sSteps * timestep
- mSteps (int) – (default: 1) multiplier to construct longest timestep (interatomic non-bonded forces) as longstep = mSteps * sSteps * timestep
- nTrotter (int) – (default: 32) Trotter number. Should be even and greather than zero.
- realKinMass (bool) – (default: True) Flag to choose whether to use real kinetic masses. If False, the higher modes' kinetic masses are multiplied with their corresponding eigenvalues of the normal mode transformation. In this way, all higher modes oscillate with the same frequency. If True, we use the kinetic masses for the higher modes which corresponding to the real dynamics (see J. Chem. Phys 147, 244104 (2017) for details)
- constKinMass (bool) – (default: False) If False, the higher modes' kinetic masses also adaptively change (AKM scheme in J. Chem. Phys 147, 244104 (2017)). If True, the higher modes' kinetic masses are constant throughout the system (CKM scheme in J. Chem. Phys 147, 244104 (2017))
- temperature (real) – (default: 2.494353 - this corresponds to 300 Kelvin) the temperature in gromacs units (Boltzmann constant kb is 1)
- gamma (real) – (default: 1.0) the Langevin thermostat's friction parameter in 1/ps
- centroidThermostat (bool) – (default: True) If True, the centroid mode is also thermostated, otherwise only the higher modes' (relevant for centroid molecular dynamics)
- CMDparameter (real) – (default: 1.0) The gamma^2 parameter used in centroid molecular dynamics. The higher modes' kinetic masses are rescaled by CMDparameter
- PILE (bool) – (default: True) If True, the higher modes are thermostated according to the PILE scheme by Ceriotti et al. (J. Chem. Phys 133, 124104 (2010)). Only makes sense in combination when using real kinetic masses (realKinMass = True)
- PILElambda (real) – (default: 0.5) lambda parameter to rescale the friction matrix. Default should be good for most applications (J. Chem. Phys 140, 234116 (2014))

- CLmassmultiplier (real) – (default: 100.0) multiplier by which the higher modes' spring masses (if constKinMass = False also the kinetic masses) are increased in the classical region
- speedup (bool) – (default: True) If True, the higher modes' are not integrated in the classical region and also the intraatomistic forces between the Trotter beads are not calculated in the classical region
- KTI (bool) – (default: False) If True, the particles' resolution parameters and adaptive masses are not updated but can be set by hand everywhere. This is necessary when running Kirkwood Thermodynamic Integration (KTI)

espressopp.integrator.PIAdressIntegrator.setVerletList(*verletlist*)

Sets the VerletList.

Parameters
verletlist (espressopp.VerletListAdress) – The VerletListAdress object.

espressopp.integrator.PIAdressIntegrator.getVerletList()

Gets the VerletList.

Returns
the Adress VerletList

Return type
std::shared_ptr<VerletListAdress>

espressopp.integrator.PIAdressIntegrator.setTimeStep(*timestep*)

Sets the inner (shortest) timestep.

Parameters
timestep (real) – the inner timestep

espressopp.integrator.PIAdressIntegrator.getTimeStep()

Gets the inner (shortest) timestep.

Returns
the inner timestep

Return type
real

espressopp.integrator.PIAdressIntegrator.setsStep(*sSteps*)

Sets the multiplier to construct medium timestep (interatomic bonded forces) as mediumstep = sSteps * timestep.

Parameters
sSteps (int) – multiplier to construct medium timestep

espressopp.integrator.PIAdressIntegrator.getsStep()

Gets the multiplier to construct medium timestep (interatomic bonded forces) as mediumstep = sSteps * timestep.

Returns
multiplier to construct medium timestep

Return type
int

espressopp.integrator.PIAdressIntegrator.setmStep(*mSteps*)

Sets the multiplier to construct longest timestep (interatomic non-bonded forces) as longstep = mSteps * sSteps * timestep.

Parameters
mSteps (int) – multiplier to construct longest timestep

espressopp.integrator.PIAdressIntegrator.getmStep()

Gets the multiplier to construct longest timestep (interatomic non-bonded forces) as longstep = mSteps * sSteps * timestep.

Returns
multiplier to construct longest timestep

Return type
int

espressopp.integrator.PIAdressIntegrator.setNtrotter(*nTrotter*)

Sets the Trotter number nTrotter. Should be even and greather than zero. Note that when calling this function, also the normal mode transformation matrix and the eigenvalues are recalculated.

Parameters
ntrotter (int) – the Trotter number

espressopp.integrator.PIAdressIntegrator.getNtrotter()

Gets the Trotter number nTrotter.

Returns
the Trotter number

Return type
int

espressopp.integrator.PIAdressIntegrator.setRealKinMass(*realKinMass*)

Sets the real kinetic mass flag.

Parameters
realKinMass (bool) – the real kinetic mass flag

espressopp.integrator.PIAdressIntegrator.getRealKinMass()

Gets the real kinetic mass flag.

Returns
the real kinetic mass flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.setConstKinMass(*constKinMass*)

Sets the constant kinetic mass flag.

Parameters
constKinMass (bool) – the constant kinetic mass flag

espressopp.integrator.PIAdressIntegrator.getConstKinMass()

Gets the constant kinetic mass flag.

Returns
the constant kinetic mass flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.setTemperature(*temperature*)

Sets the temperature (gromacs units with kb = 1).

Parameters
temperature (real) – the temperature

espressopp.integrator.PIAdressIntegrator.getTemperature()

Gets the temperature (gromacs units with kb = 1).

Returns
the temperature

Return type
real

espressopp.integrator.PIAdressIntegrator.setGamma(*gamma*)

Sets the friction constant gamma (in 1/ps).

Parameters
gamma (real) – the friction constant gamma

espressopp.integrator.PIAdressIntegrator.getGamma()

Gets the friction constant gamma (in 1/ps).

Returns
the friction constant gamma

Return type
real

espressopp.integrator.PIAdressIntegrator.setCentroidThermostat(*centroidThermostat*)

Sets the centroid thermostat flag.

Parameters
centroidThermostat (bool) – the centroid thermostat flag

espressopp.integrator.PIAdressIntegrator.getCentroidThermostat()

Gets the centroid thermostat flag.

Returns
the centroid thermostat flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.setCMDparameter(*CMDparameter*)

Sets the centroid molecular dynamics parameter gamma^2 for scaling the kinetic mass.

Parameters
CMDparameter (real) – the CMD parameter gamma^2

espressopp.integrator.PIAdressIntegrator.getCMDparameter()

Gets the centroid molecular dynamics parameter gamma^2 for scaling the kinetic mass.

Returns
the CMD parameter gamma^2

Return type
real

espressopp.integrator.PIAdressIntegrator.setPILE(*PILE*)

Sets the PILE flag.

Parameters
PILE (bool) – the PILE flag

espressopp.integrator.PIAdressIntegrator.getPILE()

Gets the PILE flag.

Returns
the PILE flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.setPILElambda(*PILElambda*)

Sets the scaling parameter lambda of the PILE thermostat.

Parameters
PILElambda (real) – the scaling parameter lambda

espressopp.integrator.PIAdressIntegrator.getPILElambda()

Gets the scaling parameter lambda of the PILE thermostat.

Returns
the scaling parameter lambda

Return type
real

espressopp.integrator.PIAdressIntegrator.setClmassmultiplier(*CLmassmultiplier*)

Sets the multiplier for the higher modes' spring masses in the classical region.

Parameters
CLmassmultiplier (real) – the classical spring mass multiplier

espressopp.integrator.PIAdressIntegrator.getClmassmultiplier()

Gets the multiplier for the higher modes' spring masses in the classical region.

Returns
the classical spring mass multiplier

Return type
real

espressopp.integrator.PIAdressIntegrator.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – the speedup flag

espressopp.integrator.PIAdressIntegrator.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.setKTI(*KTI*)

Sets the KTI flag.

Parameters
speedup (bool) – the KTI flag

espressopp.integrator.PIAdressIntegrator.getKTI()

Gets the KTI flag.

Returns
the KTI flag

Return type
bool

espressopp.integrator.PIAdressIntegrator.getVerletlistBuilds()

Gets the number of Verletlist builds.

Returns
number of Verletlist builds

Return type
int

espressopp.integrator.PIAdressIntegrator.computeRingEnergy()

Calculates the total configurational energy of all ring polymers in the system based on the springs between the Trotter beads (calculation done using mode coordinates).

Returns
total configurational ring polymer energy

Return type
real

espressopp.integrator.PIAdressIntegrator.computeRingEnergyRaw()

Calculates the total configurational energy of all ring polymers in the system based on the springs between the Trotter beads (calculation done using the Trotter beads' real space positions).

Returns
total configurational ring polymer energy

Return type
real

espressopp.integrator.PIAdressIntegrator.computeKineticEnergy()

Calculates the total kinetic energy using the modes' momenta.

Returns
total kinetic energy

Return type
real

espressopp.integrator.PIAdressIntegrator.computePositionDrift(*parttype*)

Calculates the average drift force due to the position-dependent spring masses (see Section 5.C. Eq. 63 in J. Chem. Phys 147, 244104 (2017)) on particles of type parttype. To be used during KTI for construction of free energy compensation.

Parameters
parttype (int) – the particle or atom type

Returns
average drift force due to the position-dependent spring masses

Return type
real

espressopp.integrator.PIAdressIntegrator.computeMomentumDrift(*parttype*)

Calculates the average drift force due to the position-dependent kinetic masses (see Section 5.C. Eq. 62 in J. Chem. Phys 147, 244104 (2017)) on particles of type parttype. To be used during KTI for construction of free energy compensation.

Parameters
parttype (int) – the particle or atom type

Returns
average drift force due to the position-dependent kinetic masses

Return type
real

class espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal(*system*, *verletlist*, *timestep=0.0*, *sSteps=1*, *mSteps=1*, *nTrotter=32*, *realKinMass=True*, *constKinMass=False*, *temperature=2.494353*, *gamma=1.0*, *centroidThermostat=True*, *CMDparameter=1.0*, *PILE=True*, *PILElambda=0.5*, *CLmassmultiplier=100.0*, *speedup=True*, *KTI=False*)

The (local) PIAdress Integrator.

computeKineticEnergy(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double computeKineticEnergy(espressopp::integrator::PIAdressIntegrator {lvalue})

computeMomentumDrift(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → float :

C++ signature :
double computeMomentumDrift(espressopp::integrator::PIAdressIntegrator {lvalue},int)

computePositionDrift(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → float :

C++ signature :
double computePositionDrift(espressopp::integrator::PIAdressIntegrator {lvalue},int)

computeRingEnergy(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double computeRingEnergy(espressopp::integrator::PIAdressIntegrator {lvalue})

computeRingEnergyRaw(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double computeRingEnergyRaw(espressopp::integrator::PIAdressIntegrator {lvalue})

getCMDparameter(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getCMDparameter(espressopp::integrator::PIAdressIntegrator {lvalue})

getCentroidThermostat(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getCentroidThermostat(espressopp::integrator::PIAdressIntegrator {lvalue})

getClmassmultiplier(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getClmassmultiplier(espressopp::integrator::PIAdressIntegrator {lvalue})

getConstKinMass(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getConstKinMass(espressopp::integrator::PIAdressIntegrator {lvalue})

getGamma(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getGamma(espressopp::integrator::PIAdressIntegrator {lvalue})

getKTI(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getKTI(espressopp::integrator::PIAdressIntegrator {lvalue})

getNtrotter(*(integrator_PIAdressIntegrator)arg1*) → int :

C++ signature :
int getNtrotter(espressopp::integrator::PIAdressIntegrator {lvalue})

getPILE(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getPILE(espressopp::integrator::PIAdressIntegrator {lvalue})

getPILElambda(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getPILElambda(espressopp::integrator::PIAdressIntegrator {lvalue})

getRealKinMass(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getRealKinMass(espressopp::integrator::PIAdressIntegrator {lvalue})

getSpeedup(*(integrator_PIAdressIntegrator)arg1*) → bool :

C++ signature :
bool getSpeedup(espressopp::integrator::PIAdressIntegrator {lvalue})

getTemperature(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getTemperature(espressopp::integrator::PIAdressIntegrator {lvalue})

getTimeStep(*(integrator_PIAdressIntegrator)arg1*) → float :

C++ signature :
double getTimeStep(espressopp::integrator::PIAdressIntegrator {lvalue})

getVerletList(*(integrator_PIAdressIntegrator)arg1*) → VerletListAdress :

C++ signature :
std::shared_ptr<espressopp::VerletListAdress> getVerletList(espressopp::integrator::PIAdressIntegrator {lvalue})

getVerletlistBuilds(*(integrator_PIAdressIntegrator)arg1*) → int :

C++ signature :
int getVerletlistBuilds(espressopp::integrator::PIAdressIntegrator {lvalue})

getmStep(*(integrator_PIAdressIntegrator)arg1*) → int :

C++ signature :
int getmStep(espressopp::integrator::PIAdressIntegrator {lvalue})

getsStep(*(integrator_PIAdressIntegrator)arg1*) → int :

C++ signature :
int getsStep(espressopp::integrator::PIAdressIntegrator {lvalue})

setCMDparameter(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setCMDparameter(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setCentroidThermostat(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setCentroidThermostat(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setClmassmultiplier(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setClmassmultiplier(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setConstKinMass(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setConstKinMass(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setGamma(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setGamma(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setKTI(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setKTI(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setNtrotter(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setNtrotter(espressopp::integrator::PIAdressIntegrator {lvalue},int)

setPILE(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setPILE(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setPILElambda(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setPILElambda(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setRealKinMass(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setRealKinMass(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setSpeedup(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setSpeedup(espressopp::integrator::PIAdressIntegrator {lvalue},bool)

setTemperature(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setTemperature(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setTimeStep(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setTimeStep(espressopp::integrator::PIAdressIntegrator {lvalue},double)

setVerletList(*(integrator_PIAdressIntegrator)arg1*, *(VerletListAdress)arg2*) → None :

C++ signature :
void setVerletList(espressopp::integrator::PIAdressIntegrator {lvalue},std::shared_ptr<espressopp::VerletListAdress>)

setmStep(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setmStep(espressopp::integrator::PIAdressIntegrator {lvalue},int)

setsStep(*(integrator_PIAdressIntegrator)arg1*, *(object)arg2*) → None :

C++ signature :
void setsStep(espressopp::integrator::PIAdressIntegrator {lvalue},int)

3.6.28 espressopp.integrator.Rattle

RATTLE algorithm for satisfying bond constraints and making the corresponding velocity corrections.

Refs:

Andersen, H. C. Rattle: A velocity version of the Shake algorithm for molecular dynamics calculations, J. Comp. Physics, 52, 24-34 (1983)

Allen & Tildesley, Computer Simulation of Liquids, OUP, 1987

RATTLE is implemented as an integrator extension, and takes as input a list of lists detailing, for each bond to be constrained: the indices of the two particles involved, the constraint distance, and the particle masses.

This implementation is intended for use with hydrogen-heavy atom bonds, which form isolated groups of constrained bonds, e.g NH2 or CH3 groups. The particle which participates in only one constrained bond (i.e. the hydrogen) should be listed first. The particle listed second (the heavy atom) may participate in more than one constrained bond. This implementation will not work if both particles participate in more than one constrained bond.

Note: At the moment, the RATTLE implementation only works if all atoms in an isolated group of rigid bonds are on the same CPU. This can be achieved by grouping all the particles using DomainDecompositionAdress and FixedTupleListAdress. The groups of rigid bonds can be identified using the dictionary constrainedBondsDict (see example below).

Note: The constraints are not taken into account in other parts of the code, such as temperature or pressure calculation.

Python example script for one methanol molecule where atoms are indexed in the order C H1 H2 H3 OH HO:

```
>>> # list for each constrained bond which lists: heavy atom index, light atom index, bond length,
↪heavy atom mass, light atom mass
>>> constrainedBondsList = [[1, 2, 0.109, 12.011, 1.008], [1, 3, 0.109, 12.011, 1.008], [1, 4, 0.109, 12.011,
↪1.008], [5, 6, 0.096, 15.9994, 1.008]]
>>> rattle = espressopp.integrator.Rattle(system, maxit = 1000, tol = 1e-6, rptol = 1e-6)
>>> rattle.addConstrainedBonds(constrainedBondsList)
>>> integrator.addExtension(rattle)
```

This list of lists of constrained bonds can be conveniently built using the espressopppp tool *findConstrainedBonds*.

```
>>> # Automatically identify hydrogen-containing bonds among the particles whose indices are in the
↪list pidlist
>>> # pidlist - list of indices of particles in which to search for hydrogens (list of int)
>>> # masses - list of masses of all particles (list of real)
>>> # massCutoff - atoms with mass < massCutoff are identified as hydrogens (real)
>>> # bondtypes - dictionary (e.g. obtained using espressopppp.gromacs.read()), key: bondtype (int),
↪value: list of tuples of the indices of the two particles in each bond of that bondtype (list of 2-tuples of
↪integers)
>>> # bondtypeparams - dictionary (e.g. obtained using espressopppp.gromacs.read()), key: bondtype
↪(int), value: espressopppp interaction potential instance
>>> hydrogenIDs, constrainedBondsDict, constrainedBondsList = espressopp.tools.
↪findConstrainedBonds(pidlist, bondtypes, bondtypeparams, masses, massCutoff = 1.1)
>>> # hydrogenIDs - list of indices of hydrogen atoms
>>> # constrainedBondsDict - dictionary mapping from a heavy atom to all the light atoms it is
↪bonded to, key: heavy atom index (int), value: list of light atom indices (list of int)
>>> # constrainedBondsList - list of lists, constrained bonds for use with Rattle.addConstrainedBonds()
>>> print "# found", len(hydrogenIDs)," hydrogens in the solute"
>>> print "# found", len(constrainedBondsDict)," heavy atoms involved in bonds to hydrogen"
>>> print "# will constrain", len(constrainedBondsList)," bonds using RATTLE"
```

espressopppp.integrator.Rattle(*system*, *maxit=1000*, *tol=1e-6*, *rptol=1e-6*)

Parameters

- system (espressopp.System) – espressopp system
- maxit (int) – maximum number of iterations
- tol (real) – tolerance for deciding if constraint distance and current distance are similar enough
- rptol (real) – tolerance for deciding if the angle between the bond vector at end of previous timestep and current vector has become too large

espressopppp.integrator.Rattle.addConstrainedBonds(*bondDetailsLists*)

Parameters

bondDetailsLists (list of [int, int, real, real, real]) – list of lists, each list contains pid of heavy atom, pid of light atom, constraint distance, mass of heavy atom, mass of light atom

3.6.29 espressopp.integrator.Settle

espressopp.integrator.Settle(*system*, *fixedtuplelist*, *mO*, *mH*, *distHH*, *distOH*)

Parameters

- system –
- fixedtuplelist –
- mO (real) – (default: 16.0)
- mH (real) – (default: 1.0)
- distHH (real) – (default: 1.58)
- distOH (real) – (default: 1.0)

espressopp.integrator.Settle.addMolecules(*moleculelist*)

Parameters
moleculelist –

Return type

3.6.30 espressopp.integrator.StochasticVelocityRescaling

espressopp.integrator.StochasticVelocityRescaling(*system*)

Parameters
system –

3.6.31 espressopp.integrator.TDforce

Thermodynamic force.

Example - how to turn on thermodynamic force (except for multiple moving spherical regions)

```
>>> fthd="tabletf.xvg"
>>> thdforce = espressopp.integrator.TDforce(system,verletlist) #info about centre and shape of adress␣
↪region come from the verletlist, info about size of adress region not needed, tabulated file tabletf.xvg␣
↪should be appropriate for the region size
>>> thdforce.addForce(itype=3,filename="tabletf.xvg",type=typeCG)
>>> integrator.addExtension(thdforce)
```

Example - how to turn on thermodynamic force for multiple moving spherical regions

```
>>> fthd="tabletf.xvg"
>>> thdforce = espressopp.integrator.TDforce(system, verletlist, startdist = 0.9, enddist = 2.1,␣
↪edgeweightmultiplier = 20) #info about moving centres come from the verletlist. Info about size of␣
↪adress region not needed, tabulated file tabletf.xvg should be appropriate for the region size (enddist -␣
↪startdist)
>>> thdforce.addForce(itype=3,filename="tabletf.xvg",type=typeCG)
>>> integrator.addExtension(thdforce)
```

espressopp.integrator.TDforce(*system*, *verletlist*, *startdist*, *enddist*, *edgeweightmultiplier*, *slow*)

Parameters

- system (std::shared_ptr<System>) – system object
- verletlist (std::shared_ptr<VerletListAdress>) – verletlist object
- startdist (real) – (default: 0.0) starting distance from center at which the TD force is actually applied. Needs to be altered when using several moving spherical regions (not used for static or single moving region)
- enddist (real) – (default: 0.0) end distance from center up to which the TD force is actually applied. Needs to be altered when using several moving spherical regions (not used for static or single moving region)
- edgeweightmultiplier (int) – (default: 20) interpolation parameter for multiple overlapping spherical regions (see Kreis et al., JCTC doi: 10.1021/acs.jctc.6b00440), the default should be fine for most applications (not used for static or single moving region)
- slow (bool) – (default: False) When used with RESPA Velocity Verlet, this flag decides whether the TD force is applied together with the slow, less frequently updated forces (slow=True) or with the fast, more frequently updated (slow=False) forces.

espressopp.integrator.TDforce.addForce(*itype*, *filename*, *type*)

Adds a thermodynamic force acting on particles of type "type".

Parameters

- itype (int) – interpolation type 1: linear, 2: Akima, 3: Cubic
- filename (string) – filename for TD force file
- type (int) – particle type on which the TD force needs to be applied

espressopp.integrator.TDforce.computeTDEnergy()

Computes the energy corresponding to the thermodynamics force (summing over all different particle types and thermodynamic forces on them).

Return type
real

3.6.32 espressopp.integrator.VelocityVerlet

espressopp.integrator.VelocityVerlet(*system*)

Parameters
system –

3.6.33 espressopp.integrator.VelocityVerletRESPA

This is a multiple time stepping integrator according to the RESPA scheme (J. Chem. Phys. 97, 1990 (1992)). It has two layers: All forces of type "NonbondedSlow" are updated with a frequency given by the long time step, while all other forces are calculated according to the short time step. The short time step can be defined and set as a property of the integrator object, while the long time step is given by the product of the short time step with an integer "multistep", which can also be set.

Example:

```
>>> integrator = espressopp.integrator.VelocityVerletRESPA(system)
>>> integrator.dt = timestep
>>> integrator.multistep = multistep
>>> ...
>>> integrator.run(nsteps)
```

class espressopp.integrator.VelocityVerletRESPA(*system*)

Constructs the VelocityVerletRESPA object.

Parameters
system (std::shared_ptr<System>) – system object

espressopp.integrator.VelocityVerletRESPA.setmultistep(*multistep*)

Sets the multiplier to construct the large timestep by multiplication with short time step as long_timestep = multistep * dt

Parameters
multistep – multiplier to construct the large timestep by multiplication with short time step

espressopp.integrator.VelocityVerletRESPA.getmultistep()

Gets the multiplier to construct the long timestep by multiplication with short time step as long_timestep = multistep * dt

Returns
multiplier to construct the long timestep by multiplication with short time step

Return type
int

int espressopp.integrator.VelocityVerletRESPA.multistep

Multiplier to construct the long timestep by multiplication with short time step as long_timestep = multistep * dt

real espressopp.integrator.VelocityVerletRESPA.dt

The short time step

3.6.34 espressopp.integrator.VelocityVerletOnGroup

espressopp.integrator.VelocityVerletOnGroup(*system*, *group*)

Parameters

- system –
- group –

3.6.35 espressopp.integrator.VelocityVerletOnRadius

espressopp.integrator.VelocityVerletOnRadius(*system*, *dampingmass*)

Parameters

- system –
- dampingmass –

3.7 interaction

3.7.1 Angular

espressopp.interaction.AngularPotential

Overview

espressopp.interaction.AngularCosineSquared(...)
espressopp.interaction.AngularHarmonic(...)
espressopp.interaction.Cosine(*args, **kwds)

Details

Abstract class for angular potentials that only needed to be inherited from.

class espressopp.interaction.AngularPotential

computeEnergy(**args*)

Parameters
*args –

Return type

computeForce(**args*)

Parameters
*args –

Return type

espressopp.interaction.AngularCosineSquared

Calculates the Angular Cosine Squared potential as:

$$U = K[cos(\theta) - cos(\theta_0)]^2,$$

where angle θ is the planar angle formed by three binded particles (triplet or triple).

This potential is employed by:

class espressopp.interaction.AngularCosineSquared(*K=1.0*, *theta0=0.0*)

Parameters

- K (real) – energy amplitude
- theta0 (real) – angle in radians

Return type
triple potential

A triple potential applied to every triple in the system creates an *interaction*. This is done via:

class espressopp.interaction.FixedTripleListAngularCosineSquared(*system*, *fixed_triple_list*, *potential*)

Parameters

- system (std::shared_ptr) – system object
- fixed_triple_list (list) – a fixed list of all triples in the system
- potential – triple potential (in this case, espressopp.interaction.AngularCosineSquared).

Return type
interaction

Methods

getFixedTripleList()

Return type
A Python list of fixed triples (e.g., in the chains).

setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

Example 1. Creating a fixed triple list by espressopp.FixedTripleList.

```
>>> # we assume a polymer solution of n_chains of the length chain_len each.
>>> # At first, create a list_of_triples for the system:
>>> N = n_chains * chain_len            # number of particles in the system
>>> list_of_tripples = []               # empty list of triples
>>> for n in range (n_chains):          # loop over chains
>>>     for m in range (chain_len):     # loop over chain beads
>>>         pid = n * chain_len + m
>>>         if (pid > 1) and (pid < N - 1):
>>>             list_of_tripples.append( (pid-1, pid, pid+1) )
>>>
>>> # create fixed triple list
>>> fixed_triple_list = espressopp.FixedTripleList(system.storage)
>>> fixed_triple_list.addTriples(list_of_triples)
```

Example 2. Employing an Angular Cosine Squared potential.

```
>>> # Note, the fixed_triple_list has to be generated in advance! (see Example 1)
>>>
>>> # set up the potential
>>> potAngCosSq = espressopp.interaction.AngularCosineSquared(K=0.5, theta0=0.0)
>>>
>>> # set up the interaction
>>> interAngCosSq = espressopp.interaction.FixedTripleListAngularCosineSquared(system, fixed_
↪triple_list, potAngCosSq)
>>>
>>> # finally, add the interaction to the system
>>> system.addInteraction(interAngCosSq)
```

espressopp.interaction.AngularHarmonic

Calculates the Angular Harmonic potential as:

$$U = K(\theta - \theta_0)^2,$$

where angle θ is the planar angle formed by three binded particles (triplet or triple). The usual coefficient of $1/2$ is included in K.

This potential is employed by:

class espressopp.interaction.AngularHarmonic(*K=1.0*, *theta0=0.0*)

Parameters

- K (real) – energy amplitude
- theta0 (real) – angle in radians

Return type
triple potential

A triple potential applied to every triple in the system creates an *interaction*. This is done via:

class espressopp.interaction.FixedTripleListAngularHarmonic(*system*, *fixed_triple_list*, *potential*)

Parameters

- system (std::shared_ptr) – system object
- fixed_triple_list (list) – a fixed list of all triples in the system
- potential – triple potential (in this case, espressopp.interaction.AngularHarmonic).

Return type
interaction

Methods

getFixedTripleList()

Return type
A Python list of fixed triples (e.g., in the chains).

setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

Example 1. Creating a fixed triple list by espressopp.FixedTripleList.

```
>>> # we assume a polymer solution of n_chains of the length chain_len each.
>>> # At first, create a list_of_triples for the system:
>>> N = n_chains * chain_len           # number of particles in the system
>>> list_of_tripples = []              # empty list of triples
>>> for n in range (n_chains):         # loop over chains
>>>     for m in range (chain_len):    # loop over chain beads
>>>         pid = n * chain_len + m
```

(continues on next page)

(continued from previous page)

```
>>>         if (pid > 1) and (pid < N - 1):
>>>             list_of_tripples.append( (pid-1, pid, pid+1) )
>>>
>>> # create fixed triple list
>>> fixed_triple_list = espressopp.FixedTripleList(system.storage)
>>> fixed_triple_list.addTriples(list_of_triples)
```

Example 2. Employing an Angular Harmonic potential.

```
>>> # Note, the fixed_triple_list has to be generated in advance! (see Example 1)
>>>
>>> # set up the potential
>>> potAngHarm = espressopp.interaction.AngularHarmonic(K=0.5, theta0=0.0)
>>>
>>> # set up the interaction
>>> interAngHarm = espressopp.interaction.FixedTripleListAngularHarmonic(system, fixed_triple_list,␣
↪potAngHarm)
>>>
>>> # finally, add the interaction to the system
>>> system.addInteraction(interAngHarm)
```

espressopp.interaction.Cosine

Calculates the Cosine potential as:

$$U = K(1 + cos(\theta - \theta_0))$$

espressopp.interaction.Cosine(*K*, *theta0*)

Parameters

- K (real) – (default: 1.0)
- theta0 (real) – (default: 0.0)

espressopp.interaction.FixedTripleListCosine(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedTripleListCosine.getFixedTripleList()

Return type
A Python list of lists.

espressopp.interaction.FixedTripleListCosine.setPotential(*potential*)

Parameters
potential –

3.7.2 Bonded

espressopp.interaction.FENE

Implementation of the FENE (Finitely Extensible Non-linear Elastic) potential for polymers [Kremer_1986]. It approximates the interaction between the neighboring monomers by non-linear springs. In contrast to some other packages, the FENE interaction defined here does NOT include the WCA or LJ terms. They have to be specified separately. The FENE interaction is implemented like:

$$U(r) = -\frac{1}{2} r_{\mathrm{max}}^2 K \log \left[1 - \left(\frac{r - r_0}{r_{\mathrm{max}}} \right)^2 \right].$$

class espressopp.interaction.FENE(*K=30.0*, *r0=0.0*, *rMax=1.5*, *cutoff=inf*, *shift=0.0*)

Parameters

- K (real) – attractive force strength (in ϵ/σ^2 units)
- r0 (real) – displacement parameter (in $sigma$ units)
- rMax (real) – size parameter (in $sigma$ units)
- cutoff (real) – cutoff radius
- shift (real) – shift of the potential

FENE-potential is applied to all pairs of the fixed-pair list (usually called the bondlist) via:

class espressopp.interaction.FixedPairListFENE(*system*, *bondlist*, *potential*)

Parameters

- system (object) – your system espressopp.System()
- bondlist (list) – list of bonds espressopp.FixedPairList()
- potential (object) – bonded potential, in this case espressopp.interaction.FENE()

Methods

getFixedPairList()

Return type
A Python list of pairs (the bondlist).

getPotential()

Return type
potential object

setFixedPairList(*bondlist*)

Parameters
bondlist (list) – fixed-pair list (bondlist)

setPotential(*potential*)

Parameters
potential (object) – a potential applied to all pairs of the bondlist

Example of usage

```
>>> # The following example shows how to bond particle 1 to particles 0 and 2 by a FENE potential.
>>> # We assume the particles are already in the storage of the system
>>> # Initialize list of pairs that will be bonded by FENE
>>> bondlist = espressopp.FixedPairList(system.storage)
>>> # Set which pairs belong to the pair_list i.e. particle 1 is bonded to particles 0 and 2.
>>> bondlist.addBonds([(0,1),(1,2)])
>>> # Initialize the potential and set up the parameters.
>>> potFENE  = espressopp.interaction.FENE(K=30.0, r0=0.0, rMax=1.5)
>>> # Set which system, pair list and potential is the interaction associated with.
>>> interFENE = espressopp.interaction.FixedPairListFENE(system, bondlist, potFENE)
>>> # Add the interaction to the system.
>>> system.addInteraction(interFENE)
```

espressopp.interaction.FENECapped

A capped FENE potential avoiding calculation of unreasonably large bonded forces. It is usually applied at the equilibration stage of a simulation and helps a polymer system to relax. After the system has reached its equilibrium the capped potential should be substituted by a regular FENE, espressopp.interaction.FENE.

The capped FENE potential is employed as:

$$U = -\frac{1}{2} r_{max}^2 K \cdot log \left[1 - \left(\frac{D - r_0}{r_{max}} \right)^2 \right],$$

where

$$D = min \left(r, r_{cap} \right).$$

class espressopp.interaction.FENECapped(*K=30.0*, *r0=0.0*, *rMax=1.5*, *cutoff=inf*, *r_cap=1.0*, *shift=0.0*)

Parameters

- K (real) – attractive force strength (in ϵ/σ^2 units)
- r0 (real) – displacement parameter (in $sigma$ units)
- rMax (real) – size parameter (in $sigma$ units)
- cutoff (real) – cutoff radius
- r_cap (real) – radius of capping (in $sigma$ units)
- shift (real) – shift of the potential

After setting up the potential you have to apply it to the particles in the pair list (bondlist):

class espressopp.interaction.FixedPairListFENECapped(*system*, *bondlist*, *potential*)

Parameters

- system (object) – system object espressopp.System()
- bondlist (list) – list of bonds espressopp.FixedPairList()
- potential (object) – bonded potential, in this case espressopp.interaction.FENECapped()

Methods

getFixedPairList()

Return type
A Python list of pairs (the bondlist)

getPotential()

Return type
potential object

setFixedPairList(*bondlist*)

Parameters
bondlist (list) – fixed-pair list (bondlist)

setPotential(*potential*)

Parameters
potential (object) – a potential applied to all pairs in the bondlist

Example of usage

```
>>> Please, refer to the example of FENE potential
```

Go to FENE-example espressopp.interaction.FENE

espressopp.interaction.Harmonic

$$U = K(d - r_0)^2$$

espressopp.interaction.Harmonic(*K*, *r0*, *cutoff*, *shift*)

Defines a Harmonic potential.

Parameters

- K (real) – (default: 1.0)
- r0 (real) – (default: 0.0)
- cutoff (real) – (default: infinity)
- shift (real) – (default: 0.0)

espressopp.interaction.FixedPairListHarmonic(*system*, *vl*, *potential*)

Defines a FixedPairList-based interaction using a Harmonic potential.

Parameters

- system (std::shared_ptr<System>) – system object
- vl (std::shared_ptr<FixedPairList>) – FixedPairList object
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.FixedPairListHarmonic.getFixedPairList()

Gets the FixedPairList.

Return type
std::shared_ptr<FixedPairList>

espressopp.interaction.FixedPairListHarmonic.setFixedPairList(*fixedpairlist*)

Sets the FixedPairList.

Parameters
fixedpairlist (std::shared_ptr<FixedPairList>) – FixedPairList object

espressopp.interaction.FixedPairListHarmonic.setPotential(*potential*)

Sets the Harmonic interaction potential.

Parameters
potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.FixedPairListTypesHarmonic(*system*, *vl*)

Parameters

- system (std::shared_ptr<System>) – system object
- vl (std::shared_ptr<FixedPairList>) – FixedPairList object

espressopp.interaction.FixedPairListTypesHarmonic.getFixedPairList()

Gets the FixedPairList.

Return type
std::shared_ptr<FixedPairList>

espressopp.interaction.FixedPairListTypesHarmonic.setFixedPairList(*fixedpairlist*)

Sets the FixedPairList.

Parameters
fixedpairlist (std::shared_ptr<FixedPairList>) – FixedPairList object

espressopp.interaction.FixedPairListTypesHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the Harmonic interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.FixedPairListTypesHarmonic.getPotential(*type1*, *type2*)

Gets the Harmonic interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListHarmonic(*vl*)

Defines a verletlist-based interaction using a Harmonic potential.

Parameters
vl (std::shared_ptr<VerletList>) – Verletlist object

espressopp.interaction.VerletListHarmonic.getPotential(*type1*, *type2*)

Gets the Harmonic interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the Harmonic interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListAdressATHarmonic(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based AdResS interaction using a Harmonic potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListAdressATHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListAdressATHarmonic.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListAdressATHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListAdressATHarmonic.getVerletList()

Gets the verletlist used in VerletListAdressATHarmonic interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListAdressCGHarmonic(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based AdResS interaction using a Harmonic potential for the AT interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressCGHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListAdressCGHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListAdressCGHarmonic.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListAdressCGHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListAdressCGHarmonic.getVerletList()

Gets the verletlist used in VerletListAdressCGHarmonic interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListHadressATHarmonic(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based H-AdResS interaction using a Harmonic potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListHadressATHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListHadressATHarmonic.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListHadressATHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListHadressATHarmonic.getVerletList()

Gets the verletlist used in VerletListHadressATHarmonic interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListHadressCGHarmonic(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based H-AdResS interaction using a Harmonic potential for the AT interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressCGHarmonic.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListHadressCGHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListHadressCGHarmonic.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListHadressCGHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Harmonic>

espressopp.interaction.VerletListHadressCGHarmonic.getVerletList()

Gets the verletlist used in VerletListHadressCGHarmonic interaction.

Return type
std::shared_ptr<VerletListAdress>

class espressopp.interaction.Harmonic.Harmonic(**args*, ***kwds*)

The Harmonic potential.

espressopp.interaction.HarmonicTrap

$$U = K\frac{1}{2}d^2$$

espressopp.interaction.HarmonicTrap()

espressopp.interaction.SingleParticleHarmonicTrap(*system*, *potential*)

Parameters

- system –
- potential –

espressopp.interaction.SingleParticleHarmonicTrap.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.HarmonicTrap.HarmonicTrap(**args*, ***kwds*)

The HarmonicTrap potential.

espressopp.interaction.Morse

This class provides methods to compute forces and energies of the Morse potential.

$$U = \varepsilon \left(e^{-2\alpha(r-r_{min})} - 2e^{-\alpha(r-r_{min})} \right)$$

espressopp.interaction.Morse(*epsilon*, *alpha*, *rMin*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- alpha (real) – (default: 1.0)
- rMin (real) – (default: 0.0)
- cutoff – (default: infinity)
- shift – (default: “auto”)

espressopp.interaction.VerletListMorse(*vl*)

Parameters
vl –

espressopp.interaction.VerletListMorse.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListMorse.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressMorse(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressMorse.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressMorse.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressMorse(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressMorse.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressMorse.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListMorse(*stor*)

Parameters
stor –

espressopp.interaction.CellListMorse.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListMorse(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListMorse.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.Morse.Morse(**args*, ***kwds*)

The Morse potential.

espressopp.interaction.SoftCosine

This class provides methods to compute forces and energies ofthe SoftCosine potential.

$$V(r) = A\left[1.0 + cos\left(\frac{\pi r}{r_c}\right)\right]$$

espressopp.interaction.SoftCosine(*A*, *cutoff*, *shift*)

Parameters

- A (real) – (default: 1.0)
- cutoff – (default: infinity)
- shift – (default: "auto")

espressopp.interaction.VerletListSoftCosine(*stor*)

Parameters
stor –

espressopp.interaction.VerletListSoftCosine.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –

- potential –

espressopp.interaction.CellListSoftCosine(*stor*)

Parameters
stor –

espressopp.interaction.CellListSoftCosine.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListSoftCosine(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListSoftCosine.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.SoftCosine.SoftCosine(**args*, ***kwds*)
The SoftCosine potential.

3.7.3 Charged

espressopp.interaction.CoulombKSpaceEwald

Coulomb potential and interaction Objects (*K* space part)

$$\frac{1}{2\pi V} \sum_{\substack{m\in\mathbb{Z}^3 \\ 0<|m|<k_{max}}} \frac{exp(-\frac{\pi^2}{\alpha^2}m'^2)}{m'^2} \left| \sum_{i=1}^{N} q_i \cdot exp(2\pi i r_i \cdot m') \right|^2$$

This is the *K* space part of potential of Coulomb long range interaction according to the Ewald summation technique. Good explanation of Ewald summation could be found here [Allen89], [Deserno98].

Example:

```
>>> ewaldK_pot = espressopp.interaction.CoulombKSpaceEwald(system, coulomb_prefactor, alpha,
↪kspacecutoff)
>>> ewaldK_int = espressopp.interaction.CellListCoulombKSpaceEwald(system.storage, ewaldK_pot)
>>> system.addInteraction(ewaldK_int)
```

!IMPORTANT Coulomb interaction needs *R* space part as well CoulombRSpace.

Definition:

It provides potential object *CoulombKSpaceEwald* and interaction object *CellListCoulombKSpaceEwald* based on all particles list.

The *potential* is based on the system information (System) and parameters: Coulomb prefactor (coulomb_prefactor), Ewald parameter (alpha), and the cutoff in K space (kspacecutoff).

```
>>> ewaldK_pot = espressopp.interaction.CoulombKSpaceEwald(system, coulomb_
↪prefactor, alpha, kspacecutoff)
```

Potential Properties:

- *ewaldK_pot.prefactor*

  The property 'prefactor' defines the Coulomb prefactor.

- *ewaldK_pot.alpha*

  The property 'alpha' defines the Ewald parameter $alpha$.

- *ewaldK_pot.kmax*

  The property 'kmax' defines the cutoff in *K* space.

The *interaction* is based on the all particles list. It needs the information from Storage and *K* space part of potential.

```
>>> ewaldK_int = espressopp.interaction.CellListCoulombKSpaceEwald(system.storage,␣
↪ewaldK_pot)
```

Interaction Methods:

- *getPotential()*

  Access to the local potential.

Adding the interaction to the system:

```
>>> system.addInteraction(ewaldK_int)
```

References:

espressopp.interaction.CoulombKSpaceEwald(*system*, *prefactor*, *alpha*, *kmax*)

Parameters

- system –
- prefactor –
- alpha –
- kmax –

espressopp.interaction.CellListCoulombKSpaceEwald(*storage*, *potential*)

Parameters

- storage –
- potential –

espressopp.interaction.CellListCoulombKSpaceEwald.getFixedPairList()

Return type
A Python list of lists.

espressopp.interaction.CellListCoulombKSpaceEwald.getPotential()

Return type

espressopp.interaction.CoulombKSpaceP3M

Coulomb potential and interaction Objects (*K* space part)

This is the *K* space part of potential of Coulomb long range interaction according to the P3M summation technique. Good explanation of P3M summation could be found here [Allen89], [Deserno98].

Example:

```
>>> ewaldK_pot = espressopp.interaction.CoulombKSpaceP3M(system, coulomb_prefactor, alpha,␣
↪kspacecutoff)
>>> ewaldK_int = espressopp.interaction.CellListCoulombKSpaceP3M(system.storage, ewaldK_pot)
>>> system.addInteraction(ewaldK_int)
```

!IMPORTANT Coulomb interaction needs *R* space part as well CoulombRSpace.

Definition:

It provides potential object *CoulombKSpaceP3M* and interaction object *CellListCoulombKSpaceP3M* based on all particles list.

The *potential* is based on the system information (System) and parameters: Coulomb prefactor (coulomb_prefactor), P3M parameter (alpha), and the cutoff in K space (kspacecutoff).

```
>>> ewaldK_pot = espressopp.interaction.CoulombKSpaceP3M(system, coulomb_prefactor,␣
↪alpha, kspacecutoff)
```

Potential Properties:

- *ewaldK_pot.prefactor*

  The property 'prefactor' defines the Coulomb prefactor.

- *ewaldK_pot.alpha*

  The property 'alpha' defines the P3M parameter $alpha$.

- *ewaldK_pot.kmax*

  The property 'kmax' defines the cutoff in *K* space.

The *interaction* is based on the all particles list. It needs the information from Storage and *K* space part of potential.

```
>>> ewaldK_int = espressopp.interaction.CellListCoulombKSpaceP3M(system.storage,␣
↪ewaldK_pot)
```

Interaction Methods:

- *getPotential()*

  Access to the local potential.

Adding the interaction to the system:

```
>>> system.addInteraction(ewaldK_int)
```

espressopp.interaction.CoulombKSpaceP3M(*system*, *C_pref*, *alpha*, *M*, *P*, *rcut*, *interpolation*)

Parameters

- system –
- C_pref –
- alpha –
- M –
- P –
- rcut –
- interpolation (int) – (default: 200192)

espressopp.interaction.CellListCoulombKSpaceP3M(*storage*, *potential*)

Parameters

- storage –
- potential –

espressopp.interaction.CellListCoulombKSpaceP3M.getPotential()

Return type

espressopp.interaction.CoulombRSpace

Coulomb potential and interaction Objects (*R* space part)

$$\sum_{i=1}^{N} \sum_{\substack{j>i \\ r_{ij}<k_{max}}} \frac{q_i q_j}{r_{ij}} erfc(\alpha r_{ij}) - \frac{\alpha}{\sqrt{\pi}} \sum_{i=1}^{N} q_i^2$$

This is the *R* space part of potential of Coulomb long range interaction according to the Ewald summation technique. Good explanation of Ewald summation could be found here [Allen89], [Deserno98].

Example:

```
>>> vl = espressopp.VerletList(system, rspacecutoff+skin)
>>> coulombR_pot = espressopp.interaction.CoulombRSpace(coulomb_prefactor, alpha, rspacecutoff)
>>> coulombR_int = espressopp.interaction.VerletListCoulombRSpace(vl)
>>> coulombR_int.setPotential(type1=0, type2=0, potential = coulombR_pot)
>>> system.addInteraction(coulombR_int)
```

!IMPORTANT Coulomb interaction needs k-space part as well EwaldKSpace.

Definition:

It provides potential object *CoulombRSpace* and interaction object *VerletListCoulombRSpace*

The *potential* is based on parameters: Coulomb prefactor (coulomb_prefactor), Ewald parameter (alpha), and the cutoff in R space (rspacecutoff).

```
>>> coulombR_pot = espressopp.interaction.CoulombRSpace(coulomb_prefactor, alpha,
↪rspacecutoff)
```

Potential Properties:

- *coulombR_pot.prefactor*

  The property 'prefactor' defines the Coulomb prefactor.

- *coulombR_pot.alpha*

  The property 'alpha' defines the Ewald parameter $alpha$.

- *coulombR_pot.cutoff*

  The property 'cutoff' defines the cutoff in R space.

The *interaction* is based on the Verlet list (*VerletList*)

```
>>> vl = espressopp.VerletList(system, rspacecutoff+skin)
>>> coulombR_int = espressopp.interaction.VerletListCoulombRSpace(vl)
```

It should include at least one potential

```
>>> coulombR_int.setPotential(type1=0, type2=0, potential = coulombR_pot)
```

Interaction Methods:

- *setPotential(type1, type2, potential)*

  This method sets the *potential* for the particles of *type1* and *type2*. It could be a bunch of potentials for the different particle types.

- *getVerletListLocal()*

  Access to the local Verlet list.

Adding the interaction to the system:

```
>>> system.addInteraction(coulombR_int)
```

espressopp.interaction.CoulombRSpace(*prefactor*, *alpha*, *cutoff*)

Parameters

- prefactor (real) – (default: 1.0)
- alpha (real) – (default: 1.0)
- cutoff – (default: infinity)

espressopp.interaction.VerletListCoulombRSpace(*vl*)

Parameters

vl –

espressopp.interaction.VerletListCoulombRSpace.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListCoulombRSpace.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListCoulombRSpace.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CoulombTruncated

$$U = k\frac{q_i q_j}{d_{ij}}$$

where k is the user-supplied prefactor, q_i is the charge of particle *i*, and d_{ij} is interparticle distance

In this interaction potential, a different charge can be associated with each particle. For a truncated Coulomb interaction potential where only one $q_i q_j$ value is specified for all interactions, see CoulombTruncatedUniqueCharge.

espressopppp.interaction.CoulombTruncated(*prefactor*, *cutoff*)

Parameters

- prefactor (real) – (default: 1.0) user-supplied prefactor *k*
- cutoff (real) – (default: infinity) user-supplied interaction cutoff

espressopppp.interaction.VerletListCoulombTruncated(*vl*)

Parameters
vl (espressopp.VerletList) – verlet list object defined earlier in python script

espressopppp.interaction.VerletListCoulombTruncated.getPotential(*type1*, *type2*)

Parameters

- type1 (integer) – type of first atom in pair
- type2 (integer) – type of second atom in pair

espressopppp.interaction.VerletListCoulombTruncated.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 (integer) – type of first atom in pair
- type2 (integer) – type of second atom in pair
- potential (CoulombTruncated potential) – potential object defined earlier in python script

espressopppp.interaction.FixedPairListTypesCoulombTruncated(*system*, *vl*)

Parameters

- system (espressopp.System) – system object defined earlier in the python script
- vl (espressopp.FixedPairList) – fixedpairlist object defined earlier in the python script

espressopppp.interaction.FixedPairListTypesCoulombTruncated.setPotential(*potential*)

Parameters

- type1 (integer) – type of first atom in pair
- type2 (integer) – type of second atom in pair
- potential (CoulombTruncated potential) – potential object defined earlier in python script

#Example:

```
>>> pref = 138.935485
>>> rc = 1.2
>>> fixedpairlist = espresso.FixedPairList(system.storage)
>>> fixedpairlist.addBonds([(1,2),(2,3)])
>>> pot = espressopp.interaction.CoulombTruncated(prefactor=pref, cutoff=rc)
>>> interaction=espressopp.interaction.FixedPairListTypesCoulombTruncated(system,fixedpairlist)
>>> interaction.setPotential(type1=0, type2=1, potential=pot)
>>> system.addInteraction(interaction)
```

class espressopp.interaction.CoulombTruncated.CoulombTruncated(**args*, ***kwds*)

The CoulombTruncated potential.

espressopp.interaction.CoulombTruncatedUniqueCharge

$$U = \frac{Q}{d}$$

where Q is the product of the charges of the two particles and d is their distance from each other.

In this interaction potential, a unique $Q = q_i q_j$ value is specified per potential. For a more flexible truncated Coulomb interaction potential where each individual particle has its own charge q_i, see CoulombTruncated.

espressopp.interaction.CoulombTruncatedUniqueCharge(*qq*, *cutoff*, *shift*)

Parameters

- qq (real) – (default: 1.0)
- cutoff – (default: infinity)
- shift – (default: "auto")

espressopp.interaction.VerletListCoulombTruncatedUniqueCharge(*vl*)

Parameters

vl –

espressopp.interaction.VerletListCoulombTruncatedUniqueCharge.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListCoulombTruncatedUniqueCharge.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListCoulombTruncatedUniqueCharge(*stor*)

Parameters

stor –

espressopp.interaction.CellListCoulombTruncatedUniqueCharge.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListCoulombTruncatedUniqueCharge(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListCoulombTruncatedUniqueCharge.setPotential(*potential*)

Parameters

potential –

class espressopp.interaction.CoulombTruncatedUniqueCharge.CoulombTruncatedUniqueCharge(**args*, ***kwds*)

The CoulombTruncatedUniqueCharge potential.

3.7.4 Constrained

espressopp.interaction.ConstrainCOM

This class is for calculating forces acting on constrained center of mass of subchains [Zhang_2014].

Subchains are defined as a tuple list.

$$U = k_{com} \left(\vec{r_{com}} - \vec{R_{com}} \right)^2,$$

where $\vec{r_{com}}$ stands for the center of mass of subchain and $\vec{R_{com}}$ stands for the desired center of mass of subchain.

This class implies 2 conditions on a tuple list defining subchains:

1. The length of all tuples must be the same.
2. int(key particle id / The length of a tuple) must not be redundantly, where key particle id is the smallest particle id in a tuple.

espressopp.interaction.ConstrainCOM(*k_com*)

Parameters
k_com (real) – (default: 100.)

espressopp.interaction.FixedLocalTupleListConstrainCOM(*system*, *tuplelist*, *potential*)

Parameters

- system –
- tuplelist –
- potential –

espressopp.interaction.FixedLocalTupleListConstrainCOM.getPotential()

Return type

espressopp.interaction.FixedLocalTupleListConstrainCOM.setCom(*particlelist*)

Parameters
particlelist –

espressopp.interaction.ConstrainRG

This class calculates forces acting on constrained radii of gyration of subchains [Zhang_2014].

Subchains are defined as a tuple list.

$$U = k_{rg}\left(R_g^2 - R_g^{ideal\,2}\right)^2$$

where R_g^{ideal} stands for the desired radius of gyration of subchain.

This class set 2 conditions on a tuple list. defining subchains.

1. The length of all tuples must be the same.
2. int(key particle id / The length of a tuple) must not be redundantly, where key particle id is the smallest particle id in a tuple.

espressopp.interaction.ConstrainRG(*k_rg*)

Parameters
k_rg (real) – (default: 100.)

espressopp.interaction.FixedLocalTupleListConstrainRG(*system*, *tuplelist*, *potential*)

Parameters

- system –
- tuplelist –
- potential –

espressopp.interaction.FixedLocalTupleListConstrainRG.getPotential()

Return type

espressopp.interaction.FixedLocalTupleListConstrainRG.setRG(*particlelist*)

Parameters
particlelist (python::list) –

3.7.5 Dihedral

espressopp.interaction.DihedralHarmonic

The dihedral harmonic potential

$$U(\phi_{ijkl}) = 0.5K[\phi_{ijkl} - \phi_0)]^2$$

where the K is a constant, the angles should be provided in radians.

Reference: Gromacs Manual 4.6.1, section 4.2.11 (page 79-80), equation 4.60

espressopp.interaction.DihedralHarmonic(*K*, *phi0*)

Parameters

- K (real) – (default: 0.0)
- phi0 (real) – (default: 0.0)

espressopp.interaction.FixedQuadrupleListDihedralHarmonic(*system*, *fql*, *potential*)

Parameters

- system –
- fql –
- potential –

espressopp.interaction.FixedQuadrupleListDihedralHarmonic.getFixedQuadrupleList()

Return type
A Python list of lists.

espressopp.interaction.FixedQuadrupleListDihedralHarmonic.setPotential(*potential*)

Parameters
potential –

Example of usage

```
>>> # The following example shows how to add a torsional potential to particles 1,2,3,4
>>> fql = espressopp.FixedQuadrupleList(system.storage)
>>> fql.addQuadruples([(1,2,3,4)])
>>> #phi0 is in radians, IUPAC convention definition
>>> interaction = espressopp.interaction.FixedQuadrupleListDihedralHarmonic(system,fql,
↪potential=espressopp.interaction.DihedralHarmonic(K=1.0,phi0=0.0))
>>> system.addInteraction(interaction)
```

class espressopp.interaction.DihedralHarmonic.DihedralHarmonic(**args*, ***kwds*)

The DihedralHarmonic potential.

class espressopp.interaction.DihedralHarmonic.FixedQuadrupleListDihedralHarmonicLocal(*system*, *fql*, *potential*)

The (local) DihedralHarmonic interaction using FixedQuadruple lists.

getFixedQuadrupleList(*(interaction_FixedQuadrupleListDihedralHarmonic)arg1*) → FixedQuadrupleList :

C++ signature :
std::shared_ptr<espressopp::FixedQuadrupleList> getFixedQuadrupleList(espressopp::interaction::FixedQuadrupleListInteractionTemplate<espressopp::interaction::DihedralHarmonic> {lvalue})

setPotential(*(interaction_FixedQuadrupleListDihedralHarmonic)arg1*, *(interaction_DihedralHarmonic)arg2*) → None :

C++ signature :
void setPotential(espressopp::interaction::FixedQuadrupleListInteractionTemplate<espressopp::interaction::DihedralHa {lvalue},std::shared_ptr<espressopp::interaction::DihedralHarmonic>)

espressopp.interaction.DihedralHarmonicCos

$$U = K(cos(\phi) - cos(\phi_0))^2$$

espressopp.interaction.DihedralHarmonicCos(*K*, *phi0*)

Parameters

- K (real) – (default: 0.0)
- phi0 (real) – (default: 0.0)

espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos(*system*, *fql*, *potential*)

Parameters

- system –
- fql –
- potential –

espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos.getFixedQuadrupleList()

Return type
A Python list of lists.

espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.DihedralHarmonicCos.DihedralHarmonicCos(**args*, ***kwds*)

The DihedralHarmonicCos potential.

espressopp.interaction.DihedralHarmonicNCos

The dihedral harmonic potential

$$U(\phi_{ijkl}) = K[1 + cos(N \cdot \phi_{ijkl} - \phi_0)]$$

where the *K* is a constant, the angles should be provided in radians. The *N* is a multiplicity.

Reference: http://www.uark.edu/ua/fengwang/DLPOLY2/node49.html

espressopp.interaction.DihedralHarmonicNCos(*K*, *phi0*, *multiplicity*)

Parameters

- K (real) – (default: 0.0)
- phi0 (real) – (default: 0.0)
- multiplicity (int) – (default: 1)

espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos(*system*, *fql*, *potential*)

Parameters

- system –
- fql –
- potential –

espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos.getFixedQuadrupleList()

Return type
A Python list of lists.

espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.DihedralHarmonicNCos.DihedralHarmonicNCos(**args*, ***kwds*)

The DihedralHarmonicNCos potential.

class espressopp.interaction.DihedralHarmonicNCos.FixedQuadrupleListDihedralHarmonicNCosLocal(*system*, *fql*, *potential*)

The (local) DihedralHarmonicNCos interaction using FixedQuadruple lists.

getFixedQuadrupleList(*((interaction_FixedQuadrupleListDihedralHarmonicNCos)arg1*) → FixedQuadrupleList :

C++ signature :
std::shared_ptr<espressopp::FixedQuadrupleList> getFixedQuadrupleList(espressopp::interaction::FixedQuadrupleListInteractionTemplate<espressopp::interaction::DihedralHarmonicNC
{lvalue})

setPotential(*((interaction_FixedQuadrupleListDihedralHarmonicNCos)arg1*, *(interaction_DihedralHarmonicNCos)arg2*) → None :

C++ signature :
void setPotential(espressopp::interaction::FixedQuadrupleListInteractionTemplate<espressopp::interaction::DihedralHa
{lvalue},std::shared_ptr<espressopp::interaction::DihedralHarmonicNCos>)

espressopp.interaction.DihedralPotential

This is an abstract class, only needed to be inherited from.

espressopp.interaction.DihedralPotential.computeEnergy(**args*)

Parameters
*args –

Return type

espressopp.interaction.DihedralPotential.computeForce(**args*)

Parameters
*args –

Return type

espressopp.interaction.DihedralRB

The proper dihedral with Ryckaert-Bellemans form.

$$U_{rb}(\phi_{ijkl}) = \sum_{n=0}^{5} K_n(cos(\theta))^n$$

where the $\theta = \phi - 180^\circ$ and $K_{0...5}$ are the coefficients.

By default the IUPAC convention is used, where ϕ is the angle between planes ijk and jkl. The 0° corresponds to the *cis* configuration.

Reference: http://www.gromacs.org/Documentation/Manual

espressopp.interaction.DihedralRB(*K0*, *K1*, *K2*, *K3*, *K4*, *K5*, *iupac*)

Parameters

- K0 (real) – (default: 0.0)
- K1 (real) – (default: 0.0)
- K2 (real) – (default: 0.0)
- K3 (real) – (default: 0.0)
- K4 (real) – (default: 0.0)
- K5 (real) – (default: 0.0)
- iupac – (default: True)

espressopp.interaction.FixedQuadrupleListDihedralRB(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedQuadrupleListDihedralRB.getFixedQuadrupleList()

Return type
A Python list of lists.

espressopp.interaction.FixedQuadrupleListDihedralRB.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.OPLS

This class provides methods to compute forces and energies of the OPLS dihedral potential. To create a new dihedral potential.

$$U = \sum_{j=1}^{4} K_j(1 + cos(j\phi))$$

espressopp.interaction.OPLS(*K1*, *K2*, *K3*, *K4*)

Parameters

- K1 (real) – (default: 1.0)
- K2 (real) – (default: 0.0)
- K3 (real) – (default: 0.0)
- K4 (real) – (default: 0.0)

espressopp.interaction.FixedQuadrupleListOPLS(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedQuadrupleListOPLS.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

class espressopp.interaction.OPLS.OPLS(**args*, ***kwds*)

The OPLS potential.

3.7.6 Manybody

espressopp.interaction.StillingerWeberPairTerm

This class provides methods to compute forces and energies of 2 body term of Stillinger-Weber potential.

$$U = \varepsilon A \left[\frac{d}{\sigma}^{-p} (B-1) \right] exp \left(\frac{1}{\frac{d}{\sigma} - r_c} \right)$$

where r_c is the cutoff-radius.

espressopp.interaction.StillingerWeberPairTerm(*A*, *B*, *p*, *q*, *epsilon*, *sigma*, *cutoff*)

Parameters

- A –
- B –
- p –
- q –
- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff – (default: infinity)

espressopp.interaction.VerletListStillingerWeberPairTerm(*vl*)

Parameters
vl –

espressopp.interaction.VerletListStillingerWeberPairTerm.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListStillingerWeberPairTerm.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListStillingerWeberPairTerm.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressStillingerWeberPairTerm(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressStillingerWeberPairTerm.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressStillingerWeberPairTerm.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressStillingerWeberPairTerm(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressStillingerWeberPairTerm.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressStillingerWeberPairTerm.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListStillingerWeberPairTerm(*stor*)

Parameters
stor –

espressopp.interaction.CellListStillingerWeberPairTerm.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListStillingerWeberPairTerm(*system*, *vl*, *potential*)

Parameters

- system –

- vl –
- potential –

espressopp.interaction.FixedPairListStillingerWeberPairTerm.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.StillingerWeberPairTerm.StillingerWeberPairTerm(**args*, ***kwds*)
The Lennard-Jones potential.

espressopp.interaction.StillingerWeberPairTermCapped

This class provides methods to compute forces and energies of 2 body term of Stillinger-Weber potential.

If the distance is smaller than the cap-radius:

$$U = A[d_{12}^{-p}(B-1)]e^{\frac{1}{d_{12}-r_c}}$$

where r_c is the cutoff-radius.

espressopp.interaction.StillingerWeberPairTermCapped(*A*, *B*, *p*, *q*, *epsilon*, *sigma*, *cutoff*, *caprad*)

Parameters

- A –
- B –
- p –
- q –
- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff – (default: infinity)
- caprad (real) – (default: 0.0)

espressopp.interaction.VerletListStillingerWeberPairTermCapped(*vl*)

Parameters
vl –

espressopp.interaction.VerletListStillingerWeberPairTermCapped.getCaprad()

Return type

espressopp.interaction.VerletListStillingerWeberPairTermCapped.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListStillingerWeberPairTermCapped.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListStillingerWeberPairTermCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –

- potential –

espressopp.interaction.CellListStillingerWeberPairTermCapped(*stor*)

Parameters
stor –

espressopp.interaction.CellListStillingerWeberPairTermCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListStillingerWeberPairTermCapped(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListStillingerWeberPairTermCapped.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.StillingerWeberPairTermCapped.StillingerWeberPairTermCapped(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.StillingerWeberTripleTerm

This class provides methods to compute forces and energies of the Stillinger Weber Triple Term potential.

if $d_{12} >= r_{c_1}$ or $d_{32} >= r_{c_2}$

$$U = 0.0$$

else

$$U = \varepsilon\lambda e^{\frac{\sigma\gamma_1}{|r_{12}|-\sigma r_{c_1}}} + \frac{\sigma\gamma_2}{|r_{32}|-\sigma r_{c_2}}\left(\frac{r_{12}r_{32}}{|r_{12}|\cdot|r_{32}|} - cos(\theta_0)\right)^2$$

espressopp.interaction.StillingerWeberTripleTerm(*gamma*, *theta0*, *lmbd*, *epsilon*, *sigma*, *cutoff*)

Parameters

- gamma (real) – (default: 0.0)
- theta0 (real) – (default: 0.0)
- lmbd (real) – (default: 0.0)
- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff – (default: infinity)

espressopp.interaction.VerletListStillingerWeberTripleTerm(*system*, *vl3*)

Parameters

- system –
- vl3 –

espressopp.interaction.VerletListStillingerWeberTripleTerm.getPotential(*type1*, *type2*, *type3*)

Parameters

- type1 –
- type2 –
- type3 –

Return type

espressopp.interaction.VerletListStillingerWeberTripleTerm.getVerletListTriple()

Return type
A Python list of lists.

espressopp.interaction.VerletListStillingerWeberTripleTerm.setPotential(*type1*, *type2*, *type3*, *potential*)

Parameters

- type1 –
- type2 –
- type3 –
- potential –

espressopp.interaction.FixedTripleListStillingerWeberTripleTerm(*system*, *ftl*, *potential*)

Parameters

- system –
- ftl –
- potential –

espressopp.interaction.FixedTripleListStillingerWeberTripleTerm.getFixedTripleList()

Return type
A Python list of lists.

espressopp.interaction.FixedTripleListStillingerWeberTripleTerm.setPotential(*type1*, *type2*, *type3*, *potential*)

Parameters

- type1 –
- type2 –
- type3 –
- potential –

class espressopp.interaction.StillingerWeberTripleTerm.StillingerWeberTripleTerm(**args*, ***kwds*)
The StillingerWeberTripleTerm potential.

espressopp.interaction.TersoffPairTerm

This class provides methods to compute forces and energies of 2 body term of Tersoff potential.

if $d_{12} > R + D$

$$U = 0$$

if $d_{12} < R - D$

$$U = Ae^{-\lambda 1 d_{12}}$$

else

$$U = \frac{1}{2}\left(1 - sin\left(\frac{\pi}{4D}\left(d_{12} - R\right)\right)\right) Ae^{-\lambda_1 d_{12}}$$

espressopp.interaction.TersoffPairTerm(*A*, *lambda1*, *R*, *D*, *cutoff*)

Parameters

- A –
- lambda1 –
- R –
- D –
- cutoff – (default: infinity)

espressopp.interaction.VerletListTersoffPairTerm(*vl*)

Parameters

vl –

espressopp.interaction.VerletListTersoffPairTerm.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListTersoffPairTerm.getVerletList()

Return type

A Python list of lists.

espressopp.interaction.VerletListTersoffPairTerm.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListTersoffPairTerm(*stor*)

Parameters

stor –

espressopp.interaction.CellListTersoffPairTerm.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListTersoffPairTerm(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListTersoffPairTerm.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.TersoffPairTerm.TersoffPairTerm(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.TersoffTripleTerm

This class provides methods to compute forces and energies of the Tersoff Triple Term potential.

$$U = f_{C_j} f_A \left(1 + \left(\beta f_{C_k} \gamma \left(1 + \frac{c_2}{d_2} - \frac{c_2}{d_2 + \left(\frac{r_{12} r_{32}}{|r_{12}||r_{32}|} - cos(\theta_0) \right)^2} \right) \left(e^{\lambda_3 (|r_{12}| - |r_{32}|)} \right)^m \right)^n \right)^{-\frac{1}{2n}}$$

espressopp.interaction.VerletListTersoffTripleTerm(*system*, *vl3*)

Parameters

- system –
- vl3 –

espressopp.interaction.VerletListTersoffTripleTerm.getPotential(*type1*, *type2*, *type3*)

Parameters

- type1 –
- type2 –
- type3 –

Return type

espressopp.interaction.VerletListTersoffTripleTerm.getVerletListTriple()

Return type
A Python list of lists.

espressopp.interaction.VerletListTersoffTripleTerm.setPotential(*type1*, *type2*, *type3*, *potential*)

Parameters

- type1 –
- type2 –
- type3 –
- potential –

espressopp.interaction.FixedTripleListTersoffTripleTerm(*system*, *ftl*, *potential*)

Parameters

- system –
- ftl –
- potential –

espressopp.interaction.FixedTripleListTersoffTripleTerm.getFixedTripleList()

Return type
A Python list of lists.

espressopp.interaction.FixedTripleListTersoffTripleTerm.setPotential(*type1*, *type2*, *type3*, *potential*)

Parameters

- type1 –
- type2 –
- type3 –
- potential –

3.7.7 Pair

espressopp.interaction.GravityTruncated

This is an implementation of a truncated (cutoff) Gravity Potential

$$U = P \cdot \frac{m_1 \cdot m_2}{|p_1 - p_2|}$$

where m_i is the mass of the *i* th particle, p_i its position and *P* a prefactor.

espressopp.interaction.GravityTruncated(*prefactor*, *cutoff*)

Parameters

- prefactor (real) – (default: 1.0)
- cutoff – (default: infinity)

espressopp.interaction.VerletListGravityTruncated(*vl*)

Parameters
vl –

espressopp.interaction.VerletListGravityTruncated.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListGravityTruncated.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListGravityTruncated.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.LennardJones

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

espressopp.interaction.LennardJones(*epsilon*, *sigma*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff (real or "infinity") – (default: infinity)
- shift (real or "auto") – (default: “auto”)

espressopp.interaction.VerletListLennardJones(*vl*)

Defines a verletlist-based interaction using a Lennard-Jones potential.

Parameters
vl (std::shared_ptr<VerletList>) – Verletlist object

espressopp.interaction.VerletListLennardJones.getPotential(*type1*, *type2*)

Gets the LennardJones interaction potential for interacting particles of type1 and type2..

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<LennardJones>

espressopp.interaction.VerletListLennardJones.getVerletList()

Gets the verletlist used in VerletListLennardJones interaction.

Return type
std::shared_ptr<VerletList>

espressopp.interaction.VerletListLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressLennardJones(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using a LennardJones potential for the AT and a tabulated potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressLennardJones.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressLennardJones.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressLennardJones2(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using a LennardJones potential for both the AT and the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressLennardJones2.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1

- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressLennardJones2.setPotentialCG(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressLennardJonesHarmonic(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using a LennardJones potential for the AT and a Harmonic potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressLennardJonesHarmonic.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressLennardJonesHarmonic.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Harmonic interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListHadressLennardJones(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using a LennardJones potential for the AT and a tabulated potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressLennardJones.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressLennardJones.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressLennardJones2(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using a LennardJones potential for both the AT and the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressLennardJones2.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressLennardJones2.setPotentialCG(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressLennardJonesHarmonic(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using a LennardJones potential for the AT and a Harmonic potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressLennardJonesHarmonic.setPotentialAT(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressLennardJonesHarmonic.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Harmonic interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.CellListLennardJones(*stor*)

Defines a CellList-based interaction using a LennardJones potential.

Parameters

stor (std::shared_ptr <storage::Storage>) – storage object

espressopp.interaction.CellListLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the LennardJones interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.FixedPairListLennardJones(*system*, *vl*, *potential*)

Defines a FixedPairList-based interaction using a LennardJones potential.

Parameters

- system (std::shared_ptr<System>) – system object
- vl (std::shared_ptr<FixedPairList>) – FixedPairList object
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.FixedPairListLennardJones.getFixedPairList()

Gets the FixedPairList.

Return type

std::shared_ptr<FixedPairList>

espressopp.interaction.FixedPairListLennardJones.getPotential()

Gets the LennardJones interaction potential.

Return type

std::shared_ptr<LennardJones>

espressopp.interaction.FixedPairListLennardJones.setFixedPairList(*fixedpairlist*)

Sets the FixedPairList.

Parameters

fixedpairlist (std::shared_ptr<FixedPairList>) – FixedPairList object

espressopp.interaction.FixedPairListLennardJones.setPotential(*potential*)

Sets the LennardJones interaction potential.

Parameters

potential (std::shared_ptr<LennardJones>) – tabulated potential object

espressopp.interaction.VerletListAdressATLennardJones(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based AdResS interaction using a LennardJones potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListAdressATLennardJones interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressATLennardJones.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListAdressATLennardJones interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<LennardJones>

espressopp.interaction.VerletListAdressATLennardJones.getVerletList()

Gets the verletlist used in VerletListAdressATLennardJones interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListHadressATLennardJones(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based H-AdResS interaction using a LennardJones potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListHadressATLennardJones interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressATLennardJones.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListHadressATLennardJones interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<LennardJones>

espressopp.interaction.VerletListHadressATLennardJones.getVerletList()

Gets the verletlist used in VerletListHadressATLennardJones interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListAdressCGLennardJones(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based AdResS interaction using a LennardJones potential for the CG interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressCGLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListAdressCGLennardJones interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressCGLennardJones.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListAdressCGLennardJones interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<LennardJones>

espressopp.interaction.VerletListAdressCGLennardJones.getVerletList()

Gets the verletlist used in VerletListAdressCGLennardJones interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListHadressCGLennardJones(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based H-AdResS interaction using a LennardJones potential for the CG interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressCGLennardJones.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListHadressCGLennardJones interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressCGLennardJones.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListHadressCGLennardJones interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<LennardJones>

espressopp.interaction.VerletListHadressCGLennardJones.getVerletList()

Gets the verletlist used in VerletListHadressCGLennardJones interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListAdressATLenJonesReacFieldGen(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction (this is implemented with a separate template to avoid looping twice over the particle pairs when using both a Lennard Jones and an electrostatic interaction).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATLenJonesReacFieldGen.setPotential1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListAdressATLenJonesReacFieldGen interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressATLenJonesReacFieldGen.setPotential2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListAdressATLenJonesReacFieldGen interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListHadressATLenJonesReacFieldGen(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based H-AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction (this is implemented with a separate template to avoid looping twice over the particle pairs when using both a Lennard Jones and an electrostatic interaction).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATLenJonesReacFieldGen.setPotential1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListHadressATLenJonesReacFieldGen interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressATLenJonesReacFieldGen.setPotential2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListHadressATLenJonesReacFieldGen interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenTab(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction and a Tabulated potential for the CG interaction (this is implemented with a separate template to avoid looping repeatedly over the particle pairs when using several interactions).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPotentialAT1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListAdressATLJReacFieldGenTab interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPotentialAT2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListAdressATLJReacFieldGenTab interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated CG potential in VerletListAdressATLJReacFieldGenTab interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenTab(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction and a Tabulated potential for the CG interaction (this is implemented with a separate template to avoid looping repeatedly over the particle pairs when using several interactions).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotentialAT1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListHadressATLJReacFieldGenTab interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotentialAT2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListHadressATLJReacFieldGenTab interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated CG potential in VerletListHadressATLJReacFieldGenTab interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction and a Harmonic potential for the CG interaction (this is implemented with a separate template to avoid looping repeatedly over the particle pairs when using several interactions).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic.setPotentialAT1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListAdressATLJReacFieldGenHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic.setPotentialAT2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListAdressATLJReacFieldGenHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Harmonic CG potential in VerletListAdressATLJReacFieldGenHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using both a LennardJones potential and a ReactionFieldGeneralized potential for the AT interaction and a Harmonic potential for the CG interaction (this is implemented with a separate template to avoid looping repeatedly over the particle pairs when using several interactions).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotentialAT1(*type1*, *type2*, *potential*)

Sets the LennardJones AT potential in VerletListHadressATLJReacFieldGenHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<LennardJones>) – LennardJones potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotentialAT2(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized AT potential in VerletListHadressATLJReacFieldGenHarmonic interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Harmonic CG potential in VerletListHadressATLJReacFieldGenHarmonic interaction for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Harmonic>) – Harmonic potential object

class espressopp.interaction.LennardJones.LennardJones(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.LennardJonesAutoBonds

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

espressopp.interaction.LennardJonesAutoBonds(*epsilon*, *sigma*, *cutoff*, *bondlist*, *maxcrosslinks*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)

- cutoff – (default: infinity)
- bondlist – (default: None)
- maxcrosslinks (int) – (default: 2)

espressopp.interaction.VerletListLennardJonesAutoBonds(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesAutoBonds.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesAutoBonds.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListLennardJonesAutoBonds.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesAutoBonds(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLennardJonesAutoBonds.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesAutoBonds(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressLennardJonesAutoBonds.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –

- potential –

espressopp.interaction.CellListLennardJonesAutoBonds(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesAutoBonds.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesAutoBonds(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesAutoBonds.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesAutoBonds.LennardJonesAutoBonds(**args*, ***kwds*)
The Lennard-Jones auto bonds potential.

espressopp.interaction.LennardJonesCapped

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

where *r* is either the distance or the capped distance, depending on which is greater.

espressopp.interaction.LennardJonesCapped(*epsilon*, *sigma*, *cutoff*, *caprad*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff – (default: infinity)
- caprad (real) – (default: 0.0)
- shift – (default: "auto")

espressopp.interaction.VerletListLennardJonesCapped(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesCapped.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLennardJonesCapped.getPotentialAT(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListAdressLennardJonesCapped.getPotentialCG(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListAdressLennardJonesCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressLennardJonesCapped.getPotentialAT(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListHadressLennardJonesCapped.getPotentialCG(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListHadressLennardJonesCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListLennardJonesCapped(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesCapped.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.CellListLennardJonesCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –

- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesCapped(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesCapped.getPotential()

Return type

espressopp.interaction.FixedPairListLennardJonesCapped.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesCapped.LennardJonesCapped(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.LennardJonesEnergyCapped

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

where *r* is either the distance or the capped distance, depending on which is greater.

espressopp.interaction.LennardJonesEnergyCapped(*epsilon*, *sigma*, *cutoff*, *caprad*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- cutoff – (default: infinity)
- caprad (real) – (default: 0.0)
- shift – (default: “auto”)

espressopp.interaction.VerletListLennardJonesEnergyCapped(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesEnergyCapped.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesEnergyCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesEnergyCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.getPotentialAT(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.getPotentialCG(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesEnergyCapped(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.getPotentialAT(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.getPotentialCG(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListLennardJonesEnergyCapped(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesEnergyCapped.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.CellListLennardJonesEnergyCapped.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesEnergyCapped(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesEnergyCapped.getPotential()

Return type

espressopp.interaction.FixedPairListLennardJonesEnergyCapped.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesEnergyCapped.LennardJonesEnergyCapped(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.LennardJonesExpand

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]$$

espressopp.interaction.LennardJonesExpand(*epsilon*, *sigma*, *delta*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- delta (real) – (default: 0.0)
- cutoff – (default: infinity)
- shift – (default: "auto")

espressopp.interaction.VerletListLennardJonesExpand(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesExpand.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesExpand.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –

- potential –

espressopp.interaction.CellListLennardJonesExpand(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesExpand.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesExpand(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesExpand.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesExpand.LennardJonesExpand(**args*, ***kwds*)

The LennardJonesExpand potential.

espressopp.interaction.LennardJonesGeneric

This class provides methods to compute forces and energies of a generic Lennard Jones potential with arbitrary integers a and b.

$$V(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^a - \left(\frac{\sigma}{r}\right)^b \right]$$

espressopp.interaction.LennardJonesGeneric(*epsilon*, *sigma*, *a*, *b*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- a (int) – (default: 12)
- b (int) – (default: 6)
- cutoff – (default: infinity)
- shift – (default: “auto”)

espressopp.interaction.VerletListLennardJonesGeneric(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesGeneric.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesGeneric.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListLennardJonesGeneric.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesGeneric(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLennardJonesGeneric2(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLennardJonesGeneric2.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –

- potential –

espressopp.interaction.VerletListAdressLennardJonesGeneric2.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesGeneric(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressLennardJonesGeneric.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesGeneric.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesGeneric2(*vl*, *fixedtupleList*, *KTI*)

Parameters

- vl –
- fixedtupleList –
- KTI – (default: False)

espressopp.interaction.VerletListHadressLennardJonesGeneric2.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLennardJonesGeneric2.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListLennardJonesGeneric(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesGeneric.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesGeneric(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesGeneric.getFixedPairList()

Return type
A Python list of lists.

espressopp.interaction.FixedPairListLennardJonesGeneric.getPotential()

Return type

espressopp.interaction.FixedPairListLennardJonesGeneric.setFixedPairList(*fixedpairlist*)

Parameters
fixedpairlist –

espressopp.interaction.FixedPairListLennardJonesGeneric.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesGeneric.LennardJonesGeneric(**args*, ***kwds*)
The generic Lennard-Jones potential.

espressopp.interaction.LennardJonesGromacs

if $d^2 > r_1^2$

$$U = 4\varepsilon(\frac{sigma^{12}}{d^{12}} - \frac{sigma^6}{d^6}) + (d - r_1)^3(ljsw3 + ljsw4(d - r_1) + ljsw5)$$

else

$$U = 4\varepsilon(\frac{\sigma^{12}}{d^{12}} - \frac{\sigma^6}{d^6})$$

espressopp.interaction.LennardJonesGromacs(*epsilon*, *sigma*, *r1*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- r1 (real) – (default: 0.0)
- cutoff – (default: infinity)
- shift – (default: "auto")

espressopp.interaction.VerletListLennardJonesGromacs(*vl*)

Parameters
vl –

espressopp.interaction.VerletListLennardJonesGromacs.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLennardJonesGromacs.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListLennardJonesGromacs(*stor*)

Parameters
stor –

espressopp.interaction.CellListLennardJonesGromacs.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLennardJonesGromacs(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLennardJonesGromacs.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJonesGromacs.LennardJonesGromacs(**args*, ***kwds*)

The LennardJonesGromacs potential.

espressopp.interaction.LennardJonesSoftcoreTI

This module is for performing simulations (e.g. as part of Thermodynamic Integration) where some Lennard-Jones interactions are a function of a parameter λ, used to construct a pathway between states A and B.

For those interactions which are a function of λ, the potential is softcore Lennard Jones with the following form:

$$U_S(r_{ij}, \lambda) = (1-\lambda)U_H^A(r_A) + \lambda U_H^B(r_B)$$
$$r_A = (\alpha\sigma_A^6\lambda^p + r_{ij}^6)^{1/6}$$
$$r_B = (\alpha\sigma_B^6(1-\lambda)^p + r_{ij}^6)^{1/6}$$

where ϵ_A, ϵ_B, σ_A and σ_B are the parameters of states A and B, and α and p are adjustable parameters of the softcore potential. The potentials $U_H^A(r_A)$ and $U_H^B(r_B)$ are the normal Lennard-Jones 12-6 hardcore potentials:

$$U_H^A(r_A) = 4.0\epsilon_A(\frac{\sigma_A}{r_A}^{12} - \frac{\sigma_A}{r_A}^{6})$$

The user specifies a list of particles, pidlist. For all pairs of particles with particletypes interacting via this potential, the LJ interaction between two particles i and j is calculated as follows:

if (i not in pidlist) and (j not in pidlist):
U_H^A (full state A hardcore LJ interaction)

if (i in pidlist) and (j in pidlist):

if annihilate==True:
U_S (softcore LJ interaction, function of lambda)

if annihilate==False:
U_H^A (full state A hardcore LJ interaction)

if (i in pidlist) xor (j in pidlist):
U_S (softcore LJ interaction, function of lambda)

The default is annihilation (interactions within pidlist are coupled to lambda, and cross-interactions between particles in pidlist and particles in the rest of the system are also coupled to lambda). The alternative is decoupling (only cross-interactions between particles in pidlist and particles in the rest of the system are coupled to lambda. Interactions within pidlist are not affected by the value of lambda.) If annihilation==False, then decoupling is performed. See: http://www.alchemistry.org/wiki/Decoupling_and_annihilation

Exclusions apply as normal, i.e. interactions are only calculated for pairs of particles not already excluded.

This class does not do any automatic shifting of the potential.

So far only VerletListAdressLennardJonesSoftcoreTI is implemented, however VerletListLennardJonesSoftcoreTI, VerletListHadressLennardJonesSoftcoreTI, etc. can also be easily implemented.

The λ (lambdaTI) parameter used here should not be confused with the λ (lambda_adr) particle property used in AdResS simulations.

See also the Thermodynamic Integration tutorial.

Example python script:

```
>>> #value of lambda
>>> lambdaTI = 0.3
>>> #softcore parameters
>>> alphaSC = 0.5
>>> powerSC = 1.0
>>> #make list of indices of particles whose LJ parameters are different in TI states A and B
>>> pidlist = [1,2,3,4]
>>> #create interaction using VerletListAdress object and FixedTupleListAdress object
>>> lj_adres_interaction=espressopp.interaction.VerletListAdressLennardJonesSoftcoreTI(verletlist,
↪ftpl)
>>> #loop over list of all types for particles interacting with this atomistic potential
>>> for i in types:
>>>   for k in types:
>>>     ljpot = espressopp.interaction.LennardJonesSoftcoreTI(epsilonA=epsA[i][k], sigmaA=sigA[i][k],
↪epsilonB=epsB[i][k], sigmaB=sigB[i][k], alpha=alphaSC, power=powerSC, cutoff=cutoff,
↪lambdaTI=lambdaTI, annihilate=False)
>>>     ljpot.addPids(pidlist)
>>>     lj_adres_interaction.setPotentialAT(type1=i, type2=k, potential=ljpot)
>>> system.addInteraction(lj_adres_interaction)
```

During the MD run, one can then calculate the derivative of the RF energy wrt lambda

```
>>> #calculate dU/dlambda
>>> dUdl = lj_adres_interaction.computeEnergyDeriv()
```

espressopppp.interaction.LennardJonesSoftcoreTI(*epsilonA*, *sigmaA*, *epsilonB*, *sigmaB*, *alpha*, *power*, *cutoff*, *lambdaTI*, *annihilate*)

Parameters

- epsilonA (real) – (default: 1.0) LJ interaction parameter
- sigmaA (real) – (default: 1.0) LJ interaction parameter
- epsilonB (real) – (default: 0.0) LJ interaction parameter
- sigmaB (real) – (default: 1.0) LJ interaction parameter
- alpha (real) – (default: 1.0) softcore parameter
- power (real) – (default: 1.0) softcore parameter
- cutoff (real) – (default: infinity) interaction cutoff
- lambdaTI (real) – (default: 0.0) TI lambda parameter
- annihilate (bool) – (default: True) switch between annihilation and decoupling

espressopppp.interaction.LennardJonesSoftcoreTI.addPids(*pidlist*)

Parameters

pidlist (python list) – list of particle ids of particles whose interaction parameters differ in state A and B

espressopppp.interaction.VerletListAdressLennardJones(*vl*, *fixedtupleList*)

Parameters

- vl (VerletListAdress object) – Verlet list

- fixedtupleList (FixedTupleListAdress object) – list of tuples describing mapping between CG and AT particles

espressopppp.interaction.VerletListAdressLennardJones.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 (int) – atomtype
- type2 (int) – atomtype
- potential (Potential) – espressopppp potential

espressopppp.interaction.VerletListAdressLennardJones.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 (int) – atomtype
- type2 (int) – atomtype
- potential (Potential) – espressopppp potential

class espressopp.interaction.LennardJonesSoftcoreTI.LennardJonesSoftcoreTI(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.LJcos

if $r^2 \leq border_{pot}$, then:

$$U = 4(\frac{1}{r^{12}} - \frac{1}{r^6}) + 1 - \phi$$

else:

$$U = \frac{1}{2}\phi(cos(\alpha r^2 + \beta) - 1)$$

espressopp.interaction.LJcos(*phi*)

Parameters

phi (real) – (default: 1.0)

espressopp.interaction.VerletListLJcos(*vl*)

Parameters

vl –

espressopp.interaction.VerletListLJcos.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListLJcos.getVerletList()

Return type

A Python list of lists.

espressopp.interaction.VerletListLJcos.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLJcos(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListAdressLJcos.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressLJcos.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLJcos(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressLJcos.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressLJcos.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListLJcos(*stor*)

Parameters
stor –

espressopp.interaction.CellListLJcos.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListLJcos(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListLJcos.getFixedPairList()

Return type
A Python list of lists.

espressopp.interaction.FixedPairListLJcos.setFixedPairList(*fixedpairlist*)

Parameters
fixedpairlist –

espressopp.interaction.FixedPairListLJcos.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LJcos.LJcos(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.MirrorLennardJones

This class provides methods to compute forces and energies of the Mirror Lennard-Jones potential.

$$V(r) = V_{LJ}(r_m - |r - r_m|)$$

where V_{LJ} is the 6-12 purely repulsive Lennard-Jones potential. This potential is introduced in R.L.C. Akkermans, S. Toxvaerd and & W. J. Briels. Molecular dynamics of polymer growth. The Journal of Chemical Physics, 1998, 109, 2929-2940.

espressopp.interaction.MirrorLennardJones(*epsilon*, *sigma*)

Parameters

- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 0.0)

espressopp.interaction.FixedPairListMirrorLennardJones(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListMirrorLennardJones.getFixedPairList()

Return type
A Python list of lists.

espressopp.interaction.FixedPairListMirrorLennardJones.getPotential()

Return type

espressopp.interaction.FixedPairListMirrorLennardJones.setFixedPairList(*fixedpairlist*)

Parameters
fixedpairlist –

espressopp.interaction.FixedPairListMirrorLennardJones.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.MirrorLennardJones.MirrorLennardJones(**args*, ***kwds*)
The MirrorLennardJones potential.

3.7.8 Tabulated

espressopp.interaction.Tabulated

espressopp.interaction.Tabulated(*itype*, *filename*, *cutoff*)
Defines a tabulated potential.

Parameters

- itype (int) – interpolation type (1,2, or 3 for linear, Akima, or cubic splines)
- filename (string) – table filename
- cutoff (real or "infinity") – (default: infinity) interaction cutoff

espressopp.interaction.VerletListAdressTabulated(*vl*, *fixedtupleList*)
Defines a verletlist-based AdResS interaction using tabulated potentials for both AT and CG interactions.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressTabulated.setPotentialAT(*type1*, *type2*, *potential*)
Sets the AT potential in VerletListAdressTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1

- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressTabulated.setPotentialCG(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListAdressTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressCGTabulated(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based AdResS interaction using a tabulated potential for the CG interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressCGTabulated.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListAdressCGTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressCGTabulated.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListAdressCGTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Tabulated>

espressopp.interaction.VerletListAdressCGTabulated.getVerletList()

Gets the verletlist used in VerletListAdressCGTabulated interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListHadressTabulated(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using tabulated potentials for both AT and CG interactions.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressTabulated.setPotentialAT(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListHadressTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressTabulated.setPotentialCG(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListHadressTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressCGTabulated(*vl*, *fixedtupleList*)

Defines only the CG part of a verletlist-based H-AdResS interaction using a tabulated potential for the CG interaction. It's defined as a "NonbondedSlow" interaction (which multiple time stepping integrators can make use of).

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressCGTabulated.setPotential(*type1*, *type2*, *potential*)

Sets the CG potential in VerletListHadressCGTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressCGTabulated.getPotential(*type1*, *type2*)

Gets the CG potential in VerletListHadressCGTabulated interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Tabulated>

espressopp.interaction.VerletListHadressCGTabulated.getVerletList()

Gets the verletlist used in VerletListHadressCGTabulated interaction.

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListTabulated(*vl*)

Defines a verletlist-based interaction using a tabulated potential.

Parameters
vl (std::shared_ptr<VerletList>) – Verletlist object

espressopp.interaction.VerletListTabulated.getPotential(*type1*, *type2*)

Gets the tabulated interaction potential in VerletListTabulated for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<Tabulated>

espressopp.interaction.VerletListTabulated.setPotential(*type1*, *type2*, *potential*)

Sets the tabulated interaction potential in VerletListTabulated for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.CellListTabulated(*stor*)

Defines a CellList-based interaction using a tabulated potential.

Parameters
stor (std::shared_ptr <storage::Storage>) – storage object

espressopp.interaction.CellListTabulated.setPotential(*type1*, *type2*, *potential*)

Sets the tabulated interaction potential in CellListTabulated for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.FixedPairListTabulated(*system*, *vl*, *potential*)

Defines a FixedPairList-based interaction using a tabulated potential.

Parameters

- system (std::shared_ptr<System>) – system object
- vl (std::shared_ptr<FixedPairList>) – FixedPairList list object
- potential (std::shared_ptr<Tabulated>) – tabulated potential object

espressopp.interaction.FixedPairListTabulated.setPotential(*potential*)

Sets the tabulated interaction potential in FixedPairListTabulated for interacting particles.

Parameters
potential (std::shared_ptr<Tabulated>) – tabulated potential object

espressopp.interaction.FixedPairListTypesTabulated(*system*, *fpl*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- fpl (espressopp.FixedPairList) – The FixedPairList.

espressopp.interaction.FixedPairListTypesTabulated.setPotential(*type1*, *type2*, *potential*)

Defines bond potential for interaction between particles of types type1-type2-type3.

Parameters

- type1 (int) – Type of particle 1.
- type2 (int) – Type of particle 2.
- potential (espressopp.interaction.Potential) – The potential to set up.

espressopp.interaction.FixedPairListPIadressTabulated(*system*, *fpl*, *fixedtupleList*, *potential*, *ntrotter*, *speedup*)

Defines tabulated bonded pair potential for interactions based on the fixedtuplelist in the context of Path Integral AdResS. When the speedup flag is set, it will use only the centroids in the classical region, otherwise all Trotter beads. In the quantum region, always all Trotter beads are used.

Parameters

- system (espressopp.System) – The Espresso++ system object.
- fpl (espressopp.FixedPairList) – The FixedPairList.
- fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.
- potential (espressopp.interaction.Potential) – The potential.
- ntrotter (int) – The Trotter number.
- speedup (bool) – Boolean flag to decide whether to use centroids in classical region or all Trotter beads

espressopp.interaction.FixedPairListPIadressTabulated.setPotential(*potential*)

Sets the potential.

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.FixedPairListPIadressTabulated.getPotential()

Gets the potential.

Returns
the potential

Return type
std::shared_ptr < Potential >

espressopp.interaction.FixedPairListPIadressTabulated.setFixedPairList(*fpl*)

Sets the FixedPairList.

Parameters
fpl (espressopp.FixedPairList) – The FixedPairList object.

espressopp.interaction.FixedPairListPIadressTabulated.getFixedPairList()

Gets the FixedPairList.

Returns
the FixedPairList

Return type
std::shared_ptr < FixedPairList >

espressopp.interaction.FixedPairListPIadressTabulated.setFixedTupleList(*fixedtupleList*)

Sets the FixedTupleList.

Parameters
fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.

espressopp.interaction.FixedPairListPIadressTabulated.getFixedTupleList()

Gets the FixedTupleList.

Returns
the FixedTupleList

Return type
std::shared_ptr < FixedTupleListAdress >

espressopp.interaction.FixedPairListPIadressTabulated.setNTrotter(*ntrotter*)

Sets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.FixedPairListPIadressTabulated.getNTrotter()

Gets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.FixedPairListPIadressTabulated.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – The speedup flag.

espressopp.interaction.FixedPairListPIadressTabulated.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

espressopp.interaction.VerletListPIadressTabulated(*vl*, *fixedtupleList*, *ntrotter*, *speedup*)

Defines a non-bonded interaction using an adaptive resolution VerletList in the context of Path Integral AdResS. Two different tabulated potentials can be specified: one, which is used in the quantum region, the other one in the classical region. The interpolation proceeds according to the Path Integral AdResS scheme (see J. Chem. Phys 147, 244104 (2017)). When the speedup flag is set,it will use only the centroids in the classical region, otherwise all Trotter beads. In the quantum region, always all Trotter beads are used.

Parameters

- vl (espressopp.VerletListAdress) – The AdResS VerletList.
- fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.
- ntrotter (int) – The Trotter number.
- speedup (bool) – Boolean flag to decide whether to use centroids in classical region or all Trotter beads

espressopp.interaction.VerletListPIadressTabulated.setPotentialQM(*potential*)

Sets the potential for the quantum region (has to be a tabulated one).

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.VerletListPIadressTabulated.setPotentialCL(*potential*)

Sets the potential for the classical region (has to be a tabulated one).

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.VerletListPIadressTabulated.setVerletList(*vl*)

Sets the VerletList.

Parameters
vl (espressopp.VerletListAdress) – The VerletListAdress object.

espressopp.interaction.VerletListPIadressTabulated.getVerletList()

Gets the VerletList.

Returns
the Adress VerletList

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListPIadressTabulated.setFixedTupleList(*fixedtupleList*)

Sets the FixedTupleList.

Parameters
fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.

espressopp.interaction.VerletListPIadressTabulated.getFixedTupleList()

Gets the FixedTupleList.

Returns
the FixedTupleList

Return type
std::shared_ptr < FixedTupleListAdress >

espressopp.interaction.VerletListPIadressTabulated.setNTrotter(*ntrotter*)

Sets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.VerletListPIadressTabulated.getNTrotter()

Gets the Trotter number NTrotter.

Returns
the Trotter number

Return type
int

espressopp.interaction.VerletListPIadressTabulated.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – The speedup flag.

espressopp.interaction.VerletListPIadressTabulated.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

espressopp.interaction.VerletListPIadressTabulatedLJ(*vl*, *fixedtupleList*, *ntrotter*, *speedup*)

Defines a non-bonded interaction using an adaptive resolution VerletList in the context of Path Integral AdResS. Two different potentials can be specified: one, which is used in the quantum region (tabulated), the other one in the classical region (Lennard-Jones type). The interpolation proceeds according to the Path Integral AdResS scheme (see J. Chem. Phys 147, 244104 (2017)). When the speedup flag is set,it will use only the centroids in the classical region, otherwise all Trotter beads. In the quantum region, always all Trotter beads are used.

Parameters

- vl (espressopp.VerletListAdress) – The AdResS VerletList.
- fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.
- ntrotter (int) – The Trotter number.
- speedup (bool) – Boolean flag to decide whether to use centroids in classical region or all Trotter beads

espressopp.interaction.VerletListPIadressTabulatedLJ.setPotentialQM(*potential*)

Sets the potential for the quantum region (has to be a tabulated one).

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.VerletListPIadressTabulatedLJ.setPotentialCL(*potential*)

Sets the potential for the classical region (has to be a Lennard-Jones type one).

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.VerletListPIadressTabulatedLJ.setVerletList(*vl*)

Sets the VerletList.

Parameters
vl (espressopp.VerletListAdress) – The VerletListAdress object.

espressopp.interaction.VerletListPIadressTabulatedLJ.getVerletList()

Gets the VerletList.

Returns
the Adress VerletList

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListPIadressTabulatedLJ.setFixedTupleList(*fixedtupleList*)

Sets the FixedTupleList.

Parameters
fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.

espressopp.interaction.VerletListPIadressTabulatedLJ.getFixedTupleList()

Gets the FixedTupleList.

Returns
the FixedTupleList

Return type
std::shared_ptr < FixedTupleListAdress >

espressopp.interaction.VerletListPIadressTabulatedLJ.setNTrotter(*ntrotter*)

Sets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.VerletListPIadressTabulatedLJ.getNTrotter()

Gets the Trotter number NTrotter.

Returns
the Trotter number

Return type
int

espressopp.interaction.VerletListPIadressTabulatedLJ.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – The speedup flag.

espressopp.interaction.VerletListPIadressTabulatedLJ.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

espressopp.interaction.VerletListPIadressNoDriftTabulated(*vl*, *fixedtupleList*, *ntrotter*, *speedup*)

Defines a non-bonded interaction using an adaptive resolution VerletList in the context of Path Integral AdResS. One tabulated potential can be specified, which is used thoughout the whole system. Hence, only the quantumness of the particles changes, but not the forcefield (see J. Chem. Phys 147, 244104 (2017)). When the speedup flag is set,it will use only the centroids in the classical region, otherwise all Trotter beads. In the quantum region, always all Trotter beads are used.

Parameters

- vl (espressopp.VerletListAdress) – The AdResS VerletList.
- fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.
- ntrotter (int) – The Trotter number.
- speedup (bool) – Boolean flag to decide whether to use centroids in classical region or all Trotter beads

espressopp.interaction.VerletListPIadressNoDriftTabulated.setPotential(*potential*)

Sets the potential which is used throughout the whole system (has to be a tabulated one).

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.VerletListPIadressNoDriftTabulated.setVerletList(*vl*)

Sets the VerletList.

Parameters
vl (espressopp.VerletListAdress) – The VerletListAdress object.

espressopp.interaction.VerletListPIadressNoDriftTabulated.getVerletList()

Gets the VerletList.

Returns
the Adress VerletList

Return type
std::shared_ptr<VerletListAdress>

espressopp.interaction.VerletListPIadressNoDriftTabulated.setFixedTupleList(*fixedtupleList*)

Sets the FixedTupleList.

Parameters
fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.

espressopp.interaction.VerletListPIadressNoDriftTabulated.getFixedTupleList()

Gets the FixedTupleList.

Returns
the FixedTupleList

Return type
std::shared_ptr < FixedTupleListAdress >

espressopp.interaction.VerletListPIadressNoDriftTabulated.setNTrotter(*ntrotter*)

Sets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.VerletListPIadressNoDriftTabulated.getNTrotter()

Gets the Trotter number NTrotter.

Returns
the Trotter number

Return type
int

espressopp.interaction.VerletListPIadressNoDriftTabulated.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – The speedup flag.

espressopp.interaction.VerletListPIadressNoDriftTabulated.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

class espressopp.interaction.Tabulated.Tabulated(**args*, ***kwds*)

The Tabulated potential.

espressopp.interaction.TabulatedAngular

espressopp.interaction.TabulatedAngular(*itype*, *filename*)

Parameters

- itype (int) – The interpolation type: 1 - linear, 2 - akima spline, 3 - cubic spline
- filename (str) – The tabulated potential filename.

espressopp.interaction.FixedTripleListTabulatedAngular(*system*, *ftl*, *potential*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- ftl (espressopp.FixedTripleList) – The FixedTripleList.
- potential (espressopp.interaction.Potential) – The potential.

espressopp.interaction.FixedTripleListTabulatedAngular.setPotential(*potential*)

Parameters

potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.FixedTripleListTypesTabulatedAngular(*system*, *ftl*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- ftl (espressopp.FixedTripleList) – The FixedTriple list.

espressopp.interaction.FixedTripleListTypesTabulatedAngular.setPotential(*type1*, *type2*, *type3*, *potential*)

Defines angular potential for interaction between particles of types type1-type2-type3.

Parameters

- type1 (int) – Type of particle 1.
- type2 (int) – Type of particle 2.
- type3 (int) – Type of particle 3.
- potential (espressopp.interaction.AngularPotential) – The potential to set up.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular(*system*, *ftl*, *fixedtupleList*, *potential*, *ntrotter*, *speedup*)

Defines tabulated angular potential for interactions based on the fixedtuplelist in the context of Path Integral AdResS. When the speedup flag is set, it will use only the centroids in the classical region, otherwise all Trotter beads. In the quantum region, always all Trotter beads are used.

Parameters

- system (espressopp.System) – The Espresso++ system object.
- ftl (espressopp.FixedTripleList) – The FixedTripleList.
- fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.
- potential (espressopp.interaction.Potential) – The potential.
- ntrotter (int) – The Trotter number.
- speedup (bool) – Boolean flag to decide whether to use centroids in classical region or all Trotter beads

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setPotential(*potential*)

Sets the potential.

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getPotential()

Gets the potential.

Returns
the potential

Return type
std::shared_ptr < Potential >

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setFixedTripleList(*ftl*)

Sets the FixedTripleList.

Parameters
ftl (espressopp.FixedTripleList) – The FixedTripleList object.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getFixedTripleList()

Gets the FixedTripleList.

Returns
the FixedTripleList

Return type
std::shared_ptr < FixedTripleList >

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setFixedTupleList(*fixedtupleList*)

Sets the FixedTupleList.

Parameters
fixedtupleList (espressopp.FixedTupleListAdress) – The FixedTupleListAdress object.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getFixedTupleList()

Gets the FixedTupleList.

Returns
the FixedTupleList

Return type
std::shared_ptr < FixedTupleListAdress >

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setNTrotter(*ntrotter*)

Sets the Trotter number NTrotter.

Parameters
ntrotter (int) – The Trotter number.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getNTrotter()

Gets the Trotter number NTrotter.

Returns
the Trotter number

Return type
int

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setSpeedup(*speedup*)

Sets the speedup flag.

Parameters
speedup (bool) – The speedup flag.

espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getSpeedup()

Gets the speedup flag.

Returns
the speedup flag

Return type
bool

class espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngularLocal(*system*, *ftl*, *fixedtupleList*, *potential*, *ntrotter*, *speedup*)

The (local) tanulated angular interaction using FixedTriple lists.

getFixedTripleList(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*) → FixedTripleList :

C++ signature :
std::shared_ptr<espressopp::FixedTripleList> getFixedTripleList(espressopp::interaction::FixedTripleListPIadressIntera
{lvalue})

getFixedTupleList(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*) → FixedTupleListAdress :

C++ signature :
std::shared_ptr<espressopp::FixedTupleListAdress> getFixedTupleList(espressopp::interaction::FixedTripleListPIadress
{lvalue})

getNTrotter(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*) → int :

C++ signature :
int getNTrotter(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::Tabulated
{lvalue})

getPotential(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*) → interaction_TabulatedAngular :

C++ signature :
std::shared_ptr<espressopp::interaction::TabulatedAngular> getPotential(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::TabulatedAngular>
{lvalue})

getSpeedup(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*) → bool :

C++ signature :
bool getSpeedup(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::Tabulat
{lvalue})

setFixedTripleList(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*, *(FixedTripleList)arg2*) → None :

C++ signature :
void setFixedTripleList(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::T
{lvalue},std::shared_ptr<espressopp::FixedTripleList>)

setFixedTupleList(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*, *(FixedTupleListAdress)arg2*) → None :

C++ signature :
void setFixedTupleList(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::T
{lvalue},std::shared_ptr<espressopp::FixedTupleListAdress>)

setNTrotter(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*, *(object)arg2*) → None :

C++ signature :
void setNTrotter(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::Tabulate
{lvalue},int)

setPotential(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*, *(interaction_TabulatedAngular)arg2*) → None :

C++ signature :
void setPotential(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::Tabulat
{lvalue},std::shared_ptr<espressopp::interaction::TabulatedAngular>)

setSpeedup(*(interaction_FixedTripleListPIadressTabulatedAngular)arg1*, *(object)arg2*) → None :

C++ signature :
void setSpeedup(espressopp::interaction::FixedTripleListPIadressInteractionTemplate<espressopp::interaction::Tabulate
{lvalue},bool)

class espressopp.interaction.TabulatedAngular.TabulatedAngular(**args*, ***kwds*)

The TabulatedAngular potential.

espressopp.interaction.TabulatedDihedral

Calculates energies and forces for a dihedral tabulated potential. In the tabulated potential file, angles should be in radians, and the file should cover the range -pi radians to +pi radians (-180 to +180 degrees).

Note that this class has only been tested for symmetric tabulated potentials.

espressopp.interaction.TabulatedDihedral(*itype*, *filename*)

Parameters
itype – The interpolation type: 1 - linear, 2 - akima spline, 3 - cubic spline :param filename: The tabulated potential filename. :type itype: int :type filename: str

espressopp.interaction.FixedQuadrupleListTabulatedDihedral(*system*, *fql*, *potential*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- fql (espressopp.FixedQuadrupleList) – The FixedQuadrupleList.
- potential (espressopp.interaction.Potential) – The potential.

espressopp.interaction.FixedQuadrupleListTabulatedDihedral.setPotential(*potential*)

Parameters
potential (espressopp.interaction.Potential) – The potential object.

espressopp.interaction.FixedQuadrupleListTypesTabulatedDihedral(*system*, *fql*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- ftl (espressopp.FixedQuadrupleList) – The FixedQuadrupleList list.

espressopp.interaction.FixedQuadrupleListTypesTabulatedDihedral(*system*, *ftl*)

Parameters

- system (espressopp.System) – The Espresso++ system object.
- ftl (espressopp.FixedQuadrupleList) – The FixedQuadruple list.

espressopp.interaction.FixedQuadrupleListTypesTabulatedDihedral.setPotential(*type1*, *type2*, *type3*, *type4*, *potential*)

Defines dihedral potential for interaction between particles of types type1-type2-type3-type4.

Parameters

- type1 (int) – Type of particle 1.
- type2 (int) – Type of particle 2.
- type3 (int) – Type of particle 3.
- type4 (int) – Type of particle 4.
- potential (espressopp.interaction.DihedralPotential) – The potential to set up.

class espressopp.interaction.TabulatedDihedral.TabulatedDihedral(**args*, ***kwds*)

The TabulatedDihedral potential.

3.7.9 Wall

espressopp.interaction.LennardJones93Wall

This class defines a Lennard-Jones 9-3 SingleParticlePotential in the direction x.

$$V(r) = \epsilon \left(\left(\frac{\sigma}{r}\right)^9 - \left(\frac{\sigma}{r}\right)^3 \right)$$

where r is the distance from the lower or upper wall in the x direction. $V(r) = 0$ after a distance *sigmaCutoff*.

The parameters have to be defined for every species present in the system with *setParams* and can be retrieved with *getParams*.

Example:

```
>>> LJ93 = espressopp.interaction.LennardJones93Wall()
>>> LJ93.setParams(0, 6., 1., wall_cutoff)
>>> SPLJ93 = espressopp.interaction.SingleParticleLennardJones93Wall(system, LJ93)
>>> system.addInteraction(SPLJ93)
```

espressopp.interaction.LennardJones93Wall()

espressopp.interaction.LennardJones93Wall.getParams(*type_var*)

Parameters

type_var –

Return type

espressopp.interaction.LennardJones93Wall.setParams(*type_var*, *epsilon*, *sigma*, *sigmaCutoff*, *r0*)

Parameters

- type_var –
- epsilon –
- sigma –
- sigmaCutoff –
- r0 –

espressopp.interaction.SingleParticleLennardJones93Wall(*system*, *potential*)

Parameters

- system –
- potential –

espressopp.interaction.SingleParticleLennardJones93Wall.setPotential(*potential*)

Parameters
potential –

class espressopp.interaction.LennardJones93Wall.LennardJones93Wall(**args*, ***kwds*)

The LennardJones93Wall potential.

3.7.10 Other

espressopp.interaction.Interaction

This is an abstract class, only needed to be inherited from.

espressopp.interaction.Interaction.bondType()

Return type
int

espressopp.interaction.Interaction.computeEnergy()

Return type
real

espressopp.interaction.Interaction.computeEnergyAA(*atomtype*)

Parameters
type1 (int :rtype: real) – Type of particles with respect to which the atomistic energy is calculated.

espressopp.interaction.Interaction.computeEnergyDeriv()

Return type
real

espressopp.interaction.Interaction.computeEnergyCG(*atomtype*)

Parameters
type1 (int :rtype: real) – Type of particles with respect to which the coarse-grained energy is calculated.

espressopp.interaction.Interaction.computeVirial()

Return type
real

espressopp.interaction.Potential

This is an abstract class, only needed to be inherited from.

espressopp.interaction.Potential.computeEnergy(**args*)

Parameters
*args –

Return type

espressopp.interaction.Potential.computeForce(**args*)

Parameters
*args –

Return type

espressopp.interaction.PotentialVSpherePair

This is an abstract class, only needed to be inherited from.

espressopp.interaction.PotentialVSpherePair.computeEnergy(**args*)

Parameters
*args –

Return type

espressopp.interaction.PotentialVSpherePair.computeForce(**args*)

Parameters
*args –

Return type

espressopp.interaction.Quartic

This class provides methods to compute forces and energies of the Quartic potential.

$$U = \frac{K}{4}\left(d^2 - r_0^2\right)^2$$

espressopp.interaction.Quartic(*K*, *r0*, *cutoff*, *shift*)

Parameters

- K (real) – (default: 1.0)
- r0 (real) – (default: 0.0)
- cutoff – (default: infinity)
- shift (real) – (default: 0.0)

espressopp.interaction.FixedPairListQuartic(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListQuartic.getFixedPairList()

Return type
A Python list of lists.

espressopp.interaction.FixedPairListQuartic.setFixedPairList(*fixedpairlist*)

Parameters
fixedpairlist –

espressopp.interaction.FixedPairListQuartic.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

class espressopp.interaction.Quartic.Quartic(**args*, ***kwds*)

The Quartic potential.

espressopp.interaction.ReactionFieldGeneralized

This class provides methods to compute forces and energies of the generalized reaction field.

$$U = PQ \left(\frac{1}{d} - \frac{\left(1 + \frac{(\varepsilon_1 - 4\varepsilon_2)(1+\kappa r_c) - 2\varepsilon_2 \kappa {r_c}^2}{(\varepsilon_1 + 2\varepsilon_2)(1+\kappa r_c) + \varepsilon_2 \kappa {r_c}^2}\right)}{r_c^3 2} \cdot d^2 - \frac{3\varepsilon_2}{r_c(2\varepsilon_2 + 1)} \right)$$

where *P* is a prefactor, *Q* is the product of the charges of the two particles, *d* is their distance from each other, and r_c the cutoff-radius.

espressopp.interaction.ReactionFieldGeneralized(*prefactor*, *kappa*, *epsilon1*, *epsilon2*, *cutoff*, *shift*)

Defines a ReactionFieldGeneralized potential.

Parameters

- prefactor (real) – (default: 1.0) prefactor
- kappa (real) – (default: 0.0) kappa parameter
- epsilon1 (real) – (default: 1.0) epsilon1 parameter
- epsilon2 (real) – (default: 80.0) epsilon2 parameter
- cutoff (real or "infinity") – (default: infinity) cutoff
- shift (real or "auto") – (default: “auto”) shift

espressopp.interaction.VerletListReactionFieldGeneralized(*vl*)

Defines a verletlist-based interaction using a ReactionFieldGeneralized potential.

Parameters
vl (std::shared_ptr<VerletList>) – Verletlist object

espressopp.interaction.VerletListReactionFieldGeneralized.getPotential(*type1*, *type2*)

Gets the ReactionFieldGeneralized interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<ReactionFieldGeneralized>

espressopp.interaction.VerletListReactionFieldGeneralized.setPotential(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressReactionFieldGeneralized(*vl*, *fixedtupleList*)

Defines a verletlist-based AdResS interaction using a ReactionFieldGeneralized potential for the AT and a tabulated potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialAT(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListAdressATReactionFieldGeneralized(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based AdResS interaction using a ReactionFieldGeneralized potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListAdressATReactionFieldGeneralized.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListAdressATReactionFieldGeneralized interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListAdressATReactionFieldGeneralized.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListAdressATReactionFieldGeneralized interaction for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<ReactionFieldGeneralized>

espressopp.interaction.VerletListHadressReactionFieldGeneralized(*vl*, *fixedtupleList*)

Defines a verletlist-based H-AdResS interaction using a ReactionFieldGeneralized potential for the AT and a tabulated potential for the CG interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressReactionFieldGeneralized.setPotentialAT(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized interaction potential for interacting AT particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListHadressReactionFieldGeneralized.setPotentialCG(*type1*, *type2*, *potential*)

Sets the Tabulated interaction potential for interacting CG particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

- potential (std::shared_ptr<Tabulated>) – tabulated interaction potential object

espressopp.interaction.VerletListHadressATReactionFieldGeneralized(*vl*, *fixedtupleList*)

Defines only the AT part of a verletlist-based H-AdResS interaction using a ReactionFieldGeneralized potential for the AT interaction.

Parameters

- vl (std::shared_ptr<VerletListAdress>) – Verletlist AdResS object
- fixedtupleList (std::shared_ptr<FixedTupleListAdress>) – FixedTupleList object

espressopp.interaction.VerletListHadressATReactionFieldGeneralized.setPotential(*type1*, *type2*, *potential*)

Sets the AT potential in VerletListHadressATReactionFieldGeneralized interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

espressopp.interaction.VerletListHadressATReactionFieldGeneralized.getPotential(*type1*, *type2*)

Gets the AT potential in VerletListHadressATReactionFieldGeneralized interaction for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2

Return type
std::shared_ptr<ReactionFieldGeneralized>

espressopp.interaction.CellListReactionFieldGeneralized(*stor*)

Defines a CellList-based interaction using a ReactionFieldGeneralized potential.

Parameters
stor (std::shared_ptr <storage::Storage>) – storage object

espressopp.interaction.CellListReactionFieldGeneralized.setPotential(*type1*, *type2*, *potential*)

Sets the ReactionFieldGeneralized interaction potential for interacting particles of type1 and type2.

Parameters

- type1 (int) – particle type 1
- type2 (int) – particle type 2
- potential (std::shared_ptr<ReactionFieldGeneralized>) – ReactionFieldGeneralized potential object

class espressopp.interaction.ReactionFieldGeneralized.ReactionFieldGeneralized(**args*, ***kwds*)

The ReactionFieldGeneralized potential.

espressopp.interaction.ReactionFieldGeneralizedTI

This module is for performing simulations (e.g. as part of Thermodynamic Integration) where some interactions are a linear function of a parameter λ.

$$U(\lambda) = (1 - \lambda)U_C^A$$

where U_C^A is the standard Reaction Field interaction. This allows one to perform TI where the charges in TI state A ($\lambda = 0$) are the particle charges contained in the particle property charge and the charges in TI state B ($\lambda = 1$) are zero.

The user specifies a list of particles, pidlist. For all pairs of particles with particletypes interacting via this potential, the RF interaction between two particles i and j is calculated as follows:

if (i not in pidlist) and (j not in pidlist):
U_{RF} (full RF interaction)

if (i in pidlist) and (j in pidlist):

if annihilate==True:
$(1 - \lambda)U_{RF}$ (RF interaction scaled by 1-lambda)

if annihilate==False:
U_{RF} (full RF interaction)

if (i in pidlist) xor (j in pidlist):
$(1 - \lambda)U_{RF}$ (RF interaction scaled by 1-lambda)

The default is annihilation (completely turning off charges of particles in pidlist in state B, so that interactions within pidlist are turned off and also cross-interactions between particles in pidlist and particles in the rest of the system). The alternative is decoupling (only cross-interactions between particles in pidlist and particles in the rest of the system are turned off. Interactions within pidlist are not affected.) If annihilation==False, then decoupling is performed. See: http://www.alchemistry.org/wiki/Decoupling_and_annihilation

Exclusions apply as normal, i.e. interactions are only calculated for pairs of particles not already excluded.

So far only VerletListAdressReactionFieldGeneralizedTI is implemented, however VerletListReactionFieldGeneralizedTI, VerletListHadressReactionFieldGeneralizedTI, etc. can also be easily implemented.

The λ (lambdaTI) parameter used here should not be confused with the λ (lambda_adr) particle property used in AdResS simulations.

See also the Thermodynamic Integration tutorial.

Example python script:

```
>>> #value of lambda
>>> lambdaTI = 0.3
>>> #construct RF potential with parameters prefactor,kappa,epsilon1,epsilon2,cutoff as in standard␣
↪RF interaction
>>> pot = espressopp.interaction.ReactionFieldGeneralizedTI(prefactor=prefactor, kappa=kappa,␣
↪epsilon1=epsilon1, epsilon2=epsilon2, cutoff=rc, lambdaTI=lambdaTI, annihilate=False)
>>> #add list of indices of particles whose charge is 0 in TI state B
>>> pidlist = [1,2,3,4]
>>> pot.addPids(pidlist)
>>> #create interaction using VerletListAdress object and FixedTupleListAdress object
>>> qq_adres_interaction=espressopp.interaction.VerletListAdressReactionFieldGeneralizedTI(verletlist,
↪ ftpl)
>>> #loop over list of all types for particles interacting with this atomistic potential
>>> for i in types:
```

(continues on next page)

(continued from previous page)

```
>>>   for k in types:
>>>     qq_adres_interaction.setPotentialAT(type1=i, type2=k, potential=pot)
>>> system.addInteraction(qq_adres_interaction)
```

During the MD run, one can then calculate the derivative of the RF energy wrt lambda

```
>>> #calculate dU/dlambda
>>> dUdl = qq_adres_interaction.computeEnergyDeriv()
```

espressopppp.interaction.ReactionFieldGeneralizedTI(*prefactor*, *kappa*, *epsilon1*, *epsilon2*, *cutoff*, *lambdaTI*, *annihilate*)

Parameters

- prefactor (real) – (default: 1.0) RF parameter
- kappa (real) – (default: 0.0) RF parameter
- epsilon1 (real) – (default: 1.0) RF parameter
- epsilon2 (real) – (default: 80.0) RF parameter
- cutoff (real) – (default: infinity) interaction cutoff
- lambdaTI (real) – (default: 0.0) TI lambda parameter
- annihilate (bool) – (default: True) switch between annihilation and decoupling

espressopppp.interaction.ReactionFieldGeneralizedTI.addPids(*pidlist*)

Parameters

pidlist (python list) – list of particle ids of particles whose charge is zero in state B

espressopppp.interaction.VerletListAdressReactionFieldGeneralized(*vl*, *fixedtupleList*)

Parameters

- vl (VerletListAdress object) – Verlet list
- fixedtupleList (FixedTupleListAdress object) – list of tuples describing mapping between CG and AT particles

espressopppp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 (int) – atomtype
- type2 (int) – atomtype
- potential (Potential) – espressopppp potential

espressopppp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 (int) – atomtype
- type2 (int) – atomtype
- potential (Potential) – espressopppp potential

class espressopp.interaction.ReactionFieldGeneralizedTI.ReactionFieldGeneralizedTI(**args*, ***kwds*)

The ReactionFieldGeneralizedTI potential.

espressopp.interaction.SingleParticlePotential

This class is used to define single-particle interactions, typically used for external forces on the system.

The potential may depend on any of the particle properties (type, mass, etc.).

espressopp.interaction.SingleParticlePotential.computeEnergy(*position*, *bc*)

Parameters

- position –
- bc –

Return type

espressopp.interaction.SingleParticlePotential.computeForce(*position*, *bc*)

Parameters

- position –
- bc –

Return type

espressopp.interaction.VSpherePair

This class provides methods to compute forces and energies of the VSpherePair potential.

$$V(r_{ij}, \sigma_{ij}) = \varepsilon \left(\frac{2\pi}{3} \sigma_{ij} \right)^{-\frac{3}{2}} e^{-\frac{3}{2}\frac{r_{ij}^2}{\sigma_{ij}}}, r_{ij} = |\vec{r_i} - \vec{r_j}|, \sigma_{ij} = \sigma_i^2 + \sigma_j^2$$

Reference: Flactuating soft-sphere approach to coars-graining of polymer melts, Soft matter, 2010, 6, 2282

espressopp.interaction.VSpherePair(*epsilon*, *cutoff*, *shift*)

Parameters

- epsilon (real) – (default: 1.0)
- cutoff – (default: infinity)
- shift – (default: “auto”)

espressopp.interaction.VerletListVSpherePair(*vl*)

Parameters
vl –

espressopp.interaction.VerletListVSpherePair.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListVSpherePair.getVerletList()

Return type
A Python list of lists.

espressopp.interaction.VerletListVSpherePair.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

class espressopp.interaction.VSpherePair.VSpherePair(**args*, ***kwds*)

The Lennard-Jones potential.

espressopp.interaction.VSphereSelf

This class provides methods to compute forces and energies of the VSphereSelf potential.

$$U = e_1 \left(\frac{4}{3}\pi\sigma^2\right)^{\frac{3}{2}} + \frac{a_1 {N_b}^3}{\sigma^6} + \frac{a_2}{N_b}\sigma^2$$

Reference: Flactuating soft-sphere approach to coars-graining of polymer melts, Soft matter, 2010, 6, 2282

espressopp.interaction.VSphereSelf(*e1*, *a1*, *a2*, *Nb*, *cutoff*, *shift*)

Parameters

- e1 (real) – (default: 0.0)
- a1 (real) – (default: 1.0)
- a2 (real) – (default: 0.0)
- Nb (int) – (default: 1)
- cutoff – (default: infinity)
- shift (real) – (default: 0.0)

espressopp.interaction.SelfVSphere(*system*, *potential*)

Parameters

- system –
- potential –

espressopp.interaction.SelfVSphere.getPotential()

Return type

espressopp.interaction.SelfVSphere.setPotential(*potential*)

Parameters

potential –

class espressopp.interaction.VSphereSelf.VSphereSelf(**args*, ***kwds*)

The VSphereSelf potential.

espressopp.interaction.Zero

This class provides methods for a zero potential no interactions between particles, mainly used for debugging and testing

espressopp.interaction.Zero()

espressopp.interaction.VerletListZero(*vl*)

Parameters
vl –

espressopp.interaction.VerletListZero.getPotential(*type1*, *type2*)

Parameters

- type1 –
- type2 –

Return type

espressopp.interaction.VerletListZero.setFixedTupleList(*ftpl*)

Parameters
ftpl –

espressopp.interaction.VerletListZero.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressZero(*vl*)

Parameters
vl –

espressopp.interaction.VerletListAdressZero.setFixedTupleList(*ftpl*)

Parameters
ftpl –

espressopp.interaction.VerletListAdressZero.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListAdressZero.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressZero(*vl*, *fixedtupleList*)

Parameters

- vl –
- fixedtupleList –

espressopp.interaction.VerletListHadressZero.setFixedTupleList(*ftpl*)

Parameters

ftpl –

espressopp.interaction.VerletListHadressZero.setPotentialAT(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.VerletListHadressZero.setPotentialCG(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.CellListZero(*stor*)

Parameters

stor –

espressopp.interaction.CellListZero.setPotential(*type1*, *type2*, *potential*)

Parameters

- type1 –
- type2 –
- potential –

espressopp.interaction.FixedPairListZero(*system*, *vl*, *potential*)

Parameters

- system –
- vl –
- potential –

espressopp.interaction.FixedPairListZero.setPotential(*potential*)

Parameters

potential –

class espressopp.interaction.Zero.Zero(**args*, ***kwds*)

The Zero potential.

espressopp.interaction.SmoothSquareWell

This is an implementation of the smoothed square-well potential from Leitold and Dellago JCP 141, 134901 (2014) :

$$V(r) = \frac{\varepsilon}{2}\left\{\exp\left[\frac{-(r-\sigma)}{a}\right] + \tanh\left[\frac{r-\lambda\sigma}{a}\right] - 1\right\},$$

of which a dictates the steepness of the slope of the square well, and $\lambda\sigma$ determines the width of the step, and σ is the bond length of the polymer.

To reproduce the potential in the prior reference, use the code below.

```
pot = espressopp.interaction.SmoothSquareWell(epsilon=1.0,sigma=1.0,cutoff=2.5)
pot.a = 0.002
pot.Lambda = 1.05
```

The SmoothSquareWell potential supports VerletListInteraction, FixedPairListInteraction and FixedPairListTypesInteraction.

class espressopp.interaction.SmoothSquareWell.SmoothSquareWell(**args*, ***kwds*)

The SmoothSquareWell potential.

3.8 io

3.8.1 espressopp.io.DumpGRO

- *dump()* write configuration to trajectory GRO file. By default filename is "out.gro", coordinates are folded.

  Properties

- *filename* Name of trajectory file. By default trajectory file name is "out.gro"
- *unfolded* False if coordinates are folded, True if unfolded. By default - False
- *append* True if new trajectory data is appended to existing trajectory file. By default - True
- *length_factor* If length dimension in current system is nm, and unit is 0.23 nm, for example, then length_factor should be 0.23
- *length_unit* It is length unit. Can be LJ, nm or A. By default - LJ

usage:

writing down trajectory

```
>>> dump_conf_gro = espressopp.io.DumpGRO(system, integrator, filename='trajectory.gro')
>>> for i in range (200):
>>>   integrator.run(10)
>>>   dump_conf_gro.dump()
```

writing down trajectory using ExtAnalyze extension

```
>>> dump_conf_gro = espressopp.io.DumpGRO(system, integrator, filename='trajectory.gro')
>>> ext_analyze = espressopp.integrator.ExtAnalyze(dump_conf_gro, 10)
>>> integrator.addExtension(ext_analyze)
>>> integrator.run(2000)
```

Both exapmles will give the same result: 200 configurations in trajectory .gro file.

setting up length scale

For example, the Lennard-Jones model for liquid argon with $\sigma = 0.34[nm]$

```
>>> dump_conf_gro = espressopp.io.DumpGRO(system, integrator, filename='trj.gro', unfolded=False,
↪ length_factor=0.34, length_unit='nm', append=True)
```

will produce trj.gro with in nanometers

espressopp.io.DumpGRO(*system*, *integrator*, *filename*, *unfolded*, *length_factor*, *length_unit*, *append*)

Parameters

- system –
- integrator –
- filename – (default: 'out.gro')
- unfolded – (default: False)
- length_factor (real) – (default: 1.0)
- length_unit – (default: 'LJ')
- append – (default: True)

espressopp.io.DumpGRO.dump()

Return type

3.8.2 espressopp.io.DumpGROAdress

dumps coordinates of atomistic particles instead of coarse-grained particles in Adress simulation

- *dump()* write configuration to trajectory GRO file. By default filename is "out.gro", coordinates are folded.

  Properties

- *filename* Name of trajectory file. By default trajectory file name is "out.gro"
- *unfolded* False if coordinates are folded, True if unfolded. By default - False
- *append* True if new trajectory data is appended to existing trajectory file. By default - True
- *length_factor* If length dimension in current system is nm, and unit is 0.23 nm, for example, then length_factor should be 0.23
- *length_unit* It is length unit. Can be LJ, nm or A. By default - LJ
- ftpl fixedtuplelist for the adres system

usage:

```
>>> ftpl = espressopp.FixedTupleListAdress(system.storage)
>>> ftpl.addTuples(tuples)
>>> system.storage.setFixedTuplesAdress(ftpl)
>>> system.storage.decompose()
```

writing down trajectory

```
>>> dump_conf_gro = espressopp.io.DumpGROAdress(system, ftpl, integrator, filename='trajectory.gro
↪')
>>> for i in range (200):
>>>   integrator.run(10)
>>>   dump_conf_gro.dump()
```

writing down trajectory using ExtAnalyze extension

```
>>> dump_conf_gro = espressopp.io.DumpGROAdress(system, ftpl, integrator, filename='trajectory.gro
↪')
>>> ext_analyze = espressopp.integrator.ExtAnalyze(dump_conf_gro, 10)
>>> integrator.addExtension(ext_analyze)
>>> integrator.run(2000)
```

Both exapmles will give the same result: 200 configurations in trajectory .gro file.

setting up length scale

For example, the Lennard-Jones model for liquid argon with $\sigma = 0.34[nm]$

```
>>> dump_conf_gro = espressopp.io.DumpGROAdress(system, ftpl, integrator, filename='trj.gro',␣
↪unfolded=False, length_factor=0.34, length_unit='nm', append=True)
```

will produce trj.gro with in nanometers

espressopp.io.DumpGROAdress(*system*, *fixedtuplelist*, *integrator*, *filename*, *unfolded*, *length_factor*, *length_unit*, *append*)

Parameters

- system –
- fixedtuplelist –
- integrator –
- filename – (default: 'out.gro')
- unfolded – (default: False)
- length_factor (real) – (default: 1.0)
- length_unit – (default: 'LJ')
- append – (default: True)

espressopp.io.DumpGROAdress.dump()

Return type

3.8.3 espressopp.io.DumpXYZ

- *dump()*

  write configuration to trajectory XYZ file. By default filename is out.xyz, coordinates are folded. DumpXYZ works also for Multiple communicators.

  Properties

- *filename* Name of trajectory file. By default trajectory file name is out.xyz
- *unfolded* False if coordinates are folded, True if unfolded. By default - False
- *append* True if new trajectory data is appended to existing trajectory file. By default - True
- *length_factor* If length dimension in current system is nm, and unit is 0.23 nm, for example, then length_factor should be 0.23 Default: 1.0
- *length_unit* It is length unit. Can be LJ, nm or A. By default - LJ
- ***store_pids***
  True if you want to store pids as fastwritexyz does. False otherwise (standard XYZ) Default: False
- ***store_velocities***
  True if you want to store velocities. False otherwise (XYZ doesn't require it) Default: False

usage:

writing down trajectory

```
>>> dump_conf_xyz = espressopp.io.DumpXYZ(system, integrator, filename='trajectory.xyz')
>>> for i in range (200):
>>>   integrator.run(10)
>>>   dump_conf_xyz.dump()
```

writing down trajectory using ExtAnalyze extension

```
>>> dump_conf_xyz = espressopp.io.DumpXYZ(system, integrator, filename='trajectory.xyz')
>>> ext_analyze = espressopp.integrator.ExtAnalyze(dump_conf_xyz, 10)
>>> integrator.addExtension(ext_analyze)
>>> integrator.run(2000)
```

Both examples will give the same result: 200 configurations in trajectory .xyz file.

setting up length scale

For example, the Lennard-Jones model for liquid argon with $\sigma = 0.34[nm]$

```
>>> dump_conf_xyz = espressopp.io.DumpXYZ(system, integrator, filename='trj.xyz', \
>>>                                       unfolded=False, length_factor=0.34, \
>>>                                       length_unit='nm', store_pids=True, \
>>>                                       store_velocities = True, append=True)
```

will produce trj.xyz with in nanometers

espressopp.io.DumpXYZ(*system*, *integrator*, *filename=out.xyz*, *unfolded=False*, *length_factor=1.0*, *length_unit='LJ'*, *store_pids=False*, *store_velocities=False*, *append=True*)

Parameters

- system –
- integrator –

- filename –
- unfolded (bool) –
- length_factor (real) –
- length_unit –
- store_pids (bool) –
- store_velocities (bool) –
- append (bool) –

espressopp.io.DumpXYZ.dump()

Return type

3.8.4 DumpH5MD - IO object

This module provides a writer for H5MD file format.

espressopp.io.DumpH5MD(*system*, *integrator*, *filename*, *group_name*, **args*)

Parameters

- system (espressopp.System) – The system object.
- integrator (espressopp.integrator.MDIntegrator) – System integrator.
- filename (str) – The file name.
- group_name (str) – The name of particle group.
- is_adress (bool) – If positive then store position of AdResS particles
- author (str) – The name of author of this file
- email (str) – The e-mail address to the author.
- chunk_size (int) –
- static_box (bool) – box size written as time-independent variable
- is_single_prec (bool) – Store float values with single precision (default: False)
- store_position – Saves postions of particles
- store_species – Saves types of particles.
- store_state – Saves states of particles.
- store_velocity – Saves velocities of particles.
- store_force – Saves forces of particles
- store_charge – Saves charges of particles
- store_lambda – Saves lambdas (AdResS) of particles
- store_res_id – Saves residues id of particles.
- store_mass – Saves masses of particles

Return type
The DumpH5MD writer.

Example

```
>>> traj_file = espressopp.io.DumpH5MD(
      system, integrator, output_file,
      group_name='atoms',
      static_box=False,
      author='xxx',
      email='xxx@xxx',
      store_species=True,
      store_velocity=True,
      store_state=True,
      store_lambda=True)
```

```
>>> for s in range(steps):
      integrator.run(int_steps)
      traj_file.dump(s*int_steps, s*int_steps*integrator.dt)
```

Important note. Within the current approach, this extension is not compatible with ExtAnalyze module. Therefore, this code does not work:

```
>>> ext_analyze = espressopp.integrator.ExtAnalyze(traj_file, 10)
>>> integrator.addExtension(ext_analyze)
>>> integrator.run(2000)
```

Sorting file

The content of the */particles/{}/* is not sorted with respect to the particle id. This is because of the way how the data are stored by multiple cores simultaneously.

If the flag *do_sort* is True then during the close method, the data will be sorted.

3.8.5 Parallel H5MD Output

H5MD is a format specification on top of the HDF5 file format.

Dump Simulation

Dumping a simulation with espressopp.io.DumpH5MDParallel

```
dump_h5md_parallel = espressopp.io.DumpH5MDParallel(system, 'dump.h5')
dump_h5md_parallel.dump()
```

ATTENTION

Only V=const (NVT, NVE) simulation are dumped correctly since we do not save box dimensions at every simulation step.

Configuration

- dump* controls which properties are saved to file.
- *Dataset controls the dataset name within the file of the corresponding property.

Restore Simulation

Restoring a simulation with espressopp.io.RestoreH5MDParallel

```
system.storage.removeAllParticles()
restore_h5md_parallel = espressopp.io.RestoreH5MDParallel(system, 'dump.h5')
restore_h5md_parallel.restore()
system.storage.decompose()
```

ATTENTION

- No checks for duplicates are performed nor is the particle storage cleared before inserting new particles. You might want to remove all particles from the simulation before calling restore.
- Particles are inserted equally among the processes without obeying subdomains. You have to call decompose to ensure particles are located on the correct process after restoration.

Configuration

- restore* controls which properties are loaded from file.
- *Dataset controls the dataset name within the file of the corresponding property.

3.9 espressopp

3.9.1 espressopp.Exceptions

espressopp.Error(*msg*)

Parameters
msg –

espressopp.ParticleDoesNotExistHere(*msg*)

Parameters
msg –

espressopp.UnknownParticleProperty(*msg*)

Parameters
msg –

espressopp.MissingFixedPairList(*msg*)

Parameters
msg –

exception espressopp.Exceptions.Error(*msg*)

exception espressopp.Exceptions.MissingFixedPairList(*msg*)

exception espressopp.Exceptions.ParticleDoesNotExistHere(*msg*)

exception espressopp.Exceptions.UnknownParticleProperty(*msg*)

3.9.2 espressopp.FixedLocalTupleList

This class can contain many tuple which store a arbitrary positive number, which should be more than 2, of local (real + ghost) particle id.

For using this class, there is 1 conditions:

Particles in one tuple must be in a same or neighbor cell list.

espressopp.FixedLocalTupleList(*storage*)

Parameters
storage –

espressopp.FixedLocalTupleList.addTuple(*tuple*)

Parameters
tuple (python::list) –

espressopp.FixedLocalTupleList.getTuples()

Return type

espressopp.FixedLocalTupleList.size()

Return type

3.9.3 espressopp.FixedPairDistList

espressopp.FixedPairDistList(*storage*)

Parameters
storage –

espressopp.FixedPairDistList.add(*pid1*, *pid2*)

Parameters

- pid1 –
- pid2 –

Return type

espressopp.FixedPairDistList.addPairs(*bondlist*)

Parameters
bondlist –

Return type

espressopp.FixedPairDistList.getDist(*pid1*, *pid2*)

Parameters

- pid1 –
- pid2 –

Return type

espressopp.FixedPairDistList.getPairs()

Return type

espressopp.FixedPairDistList.getPairsDist()

Return type

espressopp.FixedPairDistList.size()

Return type

3.9.4 espressopp.FixedPairList

espressopp.FixedPairList(*storage*)

Parameters
storage –

espressopp.FixedPairList.add(*pid1*, *pid2*)

Parameters

- pid1 –
- pid2 –

Return type

espressopp.FixedPairList.addBonds(*bondlist*)

Parameters
bondlist –

Return type

espressopp.FixedPairList.getBonds()

Return type

espressopp.FixedPairList.remove()
'remove the FixedPairList and disconnect'

espressopp.FixedPairList.getLongtimeMaxBond()

Return type

espressopp.FixedPairList.resetLongtimeMaxBond()

Return type

espressopp.FixedPairList.size()

Return type

espressopp.FixedPairList.totalSize()

Return type

3.9.5 espressopp.FixedPairListAdress

The FixedPairListAdress is the Fixed Pair List to be used for AdResS or H-AdResS simulations. When creating the FixedPairListAdress one has to provide the storage and the tuples. Afterwards the bonds can be added. In the example "bonds" is a python list of the form ((pid1, pid2), (pid3, pid4), ...) where each inner pair defines a bond between the particles with the given particle ids.

Example - creating the FixedPairListAdress and adding bonds:

```
>>> ftpl = espressopp.FixedTupleList(system.storage)
>>> fpl = espressopp.FixedPairListAdress(system.storage, ftpl)
>>> fpl.addBonds(bonds)
```

espressopp.FixedPairListAdress(*storage*, *fixedtupleList*)

Parameters

- storage –
- fixedtupleList –

espressopp.FixedPairListAdress.add(*pid1*, *pid2*)

Parameters

- pid1 –
- pid2 –

Return type

espressopp.FixedPairListAdress.addBonds(*bondlist*)

Parameters
bondlist –

Return type

espressopp.FixedPairListAdress.remove()
remove the FixedPairListAdress and disconnect

espressopp.FixedPairListAdress.getBonds()

Return type

3.9.6 espressopp.FixedQuadrupleAngleList

espressopp.FixedQuadrupleAngleList(*storage*)

Parameters
storage –

espressopp.FixedQuadrupleAngleList.add(*pid1*, *pid2*, *pid3*, *pid4*)

Parameters

- pid1 –
- pid2 –
- pid3 –
- pid4 –

Return type

espressopp.FixedQuadrupleAngleList.addQuadruples(*quadruplelist*)

Parameters
quadruplelist –

Return type

espressopp.FixedQuadrupleAngleList.getAngle(*pid1*, *pid2*, *pid3*, *pid4*)

Parameters

- pid1 –
- pid2 –
- pid3 –
- pid4 –

Return type

espressopp.FixedQuadrupleAngleList.getQuadruples()

Return type

espressopp.FixedQuadrupleAngleList.getQuadruplesAngles()

Return type

espressopp.FixedQuadrupleAngleList.size()

Return type

3.9.7 espressopp.FixedQuadrupleList

espressopp.FixedQuadrupleList(*storage*)

Parameters
storage –

espressopp.FixedQuadrupleList.add(*pid1*, *pid2*, *pid3*, *pid4*)

Parameters

- pid1 –
- pid2 –
- pid3 –
- pid4 –

Return type

espressopp.FixedQuadrupleList.addQuadruples(*quadruplelist*)

Parameters
quadruplelist –

Return type

espressopp.FixedQuadrupleList.remove()
remove the FixedPairList and disconnect

espressopp.FixedQuadrupleList.getQuadruples()

Return type

espressopp.FixedQuadrupleList.size()

Return type

3.9.8 espressopp.FixedQuadrupleListAdress

espressopp.FixedQuadrupleListAdress(*storage*, *fixedtupleList*)

Parameters

- storage –
- fixedtupleList –

espressopp.FixedQuadrupleListAdress.add(*pid1*, *pid2*, *pid3*, *pid4*)

Parameters

- pid1 –
- pid2 –
- pid3 –
- pid4 –

Return type

espressopp.FixedQuadrupleListAdress.addQuadruples(*quadruplelist*)

Parameters
quadruplelist –

Return type

espressopp.FixedQuadrupleListAdress.getQuadruples()

Return type

espressopp.FixedQuadrupleListAdress.size()

Return type

3.9.9 espressopp.FixedSingleList

espressopp.FixedSingleList(*storage*)

Parameters
storage –

espressopp.FixedSingleList.add(*pid1*)

Parameters
pid1 –

Return type

espressopp.FixedSingleList.addSingles(*singlelist*)

Parameters
singlelist –

Return type

espressopp.FixedSingleList.getSingles()

Return type

espressopp.FixedSingleList.size()

Return type

3.9.10 espressopp.FixedTripleAngleList

espressopp.FixedTripleAngleList(*storage*)

Parameters
storage –

espressopp.FixedTripleAngleList.add(*pid1*, *pid2*, *pid3*)

Parameters

- pid1 –
- pid2 –
- pid3 –

Return type

espressopp.FixedTripleAngleList.addTriples(*triplelist*)

Parameters
triplelist –

Return type

espressopp.FixedTripleAngleList.getAngle(*pid1*, *pid2*, *pid3*)

Parameters

- pid1 –
- pid2 –
- pid3 –

Return type

espressopp.FixedTripleAngleList.getTriples()

Return type

espressopp.FixedTripleAngleList.getTriplesAngles()

Return type

espressopp.FixedTripleAngleList.size()

Return type

3.9.11 espressopp.FixedTripleList

espressopp.FixedTripleList(*storage*)

Parameters
storage –

espressopp.FixedTripleList.add(*pid1*, *pid2*, *pid3*)

Parameters

- pid1 –
- pid2 –
- pid3 –

Return type

espressopp.FixedTripleList.addTriples(*triplelist*)

Parameters
triplelist –

Return type

espressopp.FixedTripleList.getTriples()

Return type

espressopp.FixedTripleList.size()

Return type

espressopp.FixedTripleList.remove()
remove the FixedPairList and disconnect

3.9.12 espressopp.FixedTripleListAdress

espressopp.FixedTripleListAdress(*storage*, *fixedtupleList*)

Parameters

- storage –
- fixedtupleList –

espressopp.FixedTripleListAdress.add(*pid1*, *pid2*)

Parameters

- pid1 –
- pid2 –

Return type

espressopp.FixedTripleListAdress.remove()
remove the FixedTripleListAdress and disconnect

espressopp.FixedTripleListAdress.addTriples(*triplelist*)

Parameters
triplelist –

Return type

3.9.13 espressopp.FixedTupleList

espressopp.FixedTupleList(*storage*)

Parameters
storage –

espressopp.FixedTupleList.size()

Return type

3.9.14 espressopp.FixedTupleListAdress

The FixedTupleListAdress is important for AdResS and H-AdResS simulations. It is the connection between the atomistic and coarse-grained particles. It defines which atomistic particles belong to which coarse-grained particle. In the following example "tuples" is a python list of the form ((pid_CG1, pidAT11, pidAT12, pidAT13, ...), (pid_CG2, pidAT21, pidAT22, pidAT23, ...), ...). Each inner list (pid_CG1, pidAT11, pidAT12, pidAT13, ...) defines a tuple. The first number is the particle id of the coarse-grained particle while the following numbers are the particle ids of the corresponding atomistic particles.

Example - creating the FixedTupleListAdress:

```
>>> ftpl = espressopp.FixedTupleListAdress(system.storage)
>>> ftpl.addTuples(tuples)
>>> system.storage.setFixedTuples(ftpl)
```

espressopp.FixedTupleListAdress(*storage*)

Parameters
storage –

espressopp.FixedTupleListAdress.addTuples(*tuplelist*)

Parameters
tuplelist –

Return type

3.9.15 espressopp.Int3D

espressopp.Int3D(**args*)

Parameters
*args –

espressopp.Int3D.x(*v, [0*)

Parameters

- v –
- [0 –

Return type

espressopp.Int3D.y(*v, [1*)

Parameters

- v –
- [1 –

Return type

espressopp.Int3D.z(*v, [2*)

Parameters

- v –
- [2 –

Return type

espressopp.toInt3DFromVector(**args*)

Parameters
*args –

espressopp.toInt3D(**args*)

Parameters
*args –

espressopp.Int3D.toInt3D(**args*)

Try to convert the arguments to a Int3D, returns the argument, if it is already a Int3D.

espressopp.Int3D.toInt3DFromVector(**args*)

Try to convert the arguments to a Int3D.

This function will only convert to a Int3D if x, y and z are specified.

3.9.16 espressopp.MultiSystem

espressopp.MultiSystem()

espressopp.MultiSystem.beginSystemDefinition()

Return type

espressopp.MultiSystem.runAnalysisNPart()

Return type

espressopp.MultiSystem.runAnalysisPotential()

Return type

espressopp.MultiSystem.runAnalysisTemperature()

Return type

espressopp.MultiSystem.runIntegrator(*niter*)

Parameters
niter –

Return type

espressopp.MultiSystem.setAnalysisNPart(*npart*)

Parameters
npart –

espressopp.MultiSystem.setAnalysisPotential(*potential*)

Parameters
potential –

espressopp.MultiSystem.setAnalysisTemperature(*temperature*)

Parameters
temperature –

espressopp.MultiSystem.setIntegrator(*integrator*)

Parameters
integrator –

class espressopp.MultiSystem.MultiSystem(**args*, ***kwds*)

MultiSystemIntegrator to simulate and analyze several systems in parallel.

class espressopp.MultiSystem.MultiSystemLocal

Local MultiSystem to simulate and analyze several systems in parallel.

3.9.17 espressopp.ParallelTempering

espressopp.ParallelTempering(*NumberOfSystems*, *RNG*)

Parameters

- NumberOfSystems (int) – (default: 4)
- RNG – (default: None)

espressopp.ParallelTempering.endDefiningSystem(*n*)

Parameters
n –

Return type

espressopp.ParallelTempering.exchange()

Return type

espressopp.ParallelTempering.getNumberOfCPUsPerSystem()

Return type

espressopp.ParallelTempering.getNumberOfSystems()

Return type

espressopp.ParallelTempering.run(*nsteps*)

Parameters
nsteps –

Return type

espressopp.ParallelTempering.setAnalysisE(*analysisE*)

Parameters
analysisE –

espressopp.ParallelTempering.setAnalysisNPart(*analysisNPart*)

Parameters
analysisNPart –

espressopp.ParallelTempering.setAnalysisT(*analysisT*)

Parameters
analysisT –

espressopp.ParallelTempering.setIntegrator(*integrator*, *thermostat*)

Parameters

- integrator –
- thermostat –

espressopp.ParallelTempering.startDefiningSystem(*n*)

Parameters
n –

Return type

3.9.18 espressopp.Particle

The Particle class. Particles are used to model atoms, coarse-grained beads, etc. and are the central part of all simulations. They are stored in the storage and their time evolution can be modeled using an integrator. Importantly, particles have various properties which the user and other modules can make use of and which can be accessed. They are listed below.

class espressopp.Particle(*pid*, *storage*)

The particle class.

Parameters

- pid (int) – the particle id
- storage (storage) – the storage object

Real3D espressopp.Particle.pos

position

Real3D espressopp.Particle.v

velocity

Real3D espressopp.Particle.f

force

Real3D espressopp.Particle.modepos

normal mode coordinate (position in normal mode space)

Real3D espressopp.Particle.modemom

normal mode momentum (momentum in normal mode space)

Real3D espressopp.Particle.fm

normal mode force (force in normal mode space)

int espressopp.Particle.type

particle type

int espressopp.Particle.res_id

molecule id (eg. chain id)

int espressopp.Particle.pib

path integral bead number (Trotter number)

real espressopp.Particle.q

charge

real espressopp.Particle.mass

mass

real espressopp.Particle.varmass

variable mass (for path integral-based adaptive resolution simulations)

real espressopp.Particle.radius

particle radius

real espressopp.Particle.vradius

radial velocity

real espressopp.Particle.fradius

radial force

real espressopp.Particle.lambda_adr

particle's resolution parameter (used in adaptive resolution simulations)

real espressopp.Particle.lambda_adrd

particle's gradient of the resolution function in the direction of resolution change (used in adaptive resolution simulations)

bool espressopp.Particle.isGhost

boolean flag to indicate whether particle is ghost particle or not

Int3D espressopp.Particle.imageBox

particle's image box

real espressopp.Particle.extVar

auxiliary variable associated with the particle (used in generalized Langevin friction)

real espressopp.Particle.drift_f

particle's drift force in the direction of resolution change (used in adaptive resolution simulations)

real espressopp.Particle.state

particle state (used in AssociationReaction)

class espressopp.Particle.ParticleLocal(*pid*, *storage*)

The local particle.

Throws an exception: * when the particle does not exists locally

TODO: Should throw an exception: * when a ghost particle is to be written * when data is to be read from a ghost that is not available

3.9.19 espressopp.ParticleAccess

Abstract base class for analysis/measurement/io

espressopp.ParticleAccess.perform_action()

Return type

3.9.20 espressopp.ParticleGroup

espressopp.ParticleGroup(*storage*)

Parameters
storage –

espressopp.ParticleGroup.add(*pid*)

Parameters
pid –

Return type

espressopp.ParticleGroup.has(*pid*)

Parameters
pid –

Return type

espressopp.ParticleGroup.show()

Return type

espressopp.ParticleGroup.size()

Return type

3.9.21 espressopp.pmi

Parallel Method Invocation (PMI) allows users to write serial Python scripts that use functions and classes that are executed in parallel.

PMI is intended to be used in data-parallel environments, where several threads run in parallel and can communicate via MPI.

In PMI mode, a single thread of control (a python script that runs on the *controller*, i.e. the MPI root task) can invoke arbitrary functions on all other threads (the *workers*) in parallel via *call()*, *invoke()* and *reduce()*. When the function on the workers return, the control is returned to the controller.

This model is equivalent to the "Fork-Join execution model" used e.g. in OpenMP.

PMI also allows to create parallel instances of object classes via *create()*, i.e. instances that have a corresponding object instance on all workers. *call()*, *invoke()* and *reduce()* can be used to call arbitrary methods of these instances.

to execute arbitrary code on all workers, *exec_()* can be used, and to import python modules to all workers, use 'import_()'.

Main program

On the workers, the main program of a PMI script usually consists of a single call to the function *startWorkerLoop()*. On the workers, this will start an infinite loop on the workers that waits to receive the next PMI call, while it will immediately return on the controller. On the workers, the loop ends only, when one of the commands *finalizeWorkers()* or *stopWorkerLoop()* is issued on the controller. A typical PMI main program looks like this:

```
>>> # compute 2*factorial(42) in parallel
>>> import pmi
>>>
>>> # start the worker loop
>>> # on the controller, this function returns immediately
>>> pmi.startWorkerLoop()
>>>
>>> # Do the parallel computation
>>> pmi.import_('math')
>>> pmi.reduce('lambda a,b: a+b', 'math.factorial', 42)
>>>
>>> # exit all workers
>>> pmi.finalizeWorkers()
```

Instead of using *finalizeWorkers()* at the end of the script, you can call *registerAtExit()* anywhere else, which will cause *finalizeWorkers()* to be called when the python interpreter exits.

Alternatively, it is possible to use PMI in an SPMD-like fashion, where each call to a PMI command on the controller must be accompanied by a corresponding call on the worker. This can be either a simple call to *receive()* that accepts any PMI command, or a call to the identical PMI command. In that case, the arguments of the call to the PMI command on the workers are ignored. In this way, it is possible to write SPMD scripts that profit from the PMI communication patterns.

```
>>> # compute 2*factorial(42) in parallel
>>> import pmi
>>>
>>> pmi.exec_('import math')
>>> pmi.reduce('lambda a,b: a+b', 'math.factorial', 42)
```

To start the worker loop, the command *startWorkerLoop()* can be issued on the workers. To stop the worker loop, *stopWorkerLoop()* can be issued on the controller, which will end the worker loop without exiting the workers.

Controller commands

These commands can be called in the controller script. When any of these commands is issued on a worker during the worker loop, a *UserError* is raised.

- *call()*, *invoke()*, *reduce()* to call functions and methods in parallel
- *create()* to create parallel object instances
- *exec_()* and *import_()* to execute arbitrary python code in parallel and to import classes and functions into the global namespace of pmi.
- *sync()* to make sure that all deleted PMI objects have been deleted.
- *finalizeWorkers()* to stop and exit all workers
- *registerAtExit()* to make sure that finalizeWorkers() is called when python exits on the controller
- *stopWorkerLoop()* to interrupt the worker loop an all workers and to return control to the single workers

Worker commands

These commands can be called on a worker.

- *startWorkerLoop()* to start the worker loop
- *receive()* to receive a single PMI command
- *call()*, *invoke()*, *reduce()*, *create()* and *exec_()* to receive a single corresponding PMI command. Note that these commands will ignore any arguments when called on a worker.

PMI Proxy metaclass

The *Proxy* metaclass can be used to easily generate front-end classes to distributed PMI classes. . . .

Useful constants and variables

The pmi module defines the following useful constants and variables:

- *isController* is True when used on the controller, False otherwise
- *isWorker* = not isController
- *ID* is the rank of the MPI task
- *CONTROLLER* is the rank of the Controller (normally the MPI root)
- *workerStr* is a string describing the thread ('Worker #' or 'Controller')
- *inWorkerLoop* is True, if PMI currently executes the worker loop on the workers.

class espressopp.pmi.Proxy(*name*, *bases*, *ns*)

A metaclass to be used to create frontend serial objects.

exception espressopp.pmi.UserError(*msg*)

Raised when PMI has encountered a user error.

espressopp.pmi.call(**args*, ***kwds*)

Call a function on all workers, returning only the return value on the controller.

function denotes the function that is to be called, args and kwds are the arguments to the function. If kwds contains keys that start with with the prefix '__pmictr_', they are stripped of the prefix and are passed only to the controller. If the function should return any results, it will be locally returned. Only functions that are known to PMI can be used, that is functions that have been imported to pmi via *exec_()* or *import_()*.

Example:

```
>>> pmi.exec_('import hello')
>>> hw = pmi.create('hello.HelloWorld')
>>> pmi.call(hw.hello)
>>> # equivalent:
>>> pmi.call('hello.HelloWorld', hw)
```

Note, that you can use only functions that are know to PMI when *call()* is called, i.e. functions in modules that have been imported via *exec_()*.

espressopp.pmi.create(*cls=None*, **args*, ***kwds*)

Controller command that creates an object on all workers.

cls describes the (new-style) class that should be instantiated. args are the arguments to the constructor of the class. Only classes that are known to PMI can be used, that is, classes that have been imported to pmi via *exec_()* or *import_()*.

Example:

```
>>> pmi.exec_('import hello')
>>> hw = pmi.create('hello.HelloWorld')
>>> print(hw)
MPI process #0: Hello World!
MPI process #1: Hello World!
...
```

Alternative: Note that in this case the class has to be imported to the calling module *and* via PMI.

```
>>> import hello
>>> pmi.exec_('import hello')
>>> hw = pmi.create(hello.HelloWorld)
>>> print(hw)
MPI process #0: Hello World!
MPI process #1: Hello World!
...
```

espressopp.pmi.exec_(**args*)

Controller command that executes arbitrary python code on all workers.

exec_() allows to execute arbitrary Python code on all workers. It can be used to define classes and functions on all workers. Modules should not be imported via exec_(), instead import_() should be used.

Each element of args should be string that is executed on all workers.

Example:

```
>>> pmi.exec_('import hello')
>>> hw = pmi.create('hello.HelloWorld')
```

espressopp.pmi.finalizeWorkers()

Controller command that stops and exits all workers.

espressopp.pmi.import_(**args*)

Controller command that imports python modules on all workers.

Each element of args should be a module name that is imported to all workers.

Example:

```
>>> pmi.import_('hello')
>>> hw = pmi.create('hello.HelloWorld')
```

espressopp.pmi.invoke(**args*, ***kwds*)

Invoke a function on all workers, gathering the return values into a list.

function denotes the function that is to be called, args and kwds are the arguments to the function. If kwds contains keys that start with with the prefix '__pmictr_', they are stripped of the prefix and are passed only to the controller.

On the controller, invoke() returns the results of the different workers as a list. On the workers, invoke returns None. Only functions that are known to PMI can be used, that is functions that have been imported to pmi via *exec_()* or *import_()*.

Example:

```
>>> pmi.exec_('import hello')
>>> hw = pmi.create('hello.HelloWorld')
>>> messages = pmi.invoke(hw.hello())
>>> # alternative:
>>> messages = pmi.invoke('hello.HelloWorld.hello', hw)
```

espressopp.pmi.receive(*expected=None*)

Worker command that receives and handles the next PMI command.

This function waits to receive and handle a single PMI command. If expected is not None and the received command does not equal expected, raise a *UserError*.

espressopp.pmi.reduce(**args*, ***kwds*)

Invoke a function on all workers, reducing the return values to a single value.

reduceOp is the (associative) operator that is used to process the return values, function denotes the function that is to be called, args and kwds are the arguments to the function. If kwds contains keys that start with with the prefix '__pmictr_', they are stripped of the prefix and are passed only to the controller.

reduce() reduces the results of the different workers into a single value via the operation reduceOp. reduceOp is assumed to be associative. Both reduceOp and function have to be known to PMI, that is they must have been imported to pmi via *exec_()* or *import_()*.

Example:

```
>>> pmi.exec_('import hello')
>>> pmi.exec_('joinstr=lambda a,b: "\n".join(a,b)')
```

(continues on next page)

(continued from previous page)

```
>>> hw = pmi.create('hello.HelloWorld')
>>> print(pmi.reduce('joinstr', hw.hello()))
>>> # equivalent:
>>> print(
...   pmi.reduce('lambda a,b: "\n".join(a,b)',
...              'hello.HelloWorld.hello', hw)
...              )
```

espressopp.pmi.registerAtExit()

Controller command that registers the function *finalizeWorkers()* via atexit.

espressopp.pmi.startWorkerLoop()

Worker command that starts the main worker loop.

This function starts a loop that expects to receive PMI commands until *stopWorkerLoop()* or *finalizeWorkers()* is called on the controller.

espressopp.pmi.stopWorkerLoop(*doExit=False*)

Controller command that stops all workers.

If doExit is set, the workers exit afterwards.

espressopp.pmi.sync()

Controller command that deletes the PMI objects on the workers that have already been deleted on the controller.

3.9.22 espressopp.Quaternion

This class provides quaternions with the associate methods. Quaternions can be used as an efficient representation for the orientation and rotation of 3D vector objects in 3D euclidean space. A Quaternion as such has a real part and an imaginary part. For implementation purposes, the representation through one real scalar and one real 3D vector is used here. The vector part is defined using the Real3D class of espressopp.

The format of a quaternion is "(real_part, unreal_part)" with the types "real" and "Real3D", respectively.

While there are other possible applications for quaternions (rotation) in the simulation code, they will be used at the C++-level in order to per- form the integration of the Euler equations of motion regarding the partic- les angular motion, i.e. the rigid body dynamics.

Usage:

The following methods from C++-level are available at the python-level:

- **getReal()**
  return the scalar part of the quaternion
- **setReal(real)**
  sets the scalar part of the quaternion
- **getImag()**
  returns the vector part of the quaternion
- **getImagItem(i)**
  returns element i of vector part of the quaternion
- **setImag(Real3D)**
  sets the vector part of the quaternion
- **setImagItem(i, real)**
  sets element i of vector part of the quaternion

- **sqr()**
  the inner product of the quaternion
- **abs()**
  the absolute value of the quaternion
- **normalize()**
  normalizes the quaternion to unit length
- **transpose()**
  transposes the quaternion (changes sign of unreal_part)

The multiplication operator is overloaded in order to perform quaternion multiplication, see examples below. Furthermore, it is possible to multi- ply a quaternion with a scalar, in order to rescale it.

Examples:

Initialize:

```
>>> espressopp.Quaternion()
Quaternion(0.0, Real3D(0.0, 0.0, 0.0))
```

```
>>> espressopp.Quaternion(0.0, 1.0, 2.0, 3.0)
Quaternion(1.0, Real3D(1.0, 2.0, 3.0))
```

```
>>> vec = espressopp.Real3D(1.0, 2.0, 3.0)
>>> Quaternion(vec)
Quaternion(0.0, Real3D(1.0, 2.0, 3.0))
```

```
>>> espressopp.Quaternion(1.0)
Quaternion(1.0, Real3D(0.0, 0.0, 0.0))
```

Get:

```
>>> q = espressopp.Quaternion(0.0, 1.0, 2.0, 3.0)
>>> q.getReal()
0.0
>>> q.getImag()
Real3D(1.0, 2.0, 3.0)
>>> q.getImagItem(0)
1.0
```

Set:

```
>>> q = espressopp.Quaternion(0.0, 0.0, 0.0, 0.0)
>>> q.setReal(1.0)
>>> vec = espressopp.Real3D(1.0, 2.0, 3.0)
>>> q.setImag(vec)
>>> q
Quaternion(1.0, Real3D(1.0, 2.0, 3.0))
>>> q.setImagItem(0, 0.0)
Quaternion(1.0, Real3D(0.0, 2.0, 3.0))
```

Transpose and normalize:

```
>>> q = Quaternion(0.0, 1.0, 2.0, 3.0)
>>> q.transpose()
Quaternion(0.0, Real3D(-1.0, -2.0, -3.0))
>>> q = Quaternion(0.0, 1.0, 2.0, 3.0)
>>> q.normalize()
Quaternion(0.0, Real3D(0.2672612419124244, 0.5345224838248488, 0.8017837257372732))
```

Inner product and absolute value:

```
>>> q = Quaternion(0.0, 1.0, 2.0, 3.0)
>>> q.sqr()
14.0
>>> q.abs()
3.7416573867739413
```

Quaternion multiplication (compare, e.g., wikipedia):

```
>>> p = Quaternion(0.0, 1.0, 2.0, 3.0)
>>> q = Quaternion(0.0, 1.0, 2.0, 3.0)
Quaternion(-14.0, Real3D(0.0, 0.0, 0.0))
```

espressopp.Quaternion.toQuaternion(**args*)

Try to convert the arguments to a Quaternion, return the argument if it is already a Quaternion.

espressopp.Quaternion.toQuaternionFromVector(**args*)

Try to convert the arguments to a Quaternion.

This function will only convert to a Quaternion if real_part, unreal_part[0], unreal_part[1] and unreal_part[2] are specified.

3.9.23 espressopp.Real3D

espressopp.Real3D(**args*)

Parameters

*args –

espressopp.Real3D.x(*v, [0*)

Parameters

- v –
- [0 –

Return type

espressopp.Real3D.y(*v, [1*)

Parameters

- v –
- [1 –

Return type

espressopp.Real3D.z(*v, [2*)

Parameters

- v –
- [2 –

Return type

espressopp.toReal3DFromVector(**args*)

Parameters
*args –

espressopp.toReal3D(**args*)

Parameters
*args –

espressopp.Real3D.toReal3D(**args*)

Try to convert the arguments to a Real3D, returns the argument, if it is already a Real3D.

espressopp.Real3D.toReal3DFromVector(**args*)

Try to convert the arguments to a Real3D.

This function will only convert to a Real3D if x, y and z are specified.

3.9.24 espressopp.RealND

This is the object which represents N-dimensional vector. It is an extended Real3D, basicly, it hase the same functionallity but in N-dimetions. First of all it is usefull for classes in 'espressopp.analysis'.

Description

...

espressopp.RealND(**args*)

Parameters
*args –

espressopp.toRealNDFromVector(**args*)

Parameters
*args –

espressopp.toRealND(**args*)

Parameters
*args –

espressopp.RealND.toRealND(**args*)

Try to convert the arguments to a RealND, returns the argument, if it is already a RealND.

espressopp.RealND.toRealNDFromVector(**args*)

Try to convert the arguments to a RealND.

This function will only convert to a RealND if x, y and z are specified.

3.9.25 espressopp.System

The main purpose of this class is to store pointers to some important other classes and thus make them available to C++. In a way the System class can be viewed as a container for system wide global variables. If you need to run more than one system at the same time you can combine several systems with the help of the Multisystem class.

In detail the System class holds pointers to:

- the *storage* (e.g. DomainDecomposition)
- the boundary conditions *bc* for the system (e.g. OrthorhombicBC)
- a random number generator *rng* which is for example used by a thermostat
- the *skin* which is needed for the Verlet lists and the cell grid
- a list of short range interactions that apply to the system these interactions are added with the *addInteraction()* method of the System

Example (not complete):

```
>>> LJSystem       = espressopp.System()
>>> LJSystem.bc    = espressopp.bc.OrthorhombicBC(rng, boxsize)
>>> LJSystem.rng
>>> LJSystem.skin = 0.4
>>> LJSystem.addInteraction(interLJ)
```

espressopp.System()

espressopp.System.addInteraction(*interaction*, *name*)

Parameters

- interaction –
- name (string) – The optional name of the interaction.

Return type
bool

espressopp.System.getInteraction(*number*)

Parameters
number –

Return type

espressopp.System.getNumberOfInteractions()

Return type

espressopp.System.removeInteraction(*number*)

Parameters
number –

Return type

espressopp.System.removeInteractionByName(*self*, *name*)

Parameters
name (str) – The name of the interaction to remove.

espressopp.System.getAllInteractions()

Return type
The dictionary with name as a key and Interaction object.

espressopp.System.scaleVolume(**args*)

Parameters
*args –

Return type

espressopp.System.setTrace(*switch*)

Parameters
switch –

3.9.26 espressopp.Tensor

espressopp.Tensor.toTensor(**args*)

Try to convert the arguments to a Tensor, returns the argument, if it is already a Tensor.

espressopp.Tensor.toTensorFromVector(**args*)

Try to convert the arguments to a Tensor.

This function will only convert to a Tensor if x, y and z are specified.

3.9.27 espressopp.VerletList

espressopp.VerletList(*system*, *cutoff*, *exclusionlist*, *useBuffers*, *useSOA*)

Parameters

- system –
- cutoff –
- exclusionlist – (default: [])
- useBuffers – Whether particle neighbors are buffered to improve rebuild times. (default: True)
- useSOA – Whether the alternative structure of arrays form is used for buffers. (default: False)

espressopp.VerletList.exclude(*exclusionlist*)

Parameters
exclusionlist –

Return type

espressopp.VerletList.getAllPairs()

Return type

espressopp.VerletList.localSize()

Return type

espressopp.VerletList.totalSize()

Return type

3.9.28 espressopp.VerletListAdress

The VerletListAdress is the Verlet List to be used for AdResS or H-AdResS simulations. When creating the VerletListAdress one has to provide the system and specify both cutoff for the CG interaction and adrcutoff for the atomistic interaction. Often, it is important to set the atomistic adrcutoff much bigger than the actual interaction's cutoff would be, since also the atomistic part of the VerletListAdress (adrPairs) is built based on the coarse-grained particle positions. For a much larger coarse-grained cutoff it is for example possible to also set the atomistic cutoff on the same value as the coarse-grained one.

Furthermore, the sizes of the explicit and hybrid region have to be provided (dEx and dHy in the example below) and the center of the atomistic region has to be set (adrCenter). Additionally, it can be chosen between a spherical and a slab-like geometry (sphereAdr).

The AdResS region can also be defined based on one or more particles. For a single particle, in this case a spherical region moves along with the particle. For many such region defining particles, the high-resolution/hybrid region corresponds to the overlap of the different spherical regions based on the individual particles (for details see Kreis et al., JCTC doi: 10.1021/acs.jctc.6b00440). Note that more region defining particles mean a higher computational overhead as these particles need to be communicated among all processors (also see explanations in AdResS.py). Also note that region defining particles should be normal/CG particles, not atomistic/AdResS ones.

Bascially the VerListAdress provides 4 lists:

- adrZone: A list which holds all particles in the atomistic and hybrid region
- cgZone: A list which holds all particles in the coarse-grained region
- adrPairs: A list which holds all pairs which have at least one particle in the adrZone, i.e. in the atomistic or hybrid region
- vlPairs: A list which holds all pairs which have both particles in the cgZone, i.e. in the coarse-grained region

Example - creating the VerletListAdress for a slab-type adress region fixed in space (only the x value of adrCenter is used):

```
>>> vl      = espressopp.VerletListAdress(system, cutoff=rc, adrcut=rc, dEx=ex_size, dHy=hy_size,␣
↪adrCenter=[Lx/2, Ly/2, Lz/2])
```

or

```
>>> vl      = espressopp.VerletListAdress(system, cutoff=rc, adrcut=rc, dEx=ex_size, dHy=hy_size,␣
↪adrCenter=[Lx/2, Ly/2, Lz/2], sphereAdr=False)
```

Example - creating the VerletListAdress for a spherical adress region centered on adrCenter and fixed in space:

```
>>> vl      = espressopp.VerletListAdress(system, cutoff=rc, adrcut=rc, dEx=ex_size, dHy=hy_size,␣
↪adrCenter=[Lx/2, Ly/2, Lz/2], sphereAdr=True)
```

Example - creating the VerletListAdress for a spherical adress region centered on one particle and moving with the particle

```
>>> vl      = espressopp.VerletListAdress(system, cutoff=rc, adrcut=rc, dEx=ex_size, dHy=hy_size,␣
↪pids=[adrCenterPID], sphereAdr=True)
```

Example - creating the VerletListAdress for a adress region based on the overlapping spherical regions by several particles

```
>>> vl       = espressopp.VerletListAdress(system, cutoff=rc, adrcut=rc, dEx=ex_size, dHy=hy_size,
↪pids=[adrCenterPID1,adrCenterPID2,adrCenterPID3, ... ], sphereAdr=True)
```

espressopp.VerletListAdress(*system*, *cutoff*, *adrcut*, *dEx*, *dHy*, *adrCenter*, *pids*, *exclusionlist*, *sphereAdr*)

Parameters

- system –
- cutoff –
- adrcut –
- dEx –
- dHy –
- adrCenter – (default: [])
- pids – (default: [])
- exclusionlist – (default: [])
- sphereAdr – (default: False)

espressopp.VerletListAdress.addAdrParticles(*pids*, *rebuild*)

Parameters

- pids –
- rebuild – (default: True)

Return type

espressopp.VerletListAdress.exclude(*exclusionlist*)

Parameters
exclusionlist –

Return type

espressopp.VerletListAdress.rebuild()

Return type

espressopp.VerletListAdress.totalSize()

Return type

3.9.29 espressopp.VerletListTriple

espressopp.VerletListTriple(*system*, *cutoff*, *exclusionlist*)

Parameters

- system –
- cutoff –
- exclusionlist – (default: [])

espressopp.VerletListTriple.exclude(*exclusionlist*)

Parameters
exclusionlist –

Return type

espressopp.VerletListTriple.getAllTriples()

Return type

espressopp.VerletListTriple.localSize()

Return type

espressopp.VerletListTriple.totalSize()

Return type

3.9.30 espressopp.Version

Return version information of espressopp module

Example:

```
>>> version      = espressopp.Version()
>>> print "Name                  = ", version.name
>>> print "Major version number   = ", version.major
>>> print "Minor version number   = ", version.minor
>>> print "Git revision = ", version.gitrevision
>>> print "boost version          = ", version.boostversion
>>> print "Patchlevel             = ", version.patchlevel
>>> print "Compilation date       = ", version.date
>>> print "Compilation time       = ", version.time
```

to print a full version info string:

```
>>> print version.info()
```

espressopp.Version()

3.10 standard_system

3.10.1 espressopp.standard_system.Default

espressopp.standard_system.Default(*box*, *rc=1.12246*, *skin=0.3*, *dt=0.005*, *temperature=None*)

Parameters

- box –
- rc (real) –
- skin (real) –
- dt (real) –

- temperature –

Return default system and integrator, no interactions, no particles are set if tempearture is != None then Langevin thermostat is set to temperature (gamma is 1.0)

3.10.2 espressopp.standard_system.KGMelt

espressopp.standard_system.KGMelt(*num_chains*, *chain_len*)

Parameters

- num_chains –
- chain_len –

3.10.3 espressopp.standard_system.LennardJones

espressopp.standard_system.LennardJones(*num_particles*, *box*, *rc*, *skin*, *dt*, *epsilon*, *sigma*, *shift*, *temperature*, *xyzfilename*, *xyzrfilename*)

Parameters

- num_particles –
- box – (default: (000))
- rc (real) – (default: 1.12246)
- skin (real) – (default: 0.3)
- dt (real) – (default: 0.005)
- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- shift – (default: 'auto')
- temperature – (default: None)
- xyzfilename – (default: None)
- xyzrfilename – (default: None)

return random Lennard Jones system and integrator: if tempearture is != None then Langevin thermostat is set to temperature (gamma is 1.0)

3.10.4 espressopp.standard_system.Minimal

espressopp.standard_system.Minimal(*num_particles*, *box*, *rc*, *skin*, *dt*, *temperature*)

Parameters

- num_particles –
- box –
- rc (real) – (default: 1.12246)
- skin (real) – (default: 0.3)
- dt (real) – (default: 0.005)

- temperature – (default: None)

Return minimal system and integrator whithout any interactions defined: particles have random positions in box if tempearture is != None then Langevin thermostat is set to temperature (gamma is 1.0)

3.10.5 espressopp.standard_system.PolymerMelt

espressopp.standard_system.PolymerMelt(*num_chains*, *monomers_per_chain*, *box*, *bondlen*, *rc*, *skin*, *dt*, *epsilon*, *sigma*, *shift*, *temperature*, *xyzfilename*, *xyzrfilename*)

Parameters

- num_chains –
- monomers_per_chain –
- box – (default: (000))
- bondlen (real) – (default: 0.97)
- rc (real) – (default: 1.12246)
- skin (real) – (default: 0.3)
- dt (real) – (default: 0.005)
- epsilon (real) – (default: 1.0)
- sigma (real) – (default: 1.0)
- shift – (default: 'auto')
- temperature – (default: None)
- xyzfilename – (default: None)
- xyzrfilename – (default: None)

returns random walk polymer melt system and integrator: if tempearture is != None then Langevin thermostat is set to temperature (gamma is 1.0)

3.11 storage

3.11.1 espressopp.storage.DomainDecomposition

espressopp.storage.DomainDecomposition(*system*, *nodeGrid*, *cellGrid*, *halfCellInt*)

Parameters

- system –
- nodeGrid –
- cellGrid –
- halfCellInt (int) – controls the use of half-cells (value 2), third-cells (value 3) or higher. Implicit value 1 for full cells (normal functionality).

espressopp.storage.DomainDecomposition.getCellGrid()

Return type

espressopp.storage.DomainDecomposition.getNodeGrid()

Return type

3.11.2 espressopp.storage.DomainDecompositionAdress

The DomainDecompositionAdress is the Domain Decomposition for AdResS and H- AdResS simulations. It makes sure that tuples (i.e. a coarse-grained particle and its corresponding atomistic particles) are always stored together on one CPU. When setting DomainDecompositionAdress you have to provide the system as well as the nodegrid and the cellgrid.

Example - setting DomainDecompositionAdress:

```
>>> system.storage = espressopp.storage.DomainDecompositionAdress(system, nodeGrid, cellGrid)
```

espressopp.storage.DomainDecompositionAdress(*system*, *nodeGrid*, *cellGrid*, *halfCellInt*)

Parameters

- system –
- nodeGrid –
- cellGrid –
- halfCellInt (int) – controls the use of half-cells (value 2), third-cells (value 3) or higher. Implicit value 1 for full cells (normal functionality).

3.11.3 espressopp.storage.DomainDecompositionNonBlocking

espressopp.storage.DomainDecompositionNonBlocking(*system*, *nodeGrid*, *cellGrid*)

Parameters

- system –
- nodeGrid –
- cellGrid –

3.11.4 espressopp.storage.Storage

This is the base class for all storage objects. All derived classes implement at least the following methods:

- *decompose()*

  Send all particles to their corresponding cell/cpu

- *addParticle(pid, pos)*:

  Add a particle to the storage

- *removeParticle(pid)*:

  Remove a particle with id number *pid* from the storage.

  ```
  >>> system.storage.removeParticle(4)
  ```

  There is an example in *examples* folder

- *getParticle(pid)*:

  Get a particle object. This can be used to get specific particle information:

  ```
  >>> particle = system.storage.getParticle(15)
  >>> print "Particle ID is       : ", particle.id
  >>> print "Particle position is : ", particle.pos
  ```

  you cannot use this particle object to modify particle data. You have to use the modifyParticle command for that (see below).

- *addAdrParticle(pid, pos, last_pos)*:

  Add an AdResS Particle to the storage

- *setFixedTuplesAdress(fixed_tuple_list)*:

- *addParticles(particle_list, *properties)*:

  This routine adds particles with certain properties to the storage.

  param particleList
  list of particles (and properties) to be added

  param properties
  property strings

  Each particle in the list must be itself a list where each entry corresponds to the property specified in properties.

  Example:

  ```
  >>> addParticles([[id, pos, type, ... ], ...], 'id', 'pos', 'type', ...)
  ```

- *modifyParticle(pid, property, value, decompose='yes')*

  This routine allows to modify any properties of an already existing particle.

  Example:

  ```
  >>> modifyParticle(pid, 'pos', Real3D(new_x, new_y, new_z))
  ```

- *removeAllParticles()*:

  This routine removes all particles from the storage.

- 'system':

  The property 'system' returns the System object of the storage.

Examples:

```
>>> s.storage.addParticles([[1, espressopp.Real3D(3,3,3)], [2, espressopp.Real3D(4,4,4)]],'id','pos')
>>> s.storage.decompose()
>>> s.storage.modifyParticle(15, 'pos', Real3D(new_x, new_y, new_z))
```

espressopp.storage.Storage.addAdrATParticle(*pid*, **args*)

Parameters

- pid –
- *args –

Return type

espressopp.storage.Storage.addParticle(*pid*, *pos*)

Parameters

- pid –
- pos –

Return type

espressopp.storage.Storage.addParticles(*particleList*, **properties*)

Parameters

- particleList –
- *properties –

Return type

espressopp.storage.Storage.clearSavedPositions()

Return type

espressopp.storage.Storage.getParticle(*pid*)

Parameters
pid –

Return type

espressopp.storage.Storage.getRealParticleIDs()

Return type

espressopp.storage.Storage.modifyParticle(*pid*, *property*, *value*)

Parameters

- pid –
- property –
- value –

Return type

espressopp.storage.Storage.particleExists(*pid*)

Parameters
pid –

Return type

espressopp.storage.Storage.printRealParticles()

Return type

espressopp.storage.Storage.removeAllParticles()

Return type

espressopp.storage.Storage.removeParticle(*pid*)

Parameters
pid –

Return type

espressopp.storage.Storage.restorePositions()

Return type

espressopp.storage.Storage.savePositions(*idList*)

Parameters
idList –

Return type

espressopp.storage.Storage.setFixedTuplesAdress(*fixedtuples*)

Parameters
fixedtuples –

3.12 tools

3.12.1 information and analysis

Overview

espressopp.tools.analyse

espressopp.tools.info

espressopp.tools.timers

espressopp.tools.vmd

Details

espressopp.tools.analyse

espressopp.tools.analyse.final_info(*system*, *integrator*, *vl*, *start_time*, *end_time*)
final report on the simulation statistics

espressopp.tools.analyse.info(*system*, *integrator*, *per_atom=False*)
reports on the simulation progress

espressopp.tools.info

espressopp.tools.info.getAllBonds(*system*)

return all bonds of the system (currently only FixedPairLists are supported)

espressopp.tools.info.getAllParticles(*system*, **properties*)

returns a list of all particle properties of all particles of the system (currently no atomistic AdResS particles are included)

espressopp.tools.timers

espressopp.tools.timers.show(*alltimers*, *precision=1*)

Python functions to print timings from C++.

espressopp.tools.timers.to_dict(*alltimers*)

Converts timing data into a dictionary for easy post-processing.

espressopp.tools.vmd

espressopp.tools.vmd.connect(*system*, *molsize=10*, *pqrfile=False*, *vmd_path='vmd'*)

Connects to the VMD.

Parameters

- system (espressopp.system) – The system object.
- molsize (int) – The optional size of the molecule.
- pqrfile (bool) – If set to True then the pqr vmd.pqr file will be used otherwise (default) the vmd.pdb file will be used.
- vmd_path (str) – The path to the executable of vmd, by default it is set to 'vmd'.

Returns

Socket to the VMD.

3.12.2 initializing particles

Overview

espressopp.tools.lattice	
espressopp.tools.replicate(bonds, angles, x, ...)	Replicates configuration in each dimension.
espressopp.tools.topology	
espressopp.tools.velocities	
espressopp.tools.warmup(system, integrator)	Warm up for a system with a density of 0.85.

Details

espressopp.tools.lattice

espressopp.tools.lattice.createCubic(*N*, *rho*, *perfect=True*, *RNG=None*)

Initializes particles on the sites of a simple cubic lattice. By setting perfect=False the particle positions will be given random displacements with a magnitude of one-tenth the lattice spacing.

espressopp.tools.lattice.createDiamond(*N*, *rho*, *perfect=True*, *RNG=None*)

Initializes particles on the sites of a diamond lattice.

espressopp.tools.replicate

class espressopp.tools.replicate.ReplicateParallel(**args*, ***kwds*)

Performs the replicate operation in parallel on all workers and creates the replicated particles, bonds and angles only when they are being added to the respective classes. They are also instantiated by batches of the same size as the original seed so memory footprint is reduced.

Usage:

replicate = (2,2,2) bonds, angles, x, y, z, Lx, Ly, Lz = espressopp.tools.lammps.read('polymer_melt.lammps') rp = espressopp.tools.ReplicateParallel() num_particles, Lx, Ly, Lz = rp.replicate(bonds, angles, x, y, z, Lx, Ly, Lz, *replicate) box = (Lx, Ly, Lz) system, integrator = espressopp.standard_system.Default(box=box,...) ... props = ['type', 'mass'] num_particles_seed = len(x) seed_particles = [] for i in range(num_particles_seed):

part = [0, 1.0] seed_particles.append(part)

rp.addParticles(system.storage, 1, seed_particles, *props) system.storage.decompose() ... fpl = espressopp.FixedPairList(system.storage) rp.addBonds(fpl) ... ftl = espressopp.FixedTripleList(system.storage) rp.addTriples(ftl)

For a full working example see testsuite/ReplicateParallel/polymer_melt.py

espressopp.tools.replicate.replicate(*bonds*, *angles*, *x*, *y*, *z*, *Lx*, *Ly*, *Lz*, *xdim=1*, *ydim=1*, *zdim=1*)

Replicates configuration in each dimension.

This may be used to increase the size of an equilibrated melt by a factor of 8 or more.

Presently this routine works only for semiflexible polymers. A general class should be written to deal with files containing coordinates and topology data.

xdim = ydim = zdim = 1 returns the original system not replicated. xdim = ydim = zdim = 2 returns the original system replicated to 8x. xdim = ydim = zdim = 3 returns the original system replicated to 27x. xdim = ydim = 1, zdim = 2 returns the original system replicated in the z-direction.

espressopp.tools.topology

espressopp.tools.topology.polymerRW(*pid*, *startpos*, *numberOfMonomers*, *bondlength*, *return_angles=False*, *return_dihedrals=False*, *mindist=None*, *rng=None*)

Initializes polymers through random walk

espressopp.tools.velocities

espressopp.tools.velocities.gaussian(*T*, *N*, *particle_mass=None*, *zero_momentum=True*, *seed=7654321*, *kb=1.0*)

Generates velocities with temperature T according to a Maxwell-Boltzmann distribution.

Args:

T: The desired temperature expre. N: The number of particles. particle_mass: The list of particle mass if not then every particle has mass 1.0 zero_momentum: Remove the center-of-mass motion. seed: The seed for the random number generator. kb: The Boltzmann constant.

Returns:

The tuple with lists of x,y,z components of the velocity.

espressopp.tools.warmup

espressopp.tools.warmup.warmup(*system*, *integrator*, *number=80*)

Warm up for a system with a density of 0.85.

The method needs the following parameters:

- system, integrator ESPResSo system which schoul be warmed up and the correspondig integrator e.g.:

```
>>> system, integrator = espressopp.standard_system.LennardJones(100,(10,10,10))
```

- number number of steps of the warm up

  for a system with a density of 0.85, if it explodes try a higher number

3.12.3 decomp - Domain Decomposition python functions

- *nodeGrid(n,box_size,rc,skin,eh_size=0,ratioMS=0,idealGas=0,slabMSDims=[0,0,0])*:

  It determines how the processors are distributed and how the cells are arranged. The algorithm is dimensional sensitive for both homogeneous and inhomogeneous setups. On top of such functionality it presents specific features for region divided heterogenous setups (e.g. for AdResS) [see H.V. Guzman et. al, Phys. Rev. E, 96, 053311 (2017)] Link to the paper: https://doi.org/10.1103/PhysRevE.96.053311

  box_size - 3D vector cointainig the size of the simulation box box_size [L_x,L_y,L_z] *rc* - cutoff radius of interaction *skin* - skin size for the verlet list calculation *n* - total number of processes *eh_size* - 1D length of the high-resolution region (e.g. for AdResS Atomistic/Explicit+Hybrid regions) *ratioMS* - spatial mapping ratio between high-resolution region and the low-resolution one

  (e.g. for AdResS mapping the atomistic water molecule to the CG-model; leads to a ratioMS=3)

  ***idealGas* - this is a Flag for treating the low-resolution region as an Ideal Gas, when TRUE (no interactions included,**
  thus none force computations load)

  slabMSDims - 3D vector cointainig flags describing the type of axis, if heterogeneous value is 1, else 0

- ***nodeGridSimple(n)*:**
  It determines how the processors are distributed and how the cells are arranged. Note: Use it exclusively for Lattice-Boltzmann simulations, or non-parallelized tests. *n* - number of processes

- *cherrypickTotalProcs(box_size,rc,skin,MnN,CpN,percTol=0.2,eh_size=0,ratioMS=0,idealGas=0,slabMSDims=[0,0,0])*:

To be used for heterogenous simulations where the spatial heterogeinity is known on an a-priori manner, where this function returns *n* as the total number of processes to be used for the best decomposition of the system as a function of a tolerance ratio which depending on the giving range [0, MnN*CpN] of processors availability different combinations of P_x,P_y and P_z can be found and hence several values of *n* this 'n' can become an array. Most of the parameters have been described for nodeGrid(...), except:

MnN - M number of Nodes to be available (e.g. 128 processes/cores, 16 cores per Node gives a total of 8 Nodes) *CpN* - C number of Cores available in each Node (e.g. 16 Cores per Node or 20 Cores per Node) *percTol* - Axis base tolerance percentage to the ideal distribution of P-processors per axes P_x,P_y,P_z

- *neiListHom(node_grid,box,rc,skin)*:

  The new domain decomposition divides the subdomains in a neighborlist of corse in a grid of 3 arrays [N_x,N_y,N_z]. In this case, the neighbor list is homogeneous (non a-priori load imbalance). Most of the parameters have been described above, except:

  node_grid - M number of Nodes to be available (e.g. 128 processes/cores, 16 cores per Node gives a total of 8 Nodes)

- *neiListAdress(node_grid, cell_grid,rc,skin,eh_size,adrCenter,ratioMS,idealGasFlag=True,sphereAdr=False,slabMSDims=[1,0,0*

  The new domain decomposition divides the subdomains in a neighborlist of corse in a grid of 3 arrays [N_x,N_y,N_z]. In this case, the neighbor list is homogeneous (non a-priori load imbalance). Most of the parameters have been described above, except:

  cell_grid - Based on the homogenous allocation of cells per subdomain, a referential value *adrCenter* - Box center of the heterogeneous simulation box [adrCenter_x,adrCenter_y,adrCenter_z], commonly the middle of

  high-resolution region.

  ***idealGasFlag* - This is a Flag for treating the low-resolution region as an Ideal Gas (no interactions included,**
  thus less load)

  sphereAdr - Geometry of the high-resolution region, if TRUE spherical, otherwise Slab-like

- *cellGrid(box_size, node_grid, rc, skin, halfCellInt)*:

  It returns an appropriate grid of cells. *halfCellInt* - controls the use of half-cells (value 2), third-cells (value 3) or higher. Implicit value 1 for full cells (normal functionality).

- *tuneSkin(system, integrator, minSkin=0.01, maxSkin=1.2, precision=0.001)*:

  It tunes the skin size for the current system

- *printTimeVsSkin(system, integrator, minSkin=0.01, maxSkin=1.5, skinStep = 0.01)*:

  It prints time of running versus skin size in the range [minSkin, maxSkin] with the step skinStep

3.12.4 DumpConfigurations - read/write xyz files

espressopp.tools.DumpConfigurations.fastreadxyz(*filename*)

espressopp.tools.DumpConfigurations.fastwritexyz(*filename*, *system*, *velocities=True*, *unfolded=True*, *append=False*, *scale=1.0*)

espressopp.tools.DumpConfigurations.fastwritexyz_standard(*filename*, *system*, *unfolded=False*, *append=False*)

Fast write standard xyz file. Generally standard xyz file is

```
>>> number of particles
>>> comment line
>>> type x y z
>>> ......
>>> ......
>>> ......
```

Additional information can be found here: Wiki: http://en.wikipedia.org/wiki/XYZ_file_format OpenBabel: http://openbabel.org/wiki/XYZ_%28format%29

In this case one can choose folded or unfolded coordinates. Currently it writes only particle type = 0 and pid is a line number. Later different types should be implemented.

espressopp.tools.DumpConfigurations.readxyz(*filename*)

espressopp.tools.DumpConfigurations.readxyzr(*filename*)

espressopp.tools.DumpConfigurations.writexyz(*filename*, *system*, *velocities=True*, *unfolded=False*, *append=False*)

espressopp.tools.DumpConfigurations.xyzfilewrite(*filename*, *system*, *append=False*, *atomtypes={0: 'Fe', 1: 'O', 2: 'C'}*, *velocities=False*, *charge=False*)

This method creates a xyz file with the data from a specific system: 1. row: number of the atoms 2. row: REMARK generated by ESPResSo++ following rows: atomsymbol positionX positionY positionZ (velocityX velocityY velocityZ) (charge) last row: END

The method needs the following parameters:

- filename

  name of the file where the table schould be saved in

- system

  ESPResSo system which creates the data e.g.:

  ```
  >>> system, integrator = espressopp.standard_system.LennardJones(100,(10,10,10))
  ```

- append

  =False
  : the data in the file will be overwritten

  =True
  : the data will be appended

- atomtypes the xyz file needs atom symbols, so it has to translate the numbers insert a dictionary with the right translation

- velocities

=False
does not save the velocity vectors

=True
creates collumns for the velocity vectors and saves the data

- charge

=False
does not save the charge

=True
creates collumns for the charges and saves the data

3.12.5 espresso_old - read espressomd files

This Python module allows one to use ESPResSo data files as the input to an ESPResSo++ simulation.

espressopp.tools.espresso_old.read(*file*)

Read ESPResSo data files.

Keyword argument: file – contains simulation variables, data of all particles, and information about bonds. (angles and dihedrals are currently not read)

3.12.6 gromacs - parser for Gromacs files

This Python module allows one to use GROMACS data files as the input to an ESPResSo++ simulation, set interactions for given particle types and convert GROMACS potential tables into ESPResSo++ tables. It containts functions: read(), setInteractions(), convertTable()

Some tips for using the gromacs parser:

Tip 1.

topol.top includes solvent via #include statements

If the included .itp file ONLY contains the solvent molecule you're using (e.g. spc/e water using spce.itp) then this is okay.

But if the .itp file contains info about many molecules (e.g. you want to use one ion from ions.itp), then gromacs.py will just take the first one listed. You must edit your topol.top file to explicitly include the solvent molecule you're using.

e.g. replace:

```
; Include topology for ions
#include "amber03.ff/ions.itp"
```

by:

```
; Include topology for ions
[ moleculetype ]
; molname       nrexcl
CL              1

[ atoms ]
; id    at type         res nr  residu name     at name  cg nr  charge
1       Cl              1       CL              CL       1      -1.00000
```

Tip 2. impropers

impropers in the topol.top file (function type 4) need to be labelled '[impropers]', not '[dihedrals]' as in standard gromacs format"

Also, the dihedrals should be listed before the impropers (this is usuall the case by default in gromacs-format files).

Tip 3.

For rigid SPC/E water using Settle, spce.itp file should look like this:

```
[ moleculetype ]
; molname       nrexcl
SOL             2

[ atoms ]
; id  at type     res nr  res name  at name  cg nr  charge    mass
  1   OW_spc       1       SOL       OW       1      -0.8476   15.99940
  2   HW_spc       1       SOL       HW1      1       0.4238    1.00800
  3   HW_spc       1       SOL       HW2      1       0.4238    1.00800

[ bonds ]
; i     j       funct   length  force.c.
1       2       1       0.1     345000  0.1     345000
1       3       1       0.1     345000  0.1     345000

[ angles ]
; i     j       k       funct   angle   force.c.
2       1       3       1       109.47  383     109.47  383
```

The bonds section is used to generate exclusions, but bond and angle parameters are not relevant if the Settle extension is used. The geometry is that specified in the python script when adding the Settle extension

Include modified spce file in topol.top, e.g. replace

#include "amber03.ff/spce.itp"

by

#include "amber03.ff/spce-for-espressopp.itp"

Tip 4.

Use absolute paths for any include files which are not in the standard gromacs topology directory ($GMXLIB)

e.g. replace

#include "mynewresidue.itp"

by

#include "path/to/mynewres/file/mynewresidue.itp"

Tip 5.

The parser won't work if the particles ids in the include files conflict with the particle ids in the topol.top file itself, and the bonded interaction parameters in the itp file need to be looked up via particle type in the standard gromacs topology directory ($GMXLIB)

i.e. Okay for an itp file like spce.itp above, where the bonds and angles parameters are given in the itp file, as in:

```
[ bonds ]
; i     j       funct   length  force.c.
1       2       1       0.1     345000  0.1     345000
```

Not okay for an itp file containing lines like:

```
[ bonds ]
; i     j       funct   length  force.c.
1       2       1
```

espressopp.tools.gromacs.convertTable(*gro_in_file*, *esp_out_file*, *sigma=1.0*, *epsilon=1.0*, *c6=1.0*, *c12=1.0*)

Convert GROMACS tabulated file into ESPResSo++ tabulated file (new file is created). First column of input file can be either distance or angle. For non-bonded files, c6 and c12 can be provided. Default value for sigma, epsilon, c6 and c12 is 1.0. Electrostatics are not taken into account (f and fd columns).

Keyword arguments: gro_in_file – the GROMACS tabulated file name (bonded, nonbonded, angle or dihedral). esp_out_file – filename of the ESPResSo++ tabulated file to be written. sigma – optional, depending on whether you want to convert units or not. epsilon – optional, depending on whether you want to convert units or not. c6 – optional c12 – optional

espressopp.tools.gromacs.read(*gro_file*, *top_file=''*, *doRegularExcl=True*)

Read GROMACS data files.

Arguments: :param gro_file: – contains coordinates of all particles, the number of particles, velocities and box size. :type gro_file: string :param top_file: – contains topology information. Included topology files (.itp) are also read :type gro_file: string :param doRegularExcl: – if True, exclusions are generated automatically based on the nregxcl parameter (see gromacs manual) :type doRegularExcl: bool

espressopp.tools.gromacs.setLennardJones14Interactions(*system*, *defaults*, *atomtypeparams*, *onefourlist*, *cutoff*)

Set lennard jones interactions which were read from gromacs based on the atomypes

espressopp.tools.gromacs.setLennardJonesInteractions(*system*, *defaults*, *atomtypeparams*, *verletlist*, *cutoff*, *hadress=False*, *adress=False*, *ftpl=None*)

Set lennard jones interactions which were read from gromacs based on the atomypes

espressopp.tools.gromacs.setLennardJonesInteractionsTI(*system*, *defaults*, *atomtypeparams*, *verletlist*, *cutoff*, *epsilonB*, *sigmaSC*, *alphaSC*, *powerSC*, *lambdaTI*, *pidlist*, *annihilate=True*, *hadress=False*, *adress=False*, *ftpl=None*)

Set lennard jones interactions which were read from gromacs based on the atomypes

espressopp.tools.gromacs.setTabulatedInteractions(*potentials*, *particleTypes*, *system*, *interaction*)

Set interactions for all given particle types. Return value is a system with all interactions added.

Keyword arguments: potentials – is a dictionary where key is a string composed of two particle types and value is a potential. example: {“A_A”:potAA, “A_B”:potAB, “B_B”:potBB} particleTypes – is a dictionary where key is the particle type, and value is a list of particles of that type. example: {“A”:[“A1m”, “A2m”],”B”:[“B1u”,”B2u”]} system – is the system to which the interaction will be added interaction – is the interaction to which to add the potentials

3.12.7 io_extended - read/write configurational files

This Python module allows one to read and write configurational files. One can choose folded or unfolded coordinates and write down velocities or not. It is similar to lammps read and write, but it writes down only: 1) number of particles + types 2) number of bonds (number of pairs) + types 3) number of angles (number of triples) + types 4) number of dihedrals (number of quadruples) + types 5) system size (Lx,Ly,Lz) 6) p_id, p_type, p_positions 7) velocities (if true) 8) bonds (if exist) 9) angles (if exist) 10)dihedrals (if exist)

read returns: Lx, Ly, Lz, p_ids, p_types, poss, vels, bonds, angles, dihedrals if something does not exist then it will return the empty list bonds, angles, dihedrals - will return list [type, (x,x,x,x)], where type is the type of bond, angle or dihedral (x,x,x,x) is (pid1,pid2) for bonds,

(pid1,pid2,pid3) for angles

(pid1,pid2,pid3,pid4) for dihedrals

3.12.8 lammps - read lammps files

This Python module allows one to use a LAMMPS data file as the input to an ESPResSo++ simulation.

3.12.9 pathintegral - nuclear quantum effects

- method to automatically run the system including nuclear quantum effects using the Feynman path-integral

!!WARNING: THIS IS STILL AN EXPERIMENTAL FEATURE!!

This method creates, based on the supplied topology of the system, an path-integral representation with P beads. The path-integral system is a fully classical analog, which has to be run at an effective temperature P*T.

The method needs the following parameters:

- **allParticles**
  particles of the sytem
- **props**
  particle properties
- **types**
  types, e.g. read from the gromacs parser
- system
- **exclusions**
  non-bonded exclusions
- integrator
- **langevin**
  langevin integrator
- **rcut**
  the cutoff used for the rings non-bonded interactions
- **P**
  the Trotter Number (number of imaginary time slices)
- **polymerInitR**
  polymer radius for setting up ring in 2d plane
- **hbar**
  hbar in gromacs units [kJ/mol ps]

- **disableVVl**
  disable Virtual Verlet List (slow but safe). If false, the neighbour search is based on the VirtualParticles extension, which contain the rings. This speeds up neighbour search significantly.

3.12.10 PDB - read and write pdb format

espressopp.tools.pdbwrite(*filename*, *system*, *molsize=4*, *append=False*, *typenames=None*)

Writes a file in PDB format

Parameters

- filename (string) – output file name
- system (espressopp System object) – espressopp system
- molsize (int) – if molsize>0, the molecule count is increased every molsize particles (default 4)
- append (bool) – if True, append to filename, other over-write filename (default False)
- typenames (dict, key=int, value=string) – dictionary used for mapping from espressopp's integer particle types to the particle type strings written in a pdb file

espressopp.tools.pdbread(*filename*, *natoms*, *header*)

Reads one frame of a pdb format file

Parameters

- filename (string) – input file name
- natoms (int) – number of atoms in pdf file
- header (int) – number of header lines to skip at start of file

Returns: index,atomname,locator,resname,resid,resseq,x,y,z,alpha,beta,segid,element (lists of type int,str,str,str,str,int,float,float,float,float,float,str,str)

3.12.11 povwrite - write povray files

3.12.12 prepareComplexMolecules - set up proteins

various helper functions for setting up systems containing complex molecules such as proteins

espressopp.tools.findConstrainedBonds(*atomPids*, *bondtypes*, *bondtypeparams*, *masses*, *massCutoff=1.1*)

Finds all heavyatom-hydrogen bonds in a given list of particle IDs, and outputs a list describing the bonds, in a format suitable for use with the RATTLE algorithm for constrained bonds

Parameters

- atomPids (list of int) – list of pids of atoms between which to search for bonds
- bondtypes (dict, key: (int,int), value: int) – dictionary mapping from tuple of pids to bondtypeid, e.g. as returned by tools.gromacs.read()
- bondtypeparams (dict, key: int, value: espressopp bond type) – dictionary mapping from bondtypeid to class storing parameters of that bond type, e.g. as returned by tools.gromacs.read()
- masses (list of float) – list of masses, e.g. as returned by tools.gromacs.read()

- massCutoff (float) – for identifying light atoms (hydrogens), default 1.1 mass units, can also be increased e.g. for use with deuterated systems

Returns

hydrogenIDs - list of pids (integer) of light atoms (hydrogens)

constrainedBondsDict - dict, keys: pid (integer) of heavy atom, values: list of pids of light atoms that are bonded to it

constrainedBondsList - list of lists, one entry for each constrained bond with format: [pid of heavy atom, pid of light atom, bond distance, mass of heavy atom ,mass of light atom]

Can then be used with RATTLE, e.g.

```
>>> rattle = espressopp.integrator.Rattle(system, maxit = 1000, tol = 1e-6, rptol = 1e-6)
>>> rattle.addConstrainedBonds(constrainedBondsList)
>>> integrator.addExtension(rattle)
```

espressopp.tools.getInternalNonbondedInteractions(*atExclusions*, *pidlist*)

Gets the non-bonded pairs within a list of particle indices, excluding those which are in a supplied list of exclusions. Useful for example for getting the internal atomistic non-bonded interactions in a coarse-grained particle and adding them as a fixedpairlist

Parameters

- atExclusions (list of 2-tuples of int) – list of excluded pairs
- pidlist (list of int) – list of pids among which to create pairs

Returns
list of pairs which are not in atExclusions

Return type
list of 2-tuples of int

espressopp.tools.readSimpleSystem(*filename*, *nparticles*, *header*)

Read in a column-formatted file containing information about the particles in a simple system, for example a coarsegrained protein.

This function expects the input file to have between 2 and 5 columns. The number of columns in the file is automatically detected. The function reads each column into a list and returns the lists. Column types are interpreted as follows: 2 columns: float, float 3 columns: float, float, int 4 columns: float, float, int, str 5 columns: float, float, int, str, str

For example in the case of a coarsegrained protein model, these could be: mass, charge, corresponding atomistic index, beadname, beadtype

Parameters

- filename (string) – name of file to open and read
- nparticles (int) – number of particles in file
- header (int) – number of lines to skip at start of file (default 0)

Returns: between 2 and 5 lists

espressopp.tools.applyBoreschRestraints(*system*, *restraintAtoms*, *restraintK*, *restraintR0*)

Applies restraints between ligand and protein as defined in Boresch et al, JPCB 2003, 107, 9535-9551 The restraints (one bond, two angles and three dihedrals) are applied between three ligand atoms A,B,C and three protein atoms a,b,c.

In espressopp, the potential for harmonic bond and angles is k(x-x0)^2, but the harmonic dihedral potential is 0.5*k(x-x0)^2. This is taken care of in this function, i.e. the user should supply the force constants exactly as given in Boresch et al.

Parameters

- system (System object) – espressopp system
- restraintAtoms (python dictionary) – dictionary identifying the six atoms involved in the restraints, key = atom label (one of 'A','B','C','a','b','c'), value = atom index
- restraintK (python dictionary) – dictionary of force constants, key = restraint label ('aA' for the bond, 'baA' and 'aAB' for the angles, 'aABC', 'cbaA' and 'baAB' for the dihedrals), value = force constant (units as for the simulation forcefield, angle and dihedral constants should be in rad^-2)
- restraintR0 (python dictionary) – dictionary of equilibrium values (distance or angle), key = restraint label, value = distance (in distance units) or angle (in degrees)

Examples of the three dictionaries:

```
>>> restraintAtoms = {'A':1981,'B':1966,'C':1993,'a':1588,'b':1581,'c':1567}
>>> restraintK = {'aA':4184,'baA':41.84,'aAB':41.84,'aABC':41.84,'cbaA':41.84,'baAB':41.84}
↪#kJ mol-1 nm-2,kJ mol-1 rad-2, all 10 kcal as in JPCB 2003
>>> restraintR0 = {'aA':0.31,'baA':120.0,'aAB':90.0,'aABC':100.0,'cbaA':-170.0,'baAB':-105.0}
↪#nm, degrees
```

3.12.13 prepareAdress - setup AdResS simulation

Auxiliary python functions for preparation of an Adress Simulation based on a configuration from an all-atomistic simulation.

If one uses a configuration file from an all-atomistic simulation as start configuration for an AdResS simulation, the particles are probably all located inside the simulation box. However, in AdResS only the coarse-grained center-of-mass particles have to be in the box, the atomistic particles of the coarse grained might be outside around their CoM CG particle. When in the start configuration atomistic particles belonging to a molecule are folded such that some of the atoms are on the one side of the box while the others are folded to the other side the calculation of the center of mass goes wrong and the simulation will be incorrect. This script ensures a proper center of mass calculation and a proper folding and configuration for the AdResS simulation by simply putting the CG particle in one of the atoms (AdressSetCG) first. Then the molecules will be put together properly afterwards when calling AdressDecomp.

3.12.14 PSF - read and write psf format

PSF file format given at http://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-win-html/node24.html

espressopp.tools.psfwrite(*filename*, *system*, *maxdist=None*, *molsize=4*, *typenames=None*)

Writes !NATOM and !NBOND sections of a psf format file

Parameters

- filename (string) – output file name

- system (espressopp System object) – espressopp system
- maxdist (float) – if this is specified, only bonds in which the pair of particles are separated by a distance < maxdist are written to the !NBOND section
- molsize (int) – if molsize>0, the molecule count is increased every molsize particles
- typenames (dict, key=int, value=string) – dictionary used for mapping from espressopp's integer particle types to the particle type strings written in a psf file

espressopp.tools.psfread(*filename*)

Reads !NATOM section of a psf format file

Parameters
filename (string) – input file name

Returns
pid,segname,resindex,resname,atomname,atomtype,mass,charge (lists of type int,str,int,str,str,str,float,float)

3.12.15 tabulated - write tabulated file

espressopp.tools.tabulated.writeTabFile(*pot*, *name*, *N*, *low=0.0*, *high=2.5*, *body=2*)

writeTabFile can be used to create a table for any potential Parameters are: * pot : this is any espressopp.interaction potential * name : filename * N : number of line to write * low : lowest r (default is 0.0) * high : highest r (default is 2.5)

This function has not been tested for 3 and 4 body interactions

3.12.16 topology_helper

3.12.17 units - convert to real units

Espresso++ returns temperature, energy, pressure, box length etc. in dimensionless units. Usually user should take care about real length, energy, mass and charge units. This python class is a helper in order to simplify the conversion which is based on basic units. However, user always should use it carefully for complicated systems.

Currently it is implemented for SI units. Make sure that you are using length in [nm] energy in [kJ/mol] mass in [amu] q in [e]

and it will return you pressure in [bar] temperature in [K] time in [ps] density in [kg/m^3]

Example:

3.13 Logging mechanism

ESPResSo++ uses Loggers

Logging can be switched on in your python script with the following command:

```
>>> logging.getLogger("*name of the logger*").setLevel(logging.*Level*)
```

Level is one of the following:

ERROR	for errors that might still allow the application to continue
WARN	for potentially harmful situations
INFO	informational messages highlighting progress
DEBUG	designates fine-grained informational events

Example:

```
>>> import espressopp
>>> import logging
>>> logging.getLogger("Storage").setLevel(logging.ERROR)
```

To log everything (WARNING: this will produce **lots** of output):

```
>>> logging.getLogger("").setLevel(logging.DEBUG)
```

The following loggers are currently available:

- Configurations
- Observable
- Velocities
- BC
- Logger
- FixedListComm
- FixedPairList
- FixedQuadrupleList
- FixedTripleList
- FixedTupleList
- Langevin
- MDIntegrator
- AngularPotential
- DihedralPotential
- Interaction
- InterpolationAkima
- InterpolationCubic
- InterpolationLinear
- InterpolationTable
- Potential
- CellListAllPairsIterator
- DomainDecomposition.CellGrid
- DomainDecomposition
- DomainDecomposition.NodeGrid
- Storage

- DomainDecompositionAdress
- StorageAdress
- VerletList
- VerletList

CHAPTER

FOUR

CREDITS

4.1 ESPResSo++ Developers

The core of the developer team comes from the Polymer Theory Group of Prof. Kurt Kremer at the Max Planck Institute for Polymer Research in Mainz.

A full list of active and former developers is available at the main website of ESPResSo++.

4.2 FAQ

A short list of frequently asked questions is given here.

4.3 Getting Help

If you have any questions do not hesitate to contact us.

BIBLIOGRAPHY

[Berendsen84] Berendsen et al., *J. Chem. Phys.*, 81, **1984**, p. 3684

[Quigley04] D. Quigley, M.I.J. Probert, *J. Chem. Phys.*, 120, **2004**, p. 11432

[Martyna94] G. Martyna, D. Tobias, M. Klein, *J. Chem. Phys.*, 101, **1994**, p. 4177

[Kremer_1986] Grest, G. S. and Kremer, K. (1986). Molecular dynamics simulation for polymers in the presence of a heat bath. Physical Review A, 33(5), 3628-3631. https://doi.org/10.1103/PhysRevA.33.3628

[Allen89] M.P.Allen, D.J.Tildesley, *Computer simulation of liquids*, *Clarendon Press*, **1989** 385 p.

[Deserno98] M.Deserno and C.Holm, *J. Chem. Phys.*, 109(18), **1998**, p.7678

[Zhang_2014] G. Zhang, L. A. Moreira, T. Stuehn, K. Ch. Daoulas, and K. Kremer, Equilibration of high molecular weight polymer melts: A hierarchical strategy, Macro Lett., 2014, 3, 198

PYTHON MODULE INDEX

e

espressopp.analysis.AdressDensity, 46
espressopp.analysis.AllParticlePos, 33
espressopp.analysis.AnalysisBase, 39
espressopp.analysis.Autocorrelation, 40
espressopp.analysis.CenterOfMass, 46
espressopp.analysis.CMVelocity, 40
espressopp.analysis.ConfigsParticleDecomp, 41
espressopp.analysis.Configurations, 41
espressopp.analysis.ConfigurationsExt, 42
espressopp.analysis.Energy, 43
espressopp.analysis.IntraChainDistSq, 44
espressopp.analysis.LBOutput, 33
espressopp.analysis.LBOutputScreen, 34
espressopp.analysis.LBOutputVzInTime, 34
espressopp.analysis.LBOutputVzOfX, 35
espressopp.analysis.MaxPID, 47
espressopp.analysis.MeanSquareDispl, 44
espressopp.analysis.MeanSquareInternalDist, 45
espressopp.analysis.NeighborFluctuation, 47
espressopp.analysis.NPart, 47
espressopp.analysis.NPartSubregion, 45
espressopp.analysis.Observable, 51
espressopp.analysis.OrderParameter, 35
espressopp.analysis.ParticleRadiusDistribution, 35
espressopp.analysis.PotentialEnergy, 47
espressopp.analysis.Pressure, 47
espressopp.analysis.PressureTensor, 35
espressopp.analysis.PressureTensorLayer, 36
espressopp.analysis.PressureTensorMultiLayer, 37
espressopp.analysis.RadGyrXProfilePI, 51
espressopp.analysis.RadialDistrF, 48
espressopp.analysis.RDFatomistic, 48
espressopp.analysis.StaticStructF, 50
espressopp.analysis.SubregionTracking, 52
espressopp.analysis.SystemMonitor, 53
espressopp.analysis.Temperature, 39
espressopp.analysis.Test, 39
espressopp.analysis.Velocities, 54
espressopp.analysis.VelocityAutocorrelation, 54
espressopp.analysis.Viscosity, 54
espressopp.analysis.XDensity, 50
espressopp.analysis.XPressure, 51
espressopp.analysis.XTemperature, 51
espressopp.bc.BC, 55
espressopp.bc.OrthorhombicBC, 56
espressopp.bc.SlabBC, 56
espressopp.check.System, 57
espressopp.esutil.Collectives, 57
espressopp.esutil.GammaVariate, 57
espressopp.esutil.Grid, 57
espressopp.esutil.NormalVariate, 58
espressopp.esutil.RNG, 58
espressopp.esutil.UniformOnSphere, 58
espressopp.Exceptions, 210
espressopp.external.transformations, 58
espressopp.FixedLocalTupleList, 211
espressopp.FixedPairDistList, 211
espressopp.FixedPairList, 212
espressopp.FixedPairListAdress, 213
espressopp.FixedQuadrupleAngleList, 213
espressopp.FixedQuadrupleList, 214
espressopp.FixedQuadrupleListAdress, 215
espressopp.FixedSingleList, 215
espressopp.FixedTripleAngleList, 216
espressopp.FixedTripleList, 217
espressopp.FixedTripleListAdress, 217
espressopp.FixedTupleList, 218
espressopp.FixedTupleListAdress, 218
espressopp.Int3D, 218
espressopp.integrator.Adress, 74
espressopp.integrator.AssociationReaction, 75
espressopp.integrator.BerendsenBarostat, 75
espressopp.integrator.BerendsenBarostatAnisotropic, 76
espressopp.integrator.BerendsenThermostat, 78
espressopp.integrator.CapForce, 79
espressopp.integrator.DPDThermostat, 80
espressopp.integrator.EmptyExtension, 80
espressopp.integrator.ExtAnalyze, 80
espressopp.integrator.Extension, 80
espressopp.integrator.ExtForce, 80
espressopp.integrator.FixPositions, 81
espressopp.integrator.FreeEnergyCompensation, 81

espressopp.integrator.GeneralizedLangevinThermostat, 82
espressopp.integrator.Isokinetic, 82
espressopp.integrator.LangevinBarostat, 82
espressopp.integrator.LangevinThermostat, 84
espressopp.integrator.LangevinThermostat1D, 85
espressopp.integrator.LangevinThermostatHybrid, 85
espressopp.integrator.LangevinThermostatOnGroup, 86
espressopp.integrator.LangevinThermostatOnRadius, 86
espressopp.integrator.LatticeBoltzmann, 87
espressopp.integrator.LBInit, 91
espressopp.integrator.LBInitConstForce, 92
espressopp.integrator.LBInitPeriodicForce, 93
espressopp.integrator.LBInitPopUniform, 92
espressopp.integrator.LBInitPopWave, 92
espressopp.integrator.MDIntegrator, 93
espressopp.integrator.MinimizeEnergy, 93
espressopp.integrator.OnTheFlyFEC, 95
espressopp.integrator.PIAdressIntegrator, 95
espressopp.integrator.Rattle, 105
espressopp.integrator.Settle, 106
espressopp.integrator.StochasticVelocityRescaling, 107
espressopp.integrator.TDforce, 107
espressopp.integrator.VelocityVerlet, 108
espressopp.integrator.VelocityVerletOnGroup, 109
espressopp.integrator.VelocityVerletOnRadius, 109
espressopp.integrator.VelocityVerletRESPA, 108
espressopp.interaction.AngularCosineSquared, 110
espressopp.interaction.AngularHarmonic, 112
espressopp.interaction.AngularPotential, 110
espressopp.interaction.ConstrainCOM, 131
espressopp.interaction.ConstrainRG, 132
espressopp.interaction.Cosine, 113
espressopp.interaction.CoulombKSpaceEwald, 124
espressopp.interaction.CoulombKSpaceP3M, 126
espressopp.interaction.CoulombRSpace, 127
espressopp.interaction.CoulombTruncated, 129
espressopp.interaction.CoulombTruncatedUniqueCharges, 130
espressopp.interaction.DihedralHarmonic, 133
espressopp.interaction.DihedralHarmonicCos, 134
espressopp.interaction.DihedralHarmonicNCos, 134
espressopp.interaction.DihedralPotential, 135
espressopp.interaction.DihedralRB, 136
espressopp.interaction.FENE, 114
espressopp.interaction.FENECapped, 115
espressopp.interaction.GravityTruncated, 146
espressopp.interaction.Harmonic, 116
espressopp.interaction.HarmonicTrap, 120
espressopp.interaction.Interaction, 192
espressopp.interaction.LennardJones, 147
espressopp.interaction.LennardJones93Wall, 191
espressopp.interaction.LennardJonesAutoBonds, 158
espressopp.interaction.LennardJonesCapped, 160
espressopp.interaction.LennardJonesEnergyCapped, 163
espressopp.interaction.LennardJonesExpand, 166
espressopp.interaction.LennardJonesGeneric, 167
espressopp.interaction.LennardJonesGromacs, 170
espressopp.interaction.LennardJonesSoftcoreTI, 172
espressopp.interaction.LJcos, 174
espressopp.interaction.MirrorLennardJones, 176
espressopp.interaction.Morse, 121
espressopp.interaction.OPLS, 137
espressopp.interaction.Potential, 193
espressopp.interaction.PotentialVSpherePair, 193
espressopp.interaction.Quartic, 193
espressopp.interaction.ReactionFieldGeneralized, 194
espressopp.interaction.ReactionFieldGeneralizedTI, 197
espressopp.interaction.SingleParticlePotential, 199
espressopp.interaction.SmoothSquareWell, 203
espressopp.interaction.SoftCosine, 123
espressopp.interaction.StillingerWeberPairTerm, 138
espressopp.interaction.StillingerWeberPairTermCapped, 140
espressopp.interaction.StillingerWeberTripleTerm, 142
espressopp.interaction.Tabulated, 177
espressopp.interaction.TabulatedAngular, 186
espressopp.interaction.TabulatedDihedral, 190
espressopp.interaction.TersoffPairTerm, 143
espressopp.interaction.TersoffTripleTerm, 145
espressopp.interaction.VSpherePair, 200
espressopp.interaction.VSphereSelf, 201
espressopp.interaction.Zero, 201
espressopp.io.DumpGRO, 204
espressopp.io.DumpGROAdress, 205
espressopp.io.DumpH5MD, 208
espressopp.io.DumpXYZ, 206
espressopp.MultiSystem, 219
espressopp.ParallelTempering, 220
espressopp.Particle, 221
espressopp.ParticleAccess, 223
espressopp.ParticleGroup, 223
espressopp.pmi, 224
espressopp.Quaternion, 228
espressopp.Real3D, 230
espressopp.RealND, 231
espressopp.standard_system.Default, 236
espressopp.standard_system.KGMelt, 237
espressopp.standard_system.LennardJones, 237
espressopp.standard_system.Minimal, 237

espressopp.standard_system.PolymerMelt, 238
espressopp.storage.DomainDecomposition, 238
espressopp.storage.DomainDecompositionAdress, 239
espressopp.storage.DomainDecompositionNonBlocking, 239
espressopp.storage.Storage, 239
espressopp.System, 231
espressopp.Tensor, 233
espressopp.tools.analyse, 242
espressopp.tools.decomp, 245
espressopp.tools.DumpConfigurations, 246
espressopp.tools.espresso_old, 248
espressopp.tools.gromacs, 248
espressopp.tools.info, 243
espressopp.tools.io_extended, 250
espressopp.tools.lammps, 251
espressopp.tools.lattice, 244
espressopp.tools.pathintegral, 251
espressopp.tools.pdb, 252
espressopp.tools.povwrite, 252
espressopp.tools.prepareAdress, 254
espressopp.tools.prepareComplexMolecules, 252
espressopp.tools.psf, 254
espressopp.tools.replicate, 244
espressopp.tools.tabulated, 255
espressopp.tools.timers, 243
espressopp.tools.topology, 244
espressopp.tools.topology_helper, 255
espressopp.tools.units, 255
espressopp.tools.velocities, 245
espressopp.tools.vmd, 243
espressopp.tools.warmup, 245
espressopp.VerletList, 233
espressopp.VerletListAdress, 234
espressopp.VerletListTriple, 235
espressopp.Version, 236

INDEX

A

addForce() (*espressopp.integrator.LBInit.espressopp.integrator.LBInit method*), 91
angle_between_vectors() (*in module espressopp.external.transformations*), 62
Arcball (*class in espressopp.external.transformations*), 61
arcball_constrain_to_axis() (*in module espressopp.external.transformations*), 62
arcball_map_to_sphere() (*in module espressopp.external.transformations*), 62
arcball_nearest_axis() (*in module espressopp.external.transformations*), 62

C

call() (*in module espressopp.pmi*), 226
clip_matrix() (*in module espressopp.external.transformations*), 62
compose_matrix() (*in module espressopp.external.transformations*), 63
compute() (*espressopp.analysis.RadGyrXProfilePI.RadGyrXProfilePILocal method*), 52
computeEnergy() (*espressopp.interaction.AngularPotential.espressopp.interaction.AngularPotential method*), 110
computeForce() (*espressopp.interaction.AngularPotential.espressopp.interaction.AngularPotential method*), 110
computeKineticEnergy() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
computeMomentumDrift() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
computePositionDrift() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
computeRingEnergy() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
computeRingEnergyRaw() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
concatenate_matrices() (*in module espressopp.external.transformations*), 63
connect() (*in module espressopp.tools.vmd*), 243
convertTable() (*in module espressopp.tools.gromacs*), 250
CoulombTruncated (*class in espressopp.interaction.CoulombTruncated*), 130
CoulombTruncatedUniqueCharge (*class in espressopp.interaction.CoulombTruncatedUniqueCharge*), 131
create() (*in module espressopp.pmi*), 226
createCubic() (*in module espressopp.tools.lattice*), 244
createDenVel() (*espressopp.integrator.LBInit.espressopp.integrator.LBInit method*), 91
createDiamond() (*in module espressopp.tools.lattice*), 244

D

decompose_matrix() (*in module espressopp.external.transformations*), 63
Default() (*espressopp.standard_system.Default.espressopp.standard_system method*), 236
DihedralHarmonic (*class in espressopp.interaction.DihedralHarmonic*), 133
DihedralHarmonicCos (*class in espressopp.interaction.DihedralHarmonicCos*), 134
DihedralHarmonicNCos (*class in espressopp.interaction.DihedralHarmonicNCos*), 135
displacement (*in module espressopp.integrator.MinimizeEnergy*), 95
down() (*espressopp.external.transformations.Arcball method*), 61
drag() (*espressopp.external.transformations.Arcball method*), 61

E

Error, 210
espressopp.analysis.AdressDensity

module, 46
espressopp.analysis.AdressDensity() (*in module espressopp.analysis.AdressDensity*), 46
espressopp.analysis.AdressDensity.addExclusions() (*in module espressopp.analysis.AdressDensity*), 46
espressopp.analysis.AdressDensity.compute() (*in module espressopp.analysis.AdressDensity*), 46
espressopp.analysis.AllParticlePos
module, 33
espressopp.analysis.AllParticlePos.gatherAllPositions() (*in module espressopp.analysis.AllParticlePos*), 33
espressopp.analysis.AnalysisBase
module, 39
espressopp.analysis.AnalysisBase.compute() (*in module espressopp.analysis.AnalysisBase*), 39
espressopp.analysis.AnalysisBase.getAverageValue() (*in module espressopp.analysis.AnalysisBase*), 39
espressopp.analysis.AnalysisBase.getNumberOfMeasurements() (*in module espressopp.analysis.AnalysisBase*), 40
espressopp.analysis.AnalysisBase.performMeasurement() (*in module espressopp.analysis.AnalysisBase*), 40
espressopp.analysis.AnalysisBase.reset() (*in module espressopp.analysis.AnalysisBase*), 40
espressopp.analysis.Autocorrelation
module, 40
espressopp.analysis.Autocorrelation() (*in module espressopp.analysis.Autocorrelation*), 40
espressopp.analysis.Autocorrelation.clear() (*in module espressopp.analysis.Autocorrelation*), 40
espressopp.analysis.Autocorrelation.compute() (*in module espressopp.analysis.Autocorrelation*), 40
espressopp.analysis.Autocorrelation.gather() (*in module espressopp.analysis.Autocorrelation*), 40
espressopp.analysis.CenterOfMass
module, 46
espressopp.analysis.CenterOfMass() (*in module espressopp.analysis.CenterOfMass*), 47
espressopp.analysis.CMVelocity
module, 40
espressopp.analysis.CMVelocity (*class in espressopp.analysis.CMVelocity*), 40
espressopp.analysis.CMVelocity.compute() (*in module espressopp.analysis.CMVelocity*), 40
espressopp.analysis.CMVelocity.reset() (*in module espressopp.analysis.CMVelocity*), 41
espressopp.analysis.ConfigsParticleDecomp
module, 41
espressopp.analysis.ConfigsParticleDecomp() (*in module espressopp.analysis.ConfigsParticleDecomp*), 41
espressopp.analysis.ConfigsParticleDecomp.clear() (*in module espressopp.analysis.ConfigsParticleDecomp*), 41
espressopp.analysis.ConfigsParticleDecomp.compute() (*in module espressopp.analysis.ConfigsParticleDecomp*), 41
espressopp.analysis.ConfigsParticleDecomp.gather() (*in module espressopp.analysis.ConfigsParticleDecomp*), 41
espressopp.analysis.ConfigsParticleDecomp.gatherFromFile() (*in module espressopp.analysis.ConfigsParticleDecomp*), 41
espressopp.analysis.Configurations
module, 41
espressopp.analysis.Configurations() (*in module espressopp.analysis.Configurations*), 42
espressopp.analysis.Configurations.back() (*in module espressopp.analysis.Configurations*), 42
espressopp.analysis.Configurations.clear() (*in module espressopp.analysis.Configurations*), 42
espressopp.analysis.Configurations.gather() (*in module espressopp.analysis.Configurations*), 42
espressopp.analysis.ConfigurationsExt
module, 42
espressopp.analysis.ConfigurationsExt() (*in module espressopp.analysis.ConfigurationsExt*), 43
espressopp.analysis.ConfigurationsExt.back() (*in module espressopp.analysis.ConfigurationsExt*), 43
espressopp.analysis.ConfigurationsExt.clear() (*in module espressopp.analysis.ConfigurationsExt*), 43
espressopp.analysis.ConfigurationsExt.gather() (*in module espressopp.analysis.ConfigurationsExt*), 43
espressopp.analysis.Energy
module, 43
espressopp.analysis.EnergyKin() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.EnergyKin.compute() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.EnergyPot() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.EnergyPot.compute() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.EnergyTot() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.EnergyTot.compute() (*in module espressopp.analysis.Energy*), 44
espressopp.analysis.IntraChainDistSq
module, 44
espressopp.analysis.IntraChainDistSq() (*in module*

espressopp.analysis.IntraChainDistSq), 44
espressopp.analysis.IntraChainDistSq.compute() (*in module espressopp.analysis.IntraChainDistSq*), 44
espressopp.analysis.LBOutput
module, 33
espressopp.analysis.LBOutputScreen
module, 34
espressopp.analysis.LBOutputScreen (*class in espressopp.analysis.LBOutputScreen*), 34
espressopp.analysis.LBOutputVzInTime
module, 34
espressopp.analysis.LBOutputVzInTime (*class in espressopp.analysis.LBOutputVzInTime*), 34
espressopp.analysis.LBOutputVzOfX
module, 35
espressopp.analysis.LBOutputVzOfX (*class in espressopp.analysis.LBOutputVzOfX*), 35
espressopp.analysis.MaxPID
module, 47
espressopp.analysis.MaxPID() (*in module espressopp.analysis.MaxPID*), 47
espressopp.analysis.MeanSquareDispl
module, 44
espressopp.analysis.MeanSquareDispl() (*in module espressopp.analysis.MeanSquareDispl*), 45
espressopp.analysis.MeanSquareDispl.computeG2() (*in module espressopp.analysis.MeanSquareDispl*), 45
espressopp.analysis.MeanSquareDispl.computeG3() (*in module espressopp.analysis.MeanSquareDispl*), 45
espressopp.analysis.MeanSquareDispl.strange() (*in module espressopp.analysis.MeanSquareDispl*), 45
espressopp.analysis.MeanSquareInternalDist
module, 45
espressopp.analysis.MeanSquareInternalDist() (*in module espressopp.analysis.MeanSquareInternalDist*), 45
espressopp.analysis.NeighborFluctuation
module, 47
espressopp.analysis.NeighborFluctuation() (*in module espressopp.analysis.NeighborFluctuation*), 47
espressopp.analysis.NPart
module, 47
espressopp.analysis.NPart() (*in module espressopp.analysis.NPart*), 47
espressopp.analysis.NPartSubregion
module, 45
espressopp.analysis.NPartSubregion() (*in module espressopp.analysis.NPartSubregion*), 45
espressopp.analysis.Observable
module, 51
espressopp.analysis.Observable.compute() (*in module espressopp.analysis.Observable*), 51
espressopp.analysis.OrderParameter
module, 35
espressopp.analysis.OrderParameter() (*in module espressopp.analysis.OrderParameter*), 35
espressopp.analysis.ParticleRadiusDistribution
module, 35
espressopp.analysis.ParticleRadiusDistribution() (*in module espressopp.analysis.ParticleRadiusDistribution*), 35
espressopp.analysis.PotentialEnergy
module, 47
espressopp.analysis.PotentialEnergy() (*in module espressopp.analysis.PotentialEnergy*), 47
espressopp.analysis.Pressure
module, 47
espressopp.analysis.Pressure() (*in module espressopp.analysis.Pressure*), 48
espressopp.analysis.PressureTensor
module, 35
espressopp.analysis.PressureTensor() (*in module espressopp.analysis.PressureTensor*), 36
espressopp.analysis.PressureTensorLayer
module, 36
espressopp.analysis.PressureTensorLayer() (*in module espressopp.analysis.PressureTensorLayer*), 37
espressopp.analysis.PressureTensorMultiLayer
module, 37
espressopp.analysis.PressureTensorMultiLayer() (*in module espressopp.analysis.PressureTensorMultiLayer*), 38
espressopp.analysis.RadGyrXProfilePI
module, 51
espressopp.analysis.RadGyrXProfilePI() (*in module espressopp.analysis.RadGyrXProfilePI*), 52
espressopp.analysis.RadialDistrF
module, 48
espressopp.analysis.RadialDistrF() (*in module espressopp.analysis.RadialDistrF*), 48
espressopp.analysis.RadialDistrF.compute() (*in module espressopp.analysis.RadialDistrF*), 48
espressopp.analysis.RDFatomistic
module, 48
espressopp.analysis.RDFatomistic() (*in module espressopp.analysis.RDFatomistic*), 49
espressopp.analysis.RDFatomistic.compute() (*in module espressopp.analysis.RDFatomistic*), 49
espressopp.analysis.RDFatomistic.computePathIntegral() (*in module espressopp.analysis.RDFatomistic*), 50

espressopp.analysis.StaticStructF
module, 50
espressopp.analysis.StaticStructF() (*in module espressopp.analysis.StaticStructF*), 50
espressopp.analysis.StaticStructF.compute() (*in module espressopp.analysis.StaticStructF*), 50
espressopp.analysis.StaticStructF.computeSingleChain() (*in module espressopp.analysis.StaticStructF*), 50
espressopp.analysis.SubregionTracking
module, 52
espressopp.analysis.SubregionTracking() (*in module espressopp.analysis.SubregionTracking*), 53
espressopp.analysis.SystemMonitor
module, 53
espressopp.analysis.SystemMonitor() (*in module espressopp.analysis.SystemMonitor*), 53
espressopp.analysis.SystemMonitor.add_observable() (*in module espressopp.analysis.SystemMonitor*), 53
espressopp.analysis.SystemMonitor.info() (*in module espressopp.analysis.SystemMonitor*), 53
espressopp.analysis.SystemMonitorOutputCSV() (*in module espressopp.analysis.SystemMonitor*), 54
espressopp.analysis.Temperature
module, 39
espressopp.analysis.Temperature() (*in module espressopp.analysis.Temperature*), 39
espressopp.analysis.Test
module, 39
espressopp.analysis.Test() (*in module espressopp.analysis.Test*), 39
espressopp.analysis.Velocities
module, 54
espressopp.analysis.Velocities() (*in module espressopp.analysis.Velocities*), 54
espressopp.analysis.Velocities.clear() (*in module espressopp.analysis.Velocities*), 54
espressopp.analysis.Velocities.gather() (*in module espressopp.analysis.Velocities*), 54
espressopp.analysis.VelocityAutocorrelation
module, 54
espressopp.analysis.VelocityAutocorrelation() (*in module espressopp.analysis.VelocityAutocorrelation*), 54
espressopp.analysis.Viscosity
module, 54
espressopp.analysis.Viscosity() (*in module espressopp.analysis.Viscosity*), 55
espressopp.analysis.Viscosity.compute() (*in module espressopp.analysis.Viscosity*), 55
espressopp.analysis.Viscosity.gather() (*in module espressopp.analysis.Viscosity*), 55
espressopp.analysis.XDensity
module, 50
espressopp.analysis.XDensity() (*in module espressopp.analysis.XDensity*), 51
espressopp.analysis.XDensity.compute() (*in module espressopp.analysis.XDensity*), 51
espressopp.analysis.XPressure
module, 51
espressopp.analysis.XPressure() (*in module espressopp.analysis.XPressure*), 51
espressopp.analysis.XPressure.compute() (*in module espressopp.analysis.XPressure*), 51
espressopp.analysis.XTemperature
module, 51
espressopp.analysis.XTemperature() (*in module espressopp.analysis.XTemperature*), 51
espressopp.analysis.XTemperature.compute() (*in module espressopp.analysis.XTemperature*), 51
espressopp.bc.BC
module, 55
espressopp.bc.BC (*class in espressopp.bc.BC*), 55
espressopp.bc.OrthorhombicBC
module, 56
espressopp.bc.SlabBC
module, 56
espressopp.check.System
module, 57
espressopp.Error() (*in module espressopp.Exceptions*), 210
espressopp.esutil.Collectives
module, 57
espressopp.esutil.GammaVariate
module, 57
espressopp.esutil.GammaVariate() (*in module espressopp.esutil.GammaVariate*), 57
espressopp.esutil.Grid
module, 57
espressopp.esutil.NormalVariate
module, 58
espressopp.esutil.NormalVariate() (*in module espressopp.esutil.NormalVariate*), 58
espressopp.esutil.RNG
module, 58
espressopp.esutil.UniformOnSphere
module, 58
espressopp.Exceptions
module, 210
espressopp.external.transformations
module, 58
espressopp.FixedLocalTupleList
module, 211
espressopp.FixedLocalTupleList() (*in module espressopp.FixedLocalTupleList*), 211

espressopp.FixedLocalTupleList.addTuple() (*in module espressopp.FixedLocalTupleList*), 211
espressopp.FixedLocalTupleList.getTuples() (*in module espressopp.FixedLocalTupleList*), 211
espressopp.FixedLocalTupleList.size() (*in module espressopp.FixedLocalTupleList*), 211
espressopp.FixedPairDistList
module, 211
espressopp.FixedPairDistList() (*in module espressopp.FixedPairDistList*), 211
espressopp.FixedPairDistList.add() (*in module espressopp.FixedPairDistList*), 211
espressopp.FixedPairDistList.addPairs() (*in module espressopp.FixedPairDistList*), 211
espressopp.FixedPairDistList.getDist() (*in module espressopp.FixedPairDistList*), 211
espressopp.FixedPairDistList.getPairs() (*in module espressopp.FixedPairDistList*), 212
espressopp.FixedPairDistList.getPairsDist() (*in module espressopp.FixedPairDistList*), 212
espressopp.FixedPairDistList.size() (*in module espressopp.FixedPairDistList*), 212
espressopp.FixedPairList
module, 212
espressopp.FixedPairList() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.add() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.addBonds() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.getBonds() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.getLongtimeMaxBond() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.remove() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.resetLongtimeMaxBond() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.size() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairList.totalSize() (*in module espressopp.FixedPairList*), 212
espressopp.FixedPairListAdress
module, 213
espressopp.FixedPairListAdress() (*in module espressopp.FixedPairListAdress*), 213
espressopp.FixedPairListAdress.add() (*in module espressopp.FixedPairListAdress*), 213
espressopp.FixedPairListAdress.addBonds() (*in module espressopp.FixedPairListAdress*), 213
espressopp.FixedPairListAdress.getBonds() (*in module espressopp.FixedPairListAdress*), 213
espressopp.FixedPairListAdress.remove() (*in module espressopp.FixedPairListAdress*), 213
espressopp.FixedQuadrupleAngleList
module, 213
espressopp.FixedQuadrupleAngleList() (*in module espressopp.FixedQuadrupleAngleList*), 213
espressopp.FixedQuadrupleAngleList.add() (*in module espressopp.FixedQuadrupleAngleList*), 213
espressopp.FixedQuadrupleAngleList.addQuadruples() (*in module espressopp.FixedQuadrupleAngleList*), 214
espressopp.FixedQuadrupleAngleList.getAngle() (*in module espressopp.FixedQuadrupleAngleList*), 214
espressopp.FixedQuadrupleAngleList.getQuadruples() (*in module espressopp.FixedQuadrupleAngleList*), 214
espressopp.FixedQuadrupleAngleList.getQuadruplesAngles() (*in module espressopp.FixedQuadrupleAngleList*), 214
espressopp.FixedQuadrupleAngleList.size() (*in module espressopp.FixedQuadrupleAngleList*), 214
espressopp.FixedQuadrupleList
module, 214
espressopp.FixedQuadrupleList() (*in module espressopp.FixedQuadrupleList*), 214
espressopp.FixedQuadrupleList.add() (*in module espressopp.FixedQuadrupleList*), 214
espressopp.FixedQuadrupleList.addQuadruples() (*in module espressopp.FixedQuadrupleList*), 215
espressopp.FixedQuadrupleList.getQuadruples() (*in module espressopp.FixedQuadrupleList*), 215
espressopp.FixedQuadrupleList.remove() (*in module espressopp.FixedQuadrupleList*), 215
espressopp.FixedQuadrupleList.size() (*in module espressopp.FixedQuadrupleList*), 215
espressopp.FixedQuadrupleListAdress
module, 215
espressopp.FixedQuadrupleListAdress() (*in module espressopp.FixedQuadrupleListAdress*), 215
espressopp.FixedQuadrupleListAdress.add() (*in module espressopp.FixedQuadrupleListAdress*), 215
espressopp.FixedQuadrupleListAdress.addQuadruples() (*in module espressopp.FixedQuadrupleListAdress*), 215
espressopp.FixedQuadrupleListAdress.getQuadruples() (*in module espressopp.FixedQuadrupleListAdress*), 215
espressopp.FixedQuadrupleListAdress.size() (*in module espressopp.FixedQuadrupleListAdress*), 215
espressopp.FixedSingleList
module, 215
espressopp.FixedSingleList() (*in module espressopp.FixedSingleList*), 216

espressopp.FixedSingleList.add() (*in module espressopp.FixedSingleList*), 216
espressopp.FixedSingleList.addSingles() (*in module espressopp.FixedSingleList*), 216
espressopp.FixedSingleList.getSingles() (*in module espressopp.FixedSingleList*), 216
espressopp.FixedSingleList.size() (*in module espressopp.FixedSingleList*), 216
espressopp.FixedTripleAngleList
module, 216
espressopp.FixedTripleAngleList() (*in module espressopp.FixedTripleAngleList*), 216
espressopp.FixedTripleAngleList.add() (*in module espressopp.FixedTripleAngleList*), 216
espressopp.FixedTripleAngleList.addTriples() (*in module espressopp.FixedTripleAngleList*), 216
espressopp.FixedTripleAngleList.getAngle() (*in module espressopp.FixedTripleAngleList*), 216
espressopp.FixedTripleAngleList.getTriples() (*in module espressopp.FixedTripleAngleList*), 217
espressopp.FixedTripleAngleList.getTriplesAngles() (*in module espressopp.FixedTripleAngleList*), 217
espressopp.FixedTripleAngleList.size() (*in module espressopp.FixedTripleAngleList*), 217
espressopp.FixedTripleList
module, 217
espressopp.FixedTripleList() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleList.add() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleList.addTriples() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleList.getTriples() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleList.remove() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleList.size() (*in module espressopp.FixedTripleList*), 217
espressopp.FixedTripleListAdress
module, 217
espressopp.FixedTripleListAdress() (*in module espressopp.FixedTripleListAdress*), 217
espressopp.FixedTripleListAdress.add() (*in module espressopp.FixedTripleListAdress*), 217
espressopp.FixedTripleListAdress.addTriples() (*in module espressopp.FixedTripleListAdress*), 218
espressopp.FixedTripleListAdress.remove() (*in module espressopp.FixedTripleListAdress*), 218
espressopp.FixedTupleList
module, 218
espressopp.FixedTupleList() (*in module espressopp.FixedTupleList*), 218
espressopp.FixedTupleList.size() (*in module espressopp.FixedTupleList*), 218
espressopp.FixedTupleListAdress
module, 218
espressopp.FixedTupleListAdress() (*in module espressopp.FixedTupleListAdress*), 218
espressopp.FixedTupleListAdress.addTuples() (*in module espressopp.FixedTupleListAdress*), 218
espressopp.Int3D
module, 218
espressopp.Int3D() (*in module espressopp.Int3D*), 219
espressopp.Int3D.x() (*in module espressopp.Int3D*), 219
espressopp.Int3D.y() (*in module espressopp.Int3D*), 219
espressopp.Int3D.z() (*in module espressopp.Int3D*), 219
espressopp.integrator.Adress
module, 74
espressopp.integrator.Adress (*class in espressopp.integrator.Adress*), 75
espressopp.integrator.AssociationReaction
module, 75
espressopp.integrator.BerendsenBarostat
module, 75
espressopp.integrator.BerendsenBarostat() (*in module espressopp.integrator.BerendsenBarostat*), 76
espressopp.integrator.BerendsenBarostatAnisotropic
module, 76
espressopp.integrator.BerendsenBarostatAnisotropic() (*in module espressopp.integrator.BerendsenBarostatAnisotropic*), 78
espressopp.integrator.BerendsenThermostat
module, 78
espressopp.integrator.BerendsenThermostat() (*in module espressopp.integrator.BerendsenThermostat*), 79
espressopp.integrator.CapForce
module, 79
espressopp.integrator.CapForce() (*in module espressopp.integrator.CapForce*), 79
espressopp.integrator.DPDThermostat
module, 80
espressopp.integrator.DPDThermostat() (*in module espressopp.integrator.DPDThermostat*), 80
espressopp.integrator.EmptyExtension
module, 80
espressopp.integrator.EmptyExtension() (*in module espressopp.integrator.EmptyExtension*), 80
espressopp.integrator.ExtAnalyze
module, 80
espressopp.integrator.ExtAnalyze() (*in module espressopp.integrator.ExtAnalyze*), 80
espressopp.integrator.Extension
module, 80
espressopp.integrator.Extension.connect() (*in module espressopp.integrator.Extension*), 80

espressopp.integrator.Extension.disconnect() (*in module espressopp.integrator.Extension*), 80
espressopp.integrator.ExtForce
module, 80
espressopp.integrator.ExtForce() (*in module espressopp.integrator.ExtForce*), 81
espressopp.integrator.FixPositions
module, 81
espressopp.integrator.FixPositions() (*in module espressopp.integrator.FixPositions*), 81
espressopp.integrator.FreeEnergyCompensation
module, 81
espressopp.integrator.FreeEnergyCompensation() (*in module espressopp.integrator.FreeEnergyCompensation*), 81
espressopp.integrator.FreeEnergyCompensation.addForces() (*in module espressopp.integrator.FreeEnergyCompensation*), 81
espressopp.integrator.FreeEnergyCompensation.computeCompensationEnergy() (*in module espressopp.integrator.FreeEnergyCompensation*), 82
espressopp.integrator.GeneralizedLangevinThermostat
module, 82
espressopp.integrator.GeneralizedLangevinThermostat() (*in module espressopp.integrator.GeneralizedLangevinThermostat*), 82
espressopp.integrator.GeneralizedLangevinThermostat.addCoeffs() (*in module espressopp.integrator.GeneralizedLangevinThermostat*), 82
espressopp.integrator.Isokinetic
module, 82
espressopp.integrator.Isokinetic() (*in module espressopp.integrator.Isokinetic*), 82
espressopp.integrator.LangevinBarostat
module, 82
espressopp.integrator.LangevinBarostat() (*in module espressopp.integrator.LangevinBarostat*), 84
espressopp.integrator.LangevinThermostat
module, 84
espressopp.integrator.LangevinThermostat() (*in module espressopp.integrator.LangevinThermostat*), 85
espressopp.integrator.LangevinThermostat.addExclusions() (*in module espressopp.integrator.LangevinThermostat*), 85
espressopp.integrator.LangevinThermostat1D
module, 85
espressopp.integrator.LangevinThermostat1D() (*in module espressopp.integrator.LangevinThermostat1D*), 85
espressopp.integrator.LangevinThermostatHybrid
module, 85
espressopp.integrator.LangevinThermostatOnGroup
module, 86
espressopp.integrator.LangevinThermostatOnGroup() (*in module espressopp.integrator.LangevinThermostatOnGroup*), 86
espressopp.integrator.LangevinThermostatOnRadius
module, 86
espressopp.integrator.LangevinThermostatOnRadius() (*in module espressopp.integrator.LangevinThermostatOnRadius*), 86
espressopp.integrator.LangevinThermostatOnRadius.addExclusions() (*in module espressopp.integrator.LangevinThermostatOnRadius*), 86
espressopp.integrator.LatticeBoltzmann
module, 87
espressopp.integrator.LatticeBoltzmann (*class in espressopp.integrator.LatticeBoltzmann*), 87
espressopp.integrator.LBInit
module, 91
espressopp.integrator.LBInit (*class in espressopp.integrator.LBInit*), 91
espressopp.integrator.LBInitConstForce
module, 92
espressopp.integrator.LBInitPeriodicForce
module, 93
espressopp.integrator.LBInitPopUniform
module, 92
espressopp.integrator.LBInitPopWave
module, 92
espressopp.integrator.MDIntegrator
module, 93
espressopp.integrator.MDIntegrator.addExtension() (*in module espressopp.integrator.MDIntegrator*), 93
espressopp.integrator.MDIntegrator.getExtension() (*in module espressopp.integrator.MDIntegrator*), 93
espressopp.integrator.MDIntegrator.getNumberOfExtensions() (*in module espressopp.integrator.MDIntegrator*), 93
espressopp.integrator.MDIntegrator.run() (*in module espressopp.integrator.MDIntegrator*), 93
espressopp.integrator.MinimizeEnergy
module, 93
espressopp.integrator.MinimizeEnergy() (*in module espressopp.integrator.MinimizeEnergy*), 94
espressopp.integrator.MinimizeEnergy.run() (*in mod-*

ule espressopp.integrator.MinimizeEnergy), 94
espressopp.integrator.OnTheFlyFEC module, 95
espressopp.integrator.OnTheFlyFEC() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.getBins() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.getGap() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.getSteps() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.makeArrays() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.resetCounter() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.OnTheFlyFEC.writeFEC() (in module espressopp.integrator.OnTheFlyFEC), 95
espressopp.integrator.PIAdressIntegrator module, 95
espressopp.integrator.PIAdressIntegrator() (in module espressopp.integrator.PIAdressIntegrator), 96
espressopp.integrator.PIAdressIntegrator.computeKineticEnergy() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.computeMomentumDrift() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.computePositionDrift() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.computeRingEnergy() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.computeRingEnergyRaw() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.getCentroidThermostat() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.getClmassmultiplier() (in module espressopp.integrator.PIAdressIntegrator), 100
espressopp.integrator.PIAdressIntegrator.getCMDparameter() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.getConstKinMass() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.getGamma() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.getGamma() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.getKTI() (in module espressopp.integrator.PIAdressIntegrator), 100
espressopp.integrator.PIAdressIntegrator.getmStep() (in module espressopp.integrator.PIAdressIntegrator), 97
espressopp.integrator.PIAdressIntegrator.getNtrotter() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.getPILE() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.getPILElambda() (in module espressopp.integrator.PIAdressIntegrator), 100
espressopp.integrator.PIAdressIntegrator.getRealKinMass() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.getSpeedup() (in module espressopp.integrator.PIAdressIntegrator), 100
espressopp.integrator.PIAdressIntegrator.getsStep() (in module espressopp.integrator.PIAdressIntegrator), 97
espressopp.integrator.PIAdressIntegrator.getTemperature() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.getTimeStep() (in module espressopp.integrator.PIAdressIntegrator), 97
espressopp.integrator.PIAdressIntegrator.getVerletList() (in module espressopp.integrator.PIAdressIntegrator), 97
espressopp.integrator.PIAdressIntegrator.getVerletlistBuilds() (in module espressopp.integrator.PIAdressIntegrator), 101
espressopp.integrator.PIAdressIntegrator.setCentroidThermostat() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.setClmassmultiplier() (in module espressopp.integrator.PIAdressIntegrator), 100
espressopp.integrator.PIAdressIntegrator.setCMDparameter() (in module espressopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.setConstKinMass() (in module espressopp.integrator.PIAdressIntegrator), 98
espressopp.integrator.PIAdressIntegrator.setGamma() (in module espres-

sopp.integrator.PIAdressIntegrator), 99
espressopp.integrator.PIAdressIntegrator.setKTI() (*in module espressopp.integrator.PIAdressIntegrator*), 100
espressopp.integrator.PIAdressIntegrator.setmStep() (*in module espressopp.integrator.PIAdressIntegrator*), 97
espressopp.integrator.PIAdressIntegrator.setNtrotter() (*in module espressopp.integrator.PIAdressIntegrator*), 98
espressopp.integrator.PIAdressIntegrator.setPILE() (*in module espressopp.integrator.PIAdressIntegrator*), 99
espressopp.integrator.PIAdressIntegrator.setPILElambda() (*in module espressopp.integrator.PIAdressIntegrator*), 100
espressopp.integrator.PIAdressIntegrator.setRealKinMass() (*in module espressopp.integrator.PIAdressIntegrator*), 98
espressopp.integrator.PIAdressIntegrator.setSpeedup() (*in module espressopp.integrator.PIAdressIntegrator*), 100
espressopp.integrator.PIAdressIntegrator.setsStep() (*in module espressopp.integrator.PIAdressIntegrator*), 97
espressopp.integrator.PIAdressIntegrator.setTemperature() (*in module espressopp.integrator.PIAdressIntegrator*), 98
espressopp.integrator.PIAdressIntegrator.setTimeStep() (*in module espressopp.integrator.PIAdressIntegrator*), 97
espressopp.integrator.PIAdressIntegrator.setVerletList() (*in module espressopp.integrator.PIAdressIntegrator*), 97
espressopp.integrator.Rattle
module, 105
espressopp.integrator.Settle
module, 106
espressopp.integrator.Settle() (*in module espressopp.integrator.Settle*), 107
espressopp.integrator.Settle.addMolecules() (*in module espressopp.integrator.Settle*), 107
espressopp.integrator.StochasticVelocityRescaling
module, 107
espressopp.integrator.StochasticVelocityRescaling() (*in module espressopp.integrator.StochasticVelocityRescaling*), 107
espressopp.integrator.TDforce
module, 107
espressopp.integrator.TDforce() (*in module espressopp.integrator.TDforce*), 107
espressopp.integrator.TDforce.addForce() (*in module espressopp.integrator.TDforce*), 108
espressopp.integrator.TDforce.computeTDEnergy() (*in module espressopp.integrator.TDforce*), 108
espressopp.integrator.VelocityVerlet
module, 108
espressopp.integrator.VelocityVerlet() (*in module espressopp.integrator.VelocityVerlet*), 108
espressopp.integrator.VelocityVerletOnGroup
module, 109
espressopp.integrator.VelocityVerletOnGroup() (*in module espressopp.integrator.VelocityVerletOnGroup*), 109
espressopp.integrator.VelocityVerletOnRadius
module, 109
espressopp.integrator.VelocityVerletOnRadius() (*in module espressopp.integrator.VelocityVerletOnRadius*), 109
espressopp.integrator.VelocityVerletRESPA
module, 108
espressopp.integrator.VelocityVerletRESPA (*class in espressopp.integrator.VelocityVerletRESPA*), 109
espressopp.integrator.VelocityVerletRESPA.getmultistep() (*in module espressopp.integrator.VelocityVerletRESPA*), 109
espressopp.integrator.VelocityVerletRESPA.setmultistep() (*in module espressopp.integrator.VelocityVerletRESPA*), 109
espressopp.interaction.AngularCosineSquared
module, 110
espressopp.interaction.AngularCosineSquared (*class in espressopp.interaction.AngularCosineSquared*), 110
espressopp.interaction.AngularHarmonic
module, 112
espressopp.interaction.AngularHarmonic (*class in espressopp.interaction.AngularHarmonic*), 112
espressopp.interaction.AngularPotential
module, 110
espressopp.interaction.AngularPotential (*class in espressopp.interaction.AngularPotential*), 110
espressopp.interaction.CellListCoulombKSpaceEwald() (*in module espressopp.interaction.CoulombKSpaceEwald*), 125
espressopp.interaction.CellListCoulombKSpaceEwald.getFixedPairLis (*in module espressopp.interaction.CoulombKSpaceEwald*), 125
espressopp.interaction.CellListCoulombKSpaceEwald.getPotential() (*in module espressopp.interaction.CoulombKSpaceEwald*),

126
espressopp.interaction.CellListCoulombKSpaceP3M() (*in module espressopp.interaction.CoulombKSpaceP3M*), 127
espressopp.interaction.CellListCoulombKSpaceP3M.getPotential() (*in module espressopp.interaction.CoulombKSpaceP3M*), 127
espressopp.interaction.CellListCoulombTruncatedUniqueCharge() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 131
espressopp.interaction.CellListCoulombTruncatedUniqueCharge.setPotential() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 131
espressopp.interaction.CellListLennardJones() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.CellListLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.CellListLennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*), 160
espressopp.interaction.CellListLennardJonesAutoBonds.setPotential() (*in module espressopp.interaction.LennardJonesAutoBonds*), 160
espressopp.interaction.CellListLennardJonesCapped() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.CellListLennardJonesCapped.getPotential() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.CellListLennardJonesCapped.setPotential() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.CellListLennardJonesEnergyCapped() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.CellListLennardJonesEnergyCapped.getPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.CellListLennardJonesEnergyCapped.setPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.CellListLennardJonesExpand() (*in module espressopp.interaction.LennardJonesExpand*), 167
espressopp.interaction.CellListLennardJonesExpand.setPotential() (*in module espressopp.interaction.LennardJonesExpand*), 167
espressopp.interaction.CellListLennardJonesGeneric() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.CellListLennardJonesGeneric.setPotential() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.CellListLennardJonesGromacs() (*in module espressopp.interaction.LennardJonesGromacs*), 171
espressopp.interaction.CellListLennardJonesGromacs.setPotential() (*in module espressopp.interaction.LennardJonesGromacs*), 171
espressopp.interaction.CellListLJcos() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.CellListLJcos.setPotential() (*in module espressopp.interaction.LJcos*), 176
espressopp.interaction.CellListMorse() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.CellListMorse.setPotential() (*in module espressopp.interaction.Morse*), 123
espressopp.interaction.CellListReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 197
espressopp.interaction.CellListReactionFieldGeneralized.setPotential() (*in module espressopp.interaction.ReactionFieldGeneralized*), 197
espressopp.interaction.CellListSoftCosine() (*in module espressopp.interaction.SoftCosine*), 124
espressopp.interaction.CellListSoftCosine.setPotential() (*in module espressopp.interaction.SoftCosine*), 124
espressopp.interaction.CellListStillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.CellListStillingerWeberPairTerm.setPotential() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139

espressopp.interaction.CellListStillingerWeberPairTermCapped() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 142
espressopp.interaction.CellListStillingerWeberPairTermCapped.setPotential() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 142
espressopp.interaction.CellListTabulated() (*in module espressopp.interaction.Tabulated*), 180
espressopp.interaction.CellListTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 180
espressopp.interaction.CellListTersoffPairTerm() (*in module espressopp.interaction.TersoffPairTerm*), 144
espressopp.interaction.CellListTersoffPairTerm.setPotential() (*in module espressopp.interaction.TersoffPairTerm*), 144
espressopp.interaction.CellListZero() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.CellListZero.setPotential() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.ConstrainCOM
module, 131
espressopp.interaction.ConstrainCOM() (*in module espressopp.interaction.ConstrainCOM*), 131
espressopp.interaction.ConstrainRG
module, 132
espressopp.interaction.ConstrainRG() (*in module espressopp.interaction.ConstrainRG*), 132
espressopp.interaction.Cosine
module, 113
espressopp.interaction.Cosine() (*in module espressopp.interaction.Cosine*), 113
espressopp.interaction.CoulombKSpaceEwald
module, 124
espressopp.interaction.CoulombKSpaceEwald() (*in module espressopp.interaction.CoulombKSpaceEwald*), 125
espressopp.interaction.CoulombKSpaceP3M
module, 126
espressopp.interaction.CoulombKSpaceP3M() (*in module espressopp.interaction.CoulombKSpaceP3M*), 127
espressopp.interaction.CoulombRSpace
module, 127
espressopp.interaction.CoulombRSpace() (*in module espressopp.interaction.CoulombRSpace*), 128
espressopp.interaction.CoulombTruncated
module, 129
espressopp.interaction.CoulombTruncatedUniqueCharge
module, 130
espressopp.interaction.CoulombTruncatedUniqueCharge() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 130
espressopp.interaction.DihedralHarmonic
module, 133
espressopp.interaction.DihedralHarmonic() (*in module espressopp.interaction.DihedralHarmonic*), 133
espressopp.interaction.DihedralHarmonicCos
module, 134
espressopp.interaction.DihedralHarmonicCos() (*in module espressopp.interaction.DihedralHarmonicCos*), 134
espressopp.interaction.DihedralHarmonicNCos
module, 134
espressopp.interaction.DihedralHarmonicNCos() (*in module espressopp.interaction.DihedralHarmonicNCos*), 134
espressopp.interaction.DihedralPotential
module, 135
espressopp.interaction.DihedralPotential.computeEnergy() (*in module espressopp.interaction.DihedralPotential*), 136
espressopp.interaction.DihedralPotential.computeForce() (*in module espressopp.interaction.DihedralPotential*), 136
espressopp.interaction.DihedralRB
module, 136
espressopp.interaction.DihedralRB() (*in module espressopp.interaction.DihedralRB*), 136
espressopp.interaction.FENE
module, 114
espressopp.interaction.FENE (*class in espressopp.interaction.FENE*), 114
espressopp.interaction.FENECapped
module, 115
espressopp.interaction.FENECapped (*class in espressopp.interaction.FENECapped*), 115
espressopp.interaction.FixedLocalTupleListConstrainCOM() (*in module espressopp.interaction.ConstrainCOM*), 132
espressopp.interaction.FixedLocalTupleListConstrainCOM.getPotenti
(*in module espressopp.interaction.ConstrainCOM*), 132
espressopp.interaction.FixedLocalTupleListConstrainCOM.setCom() (*in module espressopp.interaction.ConstrainCOM*), 132
espressopp.interaction.FixedLocalTupleListConstrainRG() (*in module espressopp.interaction.ConstrainRG*), 132

espressopp.interaction.FixedLocalTupleListConstrainRG.getPotential() (*in module espressopp.interaction.ConstrainRG*), 132
espressopp.interaction.FixedLocalTupleListConstrainRG.setPotential() (*in module espressopp.interaction.ConstrainRG*), 132
espressopp.interaction.FixedPairListCoulombTruncatedUniqueCharge (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 131
espressopp.interaction.FixedPairListCoulombTruncatedUniqueCharge.setPotential() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 131
espressopp.interaction.FixedPairListFENE (*class in espressopp.interaction.FENE*), 114
espressopp.interaction.FixedPairListFENECapped (*class in espressopp.interaction.FENECapped*), 115
espressopp.interaction.FixedPairListHarmonic() (*in module espressopp.interaction.Harmonic*), 116
espressopp.interaction.FixedPairListHarmonic.getFixedPairList() (*in module espressopp.interaction.Harmonic*), 116
espressopp.interaction.FixedPairListHarmonic.setFixedPairList() (*in module espressopp.interaction.Harmonic*), 116
espressopp.interaction.FixedPairListHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListLennardJones() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.FixedPairListLennardJones.getFixedPairList() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.FixedPairListLennardJones.getPotential() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.FixedPairListLennardJones.setFixedPairList() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.FixedPairListLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.FixedPairListLennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*), 160
espressopp.interaction.FixedPairListLennardJonesAutoBonds.setPotential() (*in module espressopp.interaction.LennardJonesAutoBonds*), 160
espressopp.interaction.FixedPairListLennardJonesCapped() (*in module espressopp.interaction.LennardJonesCapped*), 163
espressopp.interaction.FixedPairListLennardJonesCapped.getPotential() (*in module espressopp.interaction.LennardJonesCapped*), 163
espressopp.interaction.FixedPairListLennardJonesCapped.setPotential() (*in module espressopp.interaction.LennardJonesCapped*), 163
espressopp.interaction.FixedPairListLennardJonesEnergyCapped() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.FixedPairListLennardJonesEnergyCapped.getPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 166
espressopp.interaction.FixedPairListLennardJonesEnergyCapped.setPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 166
espressopp.interaction.FixedPairListLennardJonesExpand() (*in module espressopp.interaction.LennardJonesExpand*), 167
espressopp.interaction.FixedPairListLennardJonesExpand.setPotential() (*in module espressopp.interaction.LennardJonesExpand*), 167
espressopp.interaction.FixedPairListLennardJonesGeneric() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.FixedPairListLennardJonesGeneric.getFixedPairList() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.FixedPairListLennardJonesGeneric.getPotential() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.FixedPairListLennardJonesGeneric.setFixedPairList() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.FixedPairListLennardJonesGeneric.setPotential() (*in module espressopp.interaction.LennardJonesGeneric*), 170
espressopp.interaction.FixedPairListLennardJonesGromacs() (*in module espressopp.interaction.LennardJonesGromacs*),

171
espressopp.interaction.FixedPairListLennardJonesGromacs.setPotential() (*in module espressopp.interaction.LennardJonesGromacs*), 171
espressopp.interaction.FixedPairListLJcos() (*in module espressopp.interaction.LJcos*), 176
espressopp.interaction.FixedPairListLJcos.getFixedPairList() (*in module espressopp.interaction.LJcos*), 176
espressopp.interaction.FixedPairListLJcos.setFixedPairList() (*in module espressopp.interaction.LJcos*), 176
espressopp.interaction.FixedPairListLJcos.setPotential() (*in module espressopp.interaction.LJcos*), 176
espressopp.interaction.FixedPairListMirrorLennardJones() (*in module espressopp.interaction.MirrorLennardJones*), 176
espressopp.interaction.FixedPairListMirrorLennardJones.getFixedPairList() (*in module espressopp.interaction.MirrorLennardJones*), 177
espressopp.interaction.FixedPairListMirrorLennardJones.getPotential() (*in module espressopp.interaction.MirrorLennardJones*), 177
espressopp.interaction.FixedPairListMirrorLennardJones.setFixedPairList() (*in module espressopp.interaction.MirrorLennardJones*), 177
espressopp.interaction.FixedPairListMirrorLennardJones.setPotential() (*in module espressopp.interaction.MirrorLennardJones*), 177
espressopp.interaction.FixedPairListMorse() (*in module espressopp.interaction.Morse*), 123
espressopp.interaction.FixedPairListMorse.setPotential() (*in module espressopp.interaction.Morse*), 123
espressopp.interaction.FixedPairListPIadressTabulated() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListPIadressTabulated.getFixedPairList() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListPIadressTabulated.getFixedTupleList() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListPIadressTabulated.getNTrotter() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListPIadressTabulated.getPotential() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListPIadressTabulated.getSpeedup() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListPIadressTabulated.setFixedPairList() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListPIadressTabulated.setFixedTupleList() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListPIadressTabulated.setNTrotter() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListPIadressTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListPIadressTabulated.setSpeedup() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.FixedPairListQuartic() (*in module espressopp.interaction.Quartic*), 193
espressopp.interaction.FixedPairListQuartic.getFixedPairList() (*in module espressopp.interaction.Quartic*), 194
espressopp.interaction.FixedPairListQuartic.setFixedPairList() (*in module espressopp.interaction.Quartic*), 194
espressopp.interaction.FixedPairListQuartic.setPotential() (*in module espressopp.interaction.Quartic*), 194
espressopp.interaction.FixedPairListSoftCosine() (*in module espressopp.interaction.SoftCosine*), 124
espressopp.interaction.FixedPairListSoftCosine.setPotential() (*in module espressopp.interaction.SoftCosine*), 124
espressopp.interaction.FixedPairListStillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.FixedPairListStillingerWeberPairTerm.setPote (*in module espressopp.interaction.StillingerWeberPairTerm*), 140
espressopp.interaction.FixedPairListStillingerWeberPairTermCapped((*in module espressopp.interaction.StillingerWeberPairTermCapped*), 142
espressopp.interaction.FixedPairListStillingerWeberPairTermCapped. () *module espressopp.interaction.StillingerWeberPairTermCapped*), 142
espressopp.interaction.FixedPairListTabulated() (*in module espressopp.interaction.Tabulated*), 180
espressopp.interaction.FixedPairListTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 180
espressopp.interaction.FixedPairListTersoffPairTerm() (*in module espressopp.interaction.TersoffPairTerm*), 145
espressopp.interaction.FixedPairListTersoffPairTerm.setPotential() (*in module espressopp.interaction.TersoffPairTerm*), 145

sopp.interaction.TersoffPairTerm), 145
espressopp.interaction.FixedPairListTypesHarmonic() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListTypesHarmonic.getFixedPairList() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListTypesHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListTypesHarmonic.setFixedPairList() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListTypesHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.FixedPairListTypesTabulated() (*in module espressopp.interaction.Tabulated*), 180
espressopp.interaction.FixedPairListTypesTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 181
espressopp.interaction.FixedPairListZero() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.FixedPairListZero.setPotential() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.FixedQuadrupleListDihedralHarmonic() (*in module espressopp.interaction.DihedralHarmonic*), 133
espressopp.interaction.FixedQuadrupleListDihedralHarmonic.getFixedQuadrupleList() (*in module espressopp.interaction.DihedralHarmonic*), 133
espressopp.interaction.FixedQuadrupleListDihedralHarmonic.setPotential() (*in module espressopp.interaction.DihedralHarmonic*), 133
espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos() (*in module espressopp.interaction.DihedralHarmonicCos*), 134
espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos.getFixedQuadrupleList() (*in module espressopp.interaction.DihedralHarmonicCos*), 134
espressopp.interaction.FixedQuadrupleListDihedralHarmonicCos.setPotential() (*in module espressopp.interaction.DihedralHarmonicCos*), 134
espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos() (*in module espressopp.interaction.DihedralHarmonicNCos*), 135
espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos.getFixedQuadrupleList() (*in module espressopp.interaction.DihedralHarmonicNCos*), 135
espressopp.interaction.FixedQuadrupleListDihedralHarmonicNCos.setPotential() (*in module espressopp.interaction.DihedralHarmonicNCos*), 135
espressopp.interaction.FixedQuadrupleListDihedralRB() (*in module espressopp.interaction.DihedralRB*), 136
espressopp.interaction.FixedQuadrupleListDihedralRB.getFixedQuadrupleList() (*in module espressopp.interaction.DihedralRB*), 136
espressopp.interaction.FixedQuadrupleListDihedralRB.setPotential() (*in module espressopp.interaction.DihedralRB*), 136
espressopp.interaction.FixedQuadrupleListOPLS() (*in module espressopp.interaction.OPLS*), 137
espressopp.interaction.FixedQuadrupleListOPLS.setPotential() (*in module espressopp.interaction.OPLS*), 137
espressopp.interaction.FixedQuadrupleListTabulatedDihedral() (*in module espressopp.interaction.TabulatedDihedral*), 190
espressopp.interaction.FixedQuadrupleListTabulatedDihedral.setPotential() (*in module espressopp.interaction.TabulatedDihedral*), 190
espressopp.interaction.FixedQuadrupleListTypesTabulatedDihedral() (*in module espressopp.interaction.TabulatedDihedral*), 190, 191
espressopp.interaction.FixedQuadrupleListTypesTabulatedDihedral.setPotential() (*in module espressopp.interaction.TabulatedDihedral*), 191
espressopp.interaction.FixedTripleListAngularCosineSquared (*class in espressopp.interaction.AngularCosineSquared*), 111
espressopp.interaction.FixedTripleListAngularHarmonic (*class in espressopp.interaction.AngularHarmonic*), 112
espressopp.interaction.FixedTripleListCosine() (*in module espressopp.interaction.Cosine*), 113
espressopp.interaction.FixedTripleListCosine.getFixedTripleList() (*in module espressopp.interaction.Cosine*), 113
espressopp.interaction.FixedTripleListCosine.setPotential() (*in module espressopp.interaction.Cosine*), 113
espressopp.interaction.FixedTripleListPIadressTabulatedAngular() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getFixedTripleList() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getFixedTripleList() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getN

(*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getPotential() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.getSwapInteraction() (*in module espressopp.interaction.TabulatedAngular*), 189
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setFixedTripleList() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setPotential() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListPIadressTabulatedAngular.setSwapInteraction() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.FixedTripleListPIadressTabulatedAngularInteractionTemplate() (*in module espressopp.interaction.TabulatedAngular*), 188
espressopp.interaction.FixedTripleListStillingerWeberTripleTerm() (*in module espressopp.interaction.StillingerWeberTripleTerm*), 143
espressopp.interaction.FixedTripleListStillingerWeberTripleTerm.getFixedTripleList() (*in module espressopp.interaction.StillingerWeberTripleTerm*), 143
espressopp.interaction.FixedTripleListStillingerWeberTripleTerm.setPotential() (*in module espressopp.interaction.StillingerWeberTripleTerm*), 143
espressopp.interaction.FixedTripleListTabulatedAngular() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.FixedTripleListTabulatedAngular.setPotential() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.FixedTripleListTersoffTripleTerm() (*in module espressopp.interaction.TersoffTripleTerm*), 146
espressopp.interaction.FixedTripleListTersoffTripleTerm.getFixedTripleList() (*in module espressopp.interaction.TersoffTripleTerm*), 146
espressopp.interaction.FixedTripleListTersoffTripleTerm.setPotential() (*in module espressopp.interaction.TersoffTripleTerm*), 146
espressopp.interaction.FixedTripleListTypesTabulatedAngular() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.FixedTripleListTypesTabulatedAngular.setPotential() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.GravityTruncated module, 146
espressopp.interaction.GravityTruncated() (*in module espressopp.interaction.GravityTruncated*), 146
espressopp.interaction.Harmonic module, 116
espressopp.interaction.Harmonic() (*in module espressopp.interaction.Harmonic*), 116
espressopp.interaction.HarmonicTrap module, 120
espressopp.interaction.HarmonicTrap() (*in module espressopp.interaction.HarmonicTrap*), 121
espressopp.interaction.Interaction module, 192
espressopp.interaction.Interaction.bondType() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.Interaction.computeEnergy() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.Interaction.computeEnergyAA() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.Interaction.computeEnergyCG() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.Interaction.computeEnergyDeriv() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.Interaction.computeVirial() (*in module espressopp.interaction.Interaction*), 192
espressopp.interaction.LennardJones module, 147
espressopp.interaction.LennardJones() (*in module espressopp.interaction.LennardJones*), 147
espressopp.interaction.LennardJones93Wall module, 191
espressopp.interaction.LennardJones93Wall() (*in module espressopp.interaction.LennardJones93Wall*), 191
espressopp.interaction.LennardJones93Wall.getParams() (*in module espressopp.interaction.LennardJones93Wall*), 191
espressopp.interaction.LennardJones93Wall.setParams() (*in module espressopp.interaction.LennardJones93Wall*), 191
espressopp.interaction.LennardJonesAutoBonds module, 158
espressopp.interaction.LennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*),

158
espressopp.interaction.LennardJonesCapped
module, 160
espressopp.interaction.LennardJonesCapped()
(*in module espressopp.interaction.LennardJonesCapped*), 160
espressopp.interaction.LennardJonesEnergyCapped
module, 163
espressopp.interaction.LennardJonesEnergyCapped()
(*in module espressopp.interaction.LennardJonesEnergyCapped*), 163
espressopp.interaction.LennardJonesExpand
module, 166
espressopp.interaction.LennardJonesExpand()
(*in module espressopp.interaction.LennardJonesExpand*), 166
espressopp.interaction.LennardJonesGeneric
module, 167
espressopp.interaction.LennardJonesGeneric()
(*in module espressopp.interaction.LennardJonesGeneric*), 167
espressopp.interaction.LennardJonesGromacs
module, 170
espressopp.interaction.LennardJonesGromacs()
(*in module espressopp.interaction.LennardJonesGromacs*), 170
espressopp.interaction.LennardJonesSoftcoreTI
module, 172
espressopp.interaction.LJcos
module, 174
espressopp.interaction.LJcos() (*in module espressopp.interaction.LJcos*), 174
espressopp.interaction.MirrorLennardJones
module, 176
espressopp.interaction.MirrorLennardJones()
(*in module espressopp.interaction.MirrorLennardJones*), 176
espressopp.interaction.Morse
module, 121
espressopp.interaction.Morse() (*in module espressopp.interaction.Morse*), 121
espressopp.interaction.OPLS
module, 137
espressopp.interaction.OPLS() (*in module espressopp.interaction.OPLS*), 137
espressopp.interaction.Potential
module, 193
espressopp.interaction.Potential.computeEnergy()
(*in module espressopp.interaction.Potential*), 193
espressopp.interaction.Potential.computeForce() (*in module espressopp.interaction.Potential*), 193
espressopp.interaction.PotentialVSpherePair
module, 193
espressopp.interaction.PotentialVSpherePair.computeEnergy()
(*in module espressopp.interaction.PotentialVSpherePair*), 193
espressopp.interaction.PotentialVSpherePair.computeForce()
(*in module espressopp.interaction.PotentialVSpherePair*), 193
espressopp.interaction.Quartic
module, 193
espressopp.interaction.Quartic() (*in module espressopp.interaction.Quartic*), 193
espressopp.interaction.ReactionFieldGeneralized
module, 194
espressopp.interaction.ReactionFieldGeneralized()
(*in module espressopp.interaction.ReactionFieldGeneralized*), 194
espressopp.interaction.ReactionFieldGeneralizedTI
module, 197
espressopp.interaction.SelfVSphere() (*in module espressopp.interaction.VSphereSelf*), 201
espressopp.interaction.SelfVSphere.getPotential() (*in module espressopp.interaction.VSphereSelf*), 201
espressopp.interaction.SelfVSphere.setPotential() (*in module espressopp.interaction.VSphereSelf*), 201
espressopp.interaction.SingleParticleHarmonicTrap()
(*in module espressopp.interaction.HarmonicTrap*), 121
espressopp.interaction.SingleParticleHarmonicTrap.setPotential()
(*in module espressopp.interaction.HarmonicTrap*), 121
espressopp.interaction.SingleParticleLennardJones93Wall()
(*in module espressopp.interaction.LennardJones93Wall*), 192
espressopp.interaction.SingleParticleLennardJones93Wall.setPotential
(*in module espressopp.interaction.LennardJones93Wall*), 192
espressopp.interaction.SingleParticlePotential
module, 199
espressopp.interaction.SingleParticlePotential.computeEnergy()
(*in module espressopp.interaction.SingleParticlePotential*), 200
espressopp.interaction.SingleParticlePotential.computeForce()
(*in module espressopp.interaction.SingleParticlePotential*), 200
espressopp.interaction.SmoothSquareWell

module, 203
espressopp.interaction.SoftCosine
module, 123
espressopp.interaction.SoftCosine() (*in module espressopp.interaction.SoftCosine*), 123
espressopp.interaction.StillingerWeberPairTerm
module, 138
espressopp.interaction.StillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.StillingerWeberPairTermCapped
module, 140
espressopp.interaction.StillingerWeberPairTermCapped() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 140
espressopp.interaction.StillingerWeberTripleTerm
module, 142
espressopp.interaction.StillingerWeberTripleTerm() (*in module espressopp.interaction.StillingerWeberTripleTerm*), 142
espressopp.interaction.Tabulated
module, 177
espressopp.interaction.Tabulated() (*in module espressopp.interaction.Tabulated*), 177
espressopp.interaction.TabulatedAngular
module, 186
espressopp.interaction.TabulatedAngular() (*in module espressopp.interaction.TabulatedAngular*), 187
espressopp.interaction.TabulatedDihedral
module, 190
espressopp.interaction.TabulatedDihedral() (*in module espressopp.interaction.TabulatedDihedral*), 190
espressopp.interaction.TersoffPairTerm
module, 143
espressopp.interaction.TersoffPairTerm() (*in module espressopp.interaction.TersoffPairTerm*), 144
espressopp.interaction.TersoffTripleTerm
module, 145
espressopp.interaction.VerletListAdressATHarmonic() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListAdressATHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListAdressATHarmonic.getVerletList() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListAdressATHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListAdressATLenJonesReacFieldGen() (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListAdressATLenJonesReacFieldGen.setL (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListAdressATLenJonesReacFieldGen.setL (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListAdressATLennardJones() (*in module espressopp.interaction.LennardJones*), 151
espressopp.interaction.VerletListAdressATLennardJones.getPotential((*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListAdressATLennardJones.getVerletList (*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListAdressATLennardJones.setPotential((*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic((*in module espressopp.interaction.LennardJones*), 157
espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic. (*in module espressopp.interaction.LennardJones*), 157
espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic. (*in module espressopp.interaction.LennardJones*), 157
espressopp.interaction.VerletListAdressATLJReacFieldGenHarmonic. (*in module espressopp.interaction.LennardJones*), 157
espressopp.interaction.VerletListAdressATLJReacFieldGenTab() (*in module espressopp.interaction.LennardJones*), 155
espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPot (*in module espressopp.interaction.LennardJones*), 155
espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPot (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListAdressATLJReacFieldGenTab.setPot (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListAdressATReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListAdressATReactionFieldGeneralized.g (*in module espressopp.interaction.ReactionFieldGeneralized*), 196

espressopp.interaction.VerletListAdressATReactionFieldGeneralized.setPotential() (*in module espressopp.interaction.ReactionFieldGeneralized*), 196
espressopp.interaction.VerletListAdressCGHarmonic() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListAdressCGHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListAdressCGHarmonic.getVerletList() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListAdressCGHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListAdressCGLennardJones() (*in module espressopp.interaction.LennardJones*), 153
espressopp.interaction.VerletListAdressCGLennardJones.getPotential() (*in module espressopp.interaction.LennardJones*), 153
espressopp.interaction.VerletListAdressCGLennardJones.getVerletList() (*in module espressopp.interaction.LennardJones*), 153
espressopp.interaction.VerletListAdressCGLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 153
espressopp.interaction.VerletListAdressCGTabulated() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListAdressCGTabulated.getPotential() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListAdressCGTabulated.getVerletList() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListAdressCGTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListAdressLennardJones() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListAdressLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListAdressLennardJones.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListAdressLennardJones2() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListAdressLennardJones2.setPotential() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListAdressLennardJones2.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListAdressLennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListAdressLennardJonesAutoBonds.setPotential() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListAdressLennardJonesCapped() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListAdressLennardJonesCapped.getPotential() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListAdressLennardJonesCapped.getPotential() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListAdressLennardJonesCapped.setPotential() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListAdressLennardJonesCapped.setPotential() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListAdressLennardJonesEnergyCapped() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.getPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.getPotentialAT() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.setPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListAdressLennardJonesEnergyCapped.setPotentialAT() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListAdressLennardJonesGeneric() (*in module espressopp.interaction.LennardJonesGeneric*), 148
espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotential() (*in module espressopp.interaction.LennardJonesGeneric*), 148
espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotentialCG() (*in module espressopp.interaction.LennardJonesGeneric*), 148

sopp.interaction.LennardJonesGeneric), 168
espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotentialAT() (*in module espressopp.interaction.LennardJonesGeneric*), 168
espressopp.interaction.VerletListAdressLennardJonesGeneric.setPotentialCG() (*in module espressopp.interaction.LennardJonesGeneric*), 168
espressopp.interaction.VerletListAdressLennardJonesGeneric2() (*in module espressopp.interaction.LennardJonesGeneric*), 168
espressopp.interaction.VerletListAdressLennardJonesGeneric2.setPotentialAT() (*in module espressopp.interaction.LennardJonesGeneric*), 168
espressopp.interaction.VerletListAdressLennardJonesGeneric2.setPotentialCG() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListAdressLennardJonesHarmonic() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListAdressLennardJonesHarmonic.setPotentialAT() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListAdressLennardJonesHarmonic.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListAdressLJcos() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListAdressLJcos.setPotentialAT() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListAdressLJcos.setPotentialCG() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListAdressMorse() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListAdressMorse.setPotentialAT() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListAdressMorse.setPotentialCG() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListAdressReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialAT() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialCG() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListAdressStillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListAdressStillingerWeberPairTerm.setPotentialAT() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListAdressStillingerWeberPairTerm.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListAdressStillingerWeberPairTerm.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped.setPotentialAT() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListAdressStillingerWeberPairTermCapped.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListAdressTabulated() (*in module espressopp.interaction.Tabulated*), 177
espressopp.interaction.VerletListAdressTabulated.setPotentialAT() (*in module espressopp.interaction.Tabulated*), 177
espressopp.interaction.VerletListAdressTabulated.setPotentialCG() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListAdressZero() (*in module espressopp.interaction.Zero*), 202
espressopp.interaction.VerletListAdressZero.setFixedTupleList() (*in module espressopp.interaction.Zero*), 202
espressopp.interaction.VerletListAdressZero.setPotentialAT() (*in module espressopp.interaction.Zero*), 202
espressopp.interaction.VerletListAdressZero.setPotentialCG() (*in module espressopp.interaction.Zero*), 202
espressopp.interaction.VerletListCoulombRSpace() (*in module espressopp.interaction.CoulombRSpace*), 128
espressopp.interaction.VerletListCoulombRSpace.getPotential() (*in module espressopp.interaction.CoulombRSpace*), 128
espressopp.interaction.VerletListCoulombRSpace.getVerletList() (*in module espressopp.interaction.CoulombRSpace*), 129
espressopp.interaction.VerletListCoulombRSpace.setPotential() (*in module espressopp.interaction.CoulombRSpace*), 129

espressopp.interaction.VerletListCoulombTruncatedUniqueCharge() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 130
espressopp.interaction.VerletListCoulombTruncatedUniqueCharge() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 130
espressopp.interaction.VerletListCoulombTruncatedUniqueCharge.setPotential() (*in module espressopp.interaction.CoulombTruncatedUniqueCharge*), 130
espressopp.interaction.VerletListGravityTruncated() (*in module espressopp.interaction.GravityTruncated*), 146
espressopp.interaction.VerletListGravityTruncated.getPotential() (*in module espressopp.interaction.GravityTruncated*), 146
espressopp.interaction.VerletListGravityTruncated.getVerletList() (*in module espressopp.interaction.GravityTruncated*), 147
espressopp.interaction.VerletListGravityTruncated.setPotential() (*in module espressopp.interaction.GravityTruncated*), 147
espressopp.interaction.VerletListHadressATHarmonic() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListHadressATHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListHadressATHarmonic.getVerletList() (*in module espressopp.interaction.Harmonic*), 120
espressopp.interaction.VerletListHadressATHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 119
espressopp.interaction.VerletListHadressATLenJonesReacFieldGen() (*in module espressopp.interaction.LennardJones*), 155
espressopp.interaction.VerletListHadressATLenJonesReacFieldGen.setPotential() (*in module espressopp.interaction.LennardJones*), 155
espressopp.interaction.VerletListHadressATLenJonesReacFieldGen.setPotential2() (*in module espressopp.interaction.LennardJones*), 155
espressopp.interaction.VerletListHadressATLennardJones() (*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListHadressATLennardJones.getPotential() (*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListHadressATLennardJones.getVerletList() (*in module espressopp.interaction.LennardJones*), 153
espressopp.interaction.VerletListHadressATLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 152
espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.getPotential() (*in module espressopp.interaction.LennardJones*), 157
espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotential() (*in module espressopp.interaction.LennardJones*), 158
espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotential2() (*in module espressopp.interaction.LennardJones*), 158
espressopp.interaction.VerletListHadressATLJReacFieldGenHarmonic.setPotential3() (*in module espressopp.interaction.LennardJones*), 158
espressopp.interaction.VerletListHadressATLJReacFieldGenTab() (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotential() (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotential2() (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListHadressATLJReacFieldGenTab.setPotential3() (*in module espressopp.interaction.LennardJones*), 156
espressopp.interaction.VerletListHadressATReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 197
espressopp.interaction.VerletListHadressATReactionFieldGeneralized.setPotential() (*in module espressopp.interaction.ReactionFieldGeneralized*), 197
espressopp.interaction.VerletListHadressATReactionFieldGeneralized.setPotential2() (*in module espressopp.interaction.ReactionFieldGeneralized*), 197
espressopp.interaction.VerletListHadressCGHarmonic() (*in module espressopp.interaction.Harmonic*), 120
espressopp.interaction.VerletListHadressCGHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 120
espressopp.interaction.VerletListHadressCGHarmonic.getVerletList() (*in module espressopp.interaction.Harmonic*), 120
espressopp.interaction.VerletListHadressCGHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 120
espressopp.interaction.VerletListHadressCGLennardJones() (*in module espressopp.interaction.LennardJones*), 153

espressopp.interaction.VerletListHadressCGLennardJones.getPotential() (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListHadressCGLennardJones.getVerletList() (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListHadressCGLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 154
espressopp.interaction.VerletListHadressCGTabulated() (*in module espressopp.interaction.Tabulated*), 179
espressopp.interaction.VerletListHadressCGTabulated.getPotential() (*in module espressopp.interaction.Tabulated*), 179
espressopp.interaction.VerletListHadressCGTabulated.getVerletList() (*in module espressopp.interaction.Tabulated*), 179
espressopp.interaction.VerletListHadressCGTabulated.setPotential() (*in module espressopp.interaction.Tabulated*), 179
espressopp.interaction.VerletListHadressLennardJones() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListHadressLennardJones.setPotentialAT() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListHadressLennardJones.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 149
espressopp.interaction.VerletListHadressLennardJones2() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLennardJones2.setPotentialAT() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLennardJones2.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListHadressLennardJonesAutoBonds.setPotential() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListHadressLennardJonesCapped() (*in module espressopp.interaction.LennardJonesCapped*), 161
espressopp.interaction.VerletListHadressLennardJonesCapped.getPotential() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.VerletListHadressLennardJonesCapped.getPotentialAT() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.VerletListHadressLennardJonesCapped.setPotential() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.VerletListHadressLennardJonesCapped.setPotentialAT() (*in module espressopp.interaction.LennardJonesCapped*), 162
espressopp.interaction.VerletListHadressLennardJonesEnergyCapped() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.getPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 164
espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.getPotentialAT() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.setPotential() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.VerletListHadressLennardJonesEnergyCapped.setPotentialAT() (*in module espressopp.interaction.LennardJonesEnergyCapped*), 165
espressopp.interaction.VerletListHadressLennardJonesGeneric() (*in module espressopp.interaction.LennardJonesGeneric*), 168
espressopp.interaction.VerletListHadressLennardJonesGeneric.setPotentialCG() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListHadressLennardJonesGeneric.setPotentialCG() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListHadressLennardJonesGeneric2() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListHadressLennardJonesGeneric2.setPotentialAT() (*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListHadressLennardJonesGeneric2.setPotentialAT()

(*in module espressopp.interaction.LennardJonesGeneric*), 169
espressopp.interaction.VerletListHadressLennardJonesHarmonic() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLennardJonesHarmonic.setPotentialAT() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLennardJonesHarmonic.setPotentialCG() (*in module espressopp.interaction.LennardJones*), 150
espressopp.interaction.VerletListHadressLJcos() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListHadressLJcos.setPotentialAT() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListHadressLJcos.setPotentialCG() (*in module espressopp.interaction.LJcos*), 175
espressopp.interaction.VerletListHadressMorse() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListHadressMorse.setPotentialAT() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListHadressMorse.setPotentialCG() (*in module espressopp.interaction.Morse*), 122
espressopp.interaction.VerletListHadressReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 196
espressopp.interaction.VerletListHadressReactionFieldGeneralized.setPotentialAT() (*in module espressopp.interaction.ReactionFieldGeneralized*), 196
espressopp.interaction.VerletListHadressReactionFieldGeneralized.setPotentialCG() (*in module espressopp.interaction.ReactionFieldGeneralized*), 196
espressopp.interaction.VerletListHadressStillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.VerletListHadressStillingerWeberPairTerm.setPotentialAT() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.VerletListHadressStillingerWeberPairTerm.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTerm*), 139
espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped.setPotentialAT() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListHadressStillingerWeberPairTermCapped.setPotentialCG() (*in module espressopp.interaction.StillingerWeberPairTermCapped*), 141
espressopp.interaction.VerletListHadressTabulated() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListHadressTabulated.setPotentialAT() (*in module espressopp.interaction.Tabulated*), 178
espressopp.interaction.VerletListHadressTabulated.setPotentialCG() (*in module espressopp.interaction.Tabulated*), 179
espressopp.interaction.VerletListHadressZero() (*in module espressopp.interaction.Zero*), 202
espressopp.interaction.VerletListHadressZero.setFixedTupleList() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.VerletListHadressZero.setPotentialAT() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.VerletListHadressZero.setPotentialCG() (*in module espressopp.interaction.Zero*), 203
espressopp.interaction.VerletListHarmonic() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.VerletListHarmonic.getPotential() (*in module espressopp.interaction.Harmonic*), 117
espressopp.interaction.VerletListHarmonic.setPotential() (*in module espressopp.interaction.Harmonic*), 118
espressopp.interaction.VerletListLennardJones() (*in module espressopp.interaction.LennardJones*), 147
espressopp.interaction.VerletListLennardJones.getPotential() (*in module espressopp.interaction.LennardJones*), 147
espressopp.interaction.VerletListLennardJones.getVerletList() (*in module espressopp.interaction.LennardJones*), 147
espressopp.interaction.VerletListLennardJones.setPotential() (*in module espressopp.interaction.LennardJones*), 148
espressopp.interaction.VerletListLennardJonesAutoBonds() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListLennardJonesAutoBonds.getPotential() (*in module espressopp.interaction.LennardJonesAutoBonds*), 159
espressopp.interaction.VerletListLennardJonesAutoBonds.getVerletList() (*in module espressopp.interaction.LennardJonesAutoBonds*),

159
espressopp.interaction.VerletListLennardJonesAutoBonds.setPotential() (in module espressopp.interaction.LennardJonesAutoBonds), 159
espressopp.interaction.VerletListLennardJonesCapped() (in module espressopp.interaction.LennardJonesCapped), 160
espressopp.interaction.VerletListLennardJonesCapped.getPotential() (in module espressopp.interaction.LennardJonesCapped), 160
espressopp.interaction.VerletListLennardJonesCapped.setPotential() (in module espressopp.interaction.LennardJonesCapped), 161
espressopp.interaction.VerletListLennardJonesEnergyCapped() (in module espressopp.interaction.LennardJonesEnergyCapped), 163
espressopp.interaction.VerletListLennardJonesEnergyCapped.getPotential() (in module espressopp.interaction.LennardJonesEnergyCapped), 163
espressopp.interaction.VerletListLennardJonesEnergyCapped.setPotential() (in module espressopp.interaction.LennardJonesEnergyCapped), 163
espressopp.interaction.VerletListLennardJonesExpand() (in module espressopp.interaction.LennardJonesExpand), 166
espressopp.interaction.VerletListLennardJonesExpand.getPotential() (in module espressopp.interaction.LennardJonesExpand), 166
espressopp.interaction.VerletListLennardJonesExpand.setPotential() (in module espressopp.interaction.LennardJonesExpand), 166
espressopp.interaction.VerletListLennardJonesGeneric() (in module espressopp.interaction.LennardJonesGeneric), 167
espressopp.interaction.VerletListLennardJonesGeneric.getPotential() (in module espressopp.interaction.LennardJonesGeneric), 167
espressopp.interaction.VerletListLennardJonesGeneric.getVerletList() (in module espressopp.interaction.LennardJonesGeneric), 168
espressopp.interaction.VerletListLennardJonesGeneric.setPotential() (in module espressopp.interaction.LennardJonesGeneric), 168
espressopp.interaction.VerletListLennardJonesGromacs() (in module espressopp.interaction.LennardJonesGromacs), 171
espressopp.interaction.VerletListLennardJonesGromacs.getPotential() (in module espressopp.interaction.LennardJonesGromacs), 171
espressopp.interaction.VerletListLennardJonesGromacs.setPotential() (in module espressopp.interaction.LennardJonesGromacs), 171
espressopp.interaction.VerletListLJcos() (in module espressopp.interaction.LJcos), 174
espressopp.interaction.VerletListLJcos.getPotential() (in module espressopp.interaction.LJcos), 174
espressopp.interaction.VerletListLJcos.getVerletList() (in module espressopp.interaction.LJcos), 174
espressopp.interaction.VerletListLJcos.setPotential() (in module espressopp.interaction.LJcos), 174
espressopp.interaction.VerletListMorse() (in module espressopp.interaction.Morse), 121
espressopp.interaction.VerletListMorse.getPotential() (in module espressopp.interaction.Morse), 121
espressopp.interaction.VerletListMorse.setPotential() (in module espressopp.interaction.Morse), 121
espressopp.interaction.VerletListPIadressNoDriftTabulated() (in module espressopp.interaction.Tabulated), 185
espressopp.interaction.VerletListPIadressNoDriftTabulated.getFixedT() (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.getNTrott (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.getSpeedu (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.getVerletI (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.setFixedT() (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.setNTrott (in module espressopp.interaction.Tabulated), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated.setPotenti (in module espressopp.interaction.Tabulated), 185
espressopp.interaction.VerletListPIadressNoDriftTabulated.setSpeedu

(*in module espressopp.interaction.Tabulated*), 186
espressopp.interaction.VerletListPIadressNoDriftTabulated() (*in module espressopp.interaction.Tabulated*), 185
espressopp.interaction.VerletListPIadressTabulated() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.VerletListPIadressTabulated.getFixedTupleList() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.getNTrotter() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.getSpeedup() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.getVerletList() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.setFixedTupleList() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.setNTrotter() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.setPotentialCL() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.setPotentialQM() (*in module espressopp.interaction.Tabulated*), 182
espressopp.interaction.VerletListPIadressTabulated.setSpeedup() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulated.setVerletList() (*in module espressopp.interaction.Tabulated*), 183
espressopp.interaction.VerletListPIadressTabulatedLJ() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.getFixedTupleList() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.getNTrotter() (*in module espressopp.interaction.Tabulated*), 185
espressopp.interaction.VerletListPIadressTabulatedLJ.getSpeedup() (*in module espressopp.interaction.Tabulated*), 185
espressopp.interaction.VerletListPIadressTabulatedLJ.getVerletList() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.setFixedTupleList() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.setNTrotter() (*in module espressopp.interaction.Tabulated*), 185
espressopp.interaction.VerletListPIadressTabulatedLJ.setPotentialCL() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.setPotentialQM() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListPIadressTabulatedLJ.setSpeedup() (*in module espressopp.interaction.Tabulated*), 185
espressopp.interaction.VerletListPIadressTabulatedLJ.setVerletList() (*in module espressopp.interaction.Tabulated*), 184
espressopp.interaction.VerletListReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralized*), 194
espressopp.interaction.VerletListReactionFieldGeneralized.getPotential() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListReactionFieldGeneralized.setPotential() (*in module espressopp.interaction.ReactionFieldGeneralized*), 195
espressopp.interaction.VerletListSoftCosine() (*in module espressopp.interaction.SoftCosine*), 123
espressopp.interaction.VerletListSoftCosine.setPotential() (*in module espressopp.interaction.SoftCosine*), 123
espressopp.interaction.VerletListStillingerWeberPairTerm() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListStillingerWeberPairTerm.getPotential() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListStillingerWeberPairTerm.getVerletList() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListStillingerWeberPairTerm.setPotential() (*in module espressopp.interaction.StillingerWeberPairTerm*), 138
espressopp.interaction.VerletListStillingerWeberPairTermCapped() (*in module espressopp.interaction.StillingerWeberPairTermCapped*),

140
espressopp.interaction.VerletListStillingerWeberPairTermCapped.getCapped() (in module espressopp.interaction.StillingerWeberPairTermCapped), 140
espressopp.interaction.VerletListStillingerWeberPairTermCapped.getPotential() (in module espressopp.interaction.StillingerWeberPairTermCapped), 140
espressopp.interaction.VerletListStillingerWeberPairTermCapped.getVerletList() (in module espressopp.interaction.StillingerWeberPairTermCapped), 140
espressopp.interaction.VerletListStillingerWeberPairTermCapped.setPotential() (in module espressopp.interaction.StillingerWeberPairTermCapped), 140
espressopp.interaction.VerletListStillingerWeberTripleTerm() (in module espressopp.interaction.StillingerWeberTripleTerm), 142
espressopp.interaction.VerletListStillingerWeberTripleTerm.getPotential() (in module espressopp.interaction.StillingerWeberTripleTerm), 143
espressopp.interaction.VerletListStillingerWeberTripleTerm.getVerletListTriple() (in module espressopp.interaction.StillingerWeberTripleTerm), 143
espressopp.interaction.VerletListStillingerWeberTripleTerm.setPotential() (in module espressopp.interaction.StillingerWeberTripleTerm), 143
espressopp.interaction.VerletListTabulated() (in module espressopp.interaction.Tabulated), 179
espressopp.interaction.VerletListTabulated.getPotential() (in module espressopp.interaction.Tabulated), 180
espressopp.interaction.VerletListTabulated.setPotential() (in module espressopp.interaction.Tabulated), 180
espressopp.interaction.VerletListTersoffPairTerm() (in module espressopp.interaction.TersoffPairTerm), 144
espressopp.interaction.VerletListTersoffPairTerm.getPotential() (in module espressopp.interaction.TersoffPairTerm), 144
espressopp.interaction.VerletListTersoffPairTerm.getVerletList() (in module espressopp.interaction.TersoffPairTerm), 144
espressopp.interaction.VerletListTersoffPairTerm.setPotential() (in module espressopp.interaction.TersoffPairTerm), 144
espressopp.interaction.VerletListTersoffTripleTerm() (in module espressopp.interaction.TersoffTripleTerm), 145
espressopp.interaction.VerletListTersoffTripleTerm.getPotential() (in module espressopp.interaction.TersoffTripleTerm), 145
espressopp.interaction.VerletListTersoffTripleTerm.getVerletListTriple() (in module espressopp.interaction.TersoffTripleTerm), 145
espressopp.interaction.VerletListTersoffTripleTerm.setPotential() (in module espressopp.interaction.TersoffTripleTerm), 145
espressopp.interaction.VerletListVSpherePair() (in module espressopp.interaction.VSpherePair), 200
espressopp.interaction.VerletListVSpherePair.getPotential() (in module espressopp.interaction.VSpherePair), 200
espressopp.interaction.VerletListVSpherePair.getVerletList() (in module espressopp.interaction.VSpherePair), 200
espressopp.interaction.VerletListVSpherePair.setPotential() (in module espressopp.interaction.VSpherePair), 200
espressopp.interaction.VerletListZero() (in module espressopp.interaction.Zero), 202
espressopp.interaction.VerletListZero.getPotential() (in module espressopp.interaction.Zero), 202
espressopp.interaction.VerletListZero.setFixedTupleList() (in module espressopp.interaction.Zero), 202
espressopp.interaction.VerletListZero.setPotential() (in module espressopp.interaction.Zero), 202
espressopp.interaction.VSpherePair module, 200
espressopp.interaction.VSpherePair() (in module espressopp.interaction.VSpherePair), 200
espressopp.interaction.VSphereSelf module, 201
espressopp.interaction.VSphereSelf() (in module espressopp.interaction.VSphereSelf), 201
espressopp.interaction.Zero module, 201
espressopp.interaction.Zero() (in module espressopp.interaction.Zero), 202
espressopp.io.DumpGRO module, 204
espressopp.io.DumpGRO() (in module espressopp.io.DumpGRO), 205
espressopp.io.DumpGRO.dump() (in module espressopp.io.DumpGRO), 205
espressopp.io.DumpGROAdress module, 205
espressopp.io.DumpGROAdress() (in module espressopp.io.DumpGROAdress), 206
espressopp.io.DumpGROAdress.dump() (in module

espressopp.io.DumpGROAdress), 206
espressopp.io.DumpH5MD
module, 208
espressopp.io.DumpH5MD() (*in module espressopp.io.DumpH5MD*), 208
espressopp.io.DumpXYZ
module, 206
espressopp.io.DumpXYZ() (*in module espressopp.io.DumpXYZ*), 207
espressopp.io.DumpXYZ.dump() (*in module espressopp.io.DumpXYZ*), 208
espressopp.MissingFixedPairList() (*in module espressopp.Exceptions*), 210
espressopp.MultiSystem
module, 219
espressopp.MultiSystem() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.beginSystemDefinition() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.runAnalysisNPart() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.runAnalysisPotential() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.runAnalysisTemperature() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.runIntegrator() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.setAnalysisNPart() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.setAnalysisPotential() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.setAnalysisTemperature() (*in module espressopp.MultiSystem*), 220
espressopp.MultiSystem.setIntegrator() (*in module espressopp.MultiSystem*), 220
espressopp.ParallelTempering
module, 220
espressopp.ParallelTempering() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.endDefiningSystem() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.exchange() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.getNumberOfCPUsPerSystem() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.getNumberOfSystems() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.run() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.setAnalysisE() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.setAnalysisNPart() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.setAnalysisT() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.setIntegrator() (*in module espressopp.ParallelTempering*), 221
espressopp.ParallelTempering.startDefiningSystem() (*in module espressopp.ParallelTempering*), 221
espressopp.Particle
module, 221
espressopp.Particle (*class in espressopp.Particle*), 222
espressopp.ParticleAccess
module, 223
espressopp.ParticleAccess.perform_action() (*in module espressopp.ParticleAccess*), 223
espressopp.ParticleDoesNotExistHere() (*in module espressopp.Exceptions*), 210
espressopp.ParticleGroup
module, 223
espressopp.ParticleGroup() (*in module espressopp.ParticleGroup*), 223
espressopp.ParticleGroup.add() (*in module espressopp.ParticleGroup*), 223
espressopp.ParticleGroup.has() (*in module espressopp.ParticleGroup*), 223
espressopp.ParticleGroup.show() (*in module espressopp.ParticleGroup*), 224
espressopp.ParticleGroup.size() (*in module espressopp.ParticleGroup*), 224
espressopp.pmi
module, 224
espressopp.Quaternion
module, 228
espressopp.Real3D
module, 230
espressopp.Real3D() (*in module espressopp.Real3D*), 230
espressopp.Real3D.x() (*in module espressopp.Real3D*), 230
espressopp.Real3D.y() (*in module espressopp.Real3D*), 230
espressopp.Real3D.z() (*in module espressopp.Real3D*), 230
espressopp.RealND
module, 231
espressopp.RealND() (*in module espressopp.RealND*), 231
espressopp.standard_system.Default
module, 236
espressopp.standard_system.KGMelt
module, 237
espressopp.standard_system.KGMelt() (*in module espressopp.standard_system.KGMelt*), 237
espressopp.standard_system.LennardJones
module, 237
espressopp.standard_system.LennardJones() (*in module espres-*

sopp.standard_system.LennardJones), 237
espressopp.standard_system.Minimal
module, 237
espressopp.standard_system.Minimal() (*in module espressopp.standard_system.Minimal*), 237
espressopp.standard_system.PolymerMelt
module, 238
espressopp.standard_system.PolymerMelt() (*in module espressopp.standard_system.PolymerMelt*), 238
espressopp.storage.DomainDecomposition
module, 238
espressopp.storage.DomainDecomposition() (*in module espressopp.storage.DomainDecomposition*), 238
espressopp.storage.DomainDecomposition.getCellGrid() (*in module espressopp.storage.DomainDecomposition*), 238
espressopp.storage.DomainDecomposition.getNodeGrid() (*in module espressopp.storage.DomainDecomposition*), 238
espressopp.storage.DomainDecompositionAdress
module, 239
espressopp.storage.DomainDecompositionAdress() (*in module espressopp.storage.DomainDecompositionAdress*), 239
espressopp.storage.DomainDecompositionNonBlocking
module, 239
espressopp.storage.DomainDecompositionNonBlocking() (*in module espressopp.storage.DomainDecompositionNonBlocking*), 239
espressopp.storage.Storage
module, 239
espressopp.storage.Storage.addAdrATParticle() (*in module espressopp.storage.Storage*), 240
espressopp.storage.Storage.addParticle() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.addParticles() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.clearSavedPositions() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.getParticle() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.getRealParticleIDs() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.modifyParticle() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.particleExists() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.printRealParticles() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.removeAllParticles() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.removeParticle() (*in module espressopp.storage.Storage*), 241
espressopp.storage.Storage.restorePositions() (*in module espressopp.storage.Storage*), 242
espressopp.storage.Storage.savePositions() (*in module espressopp.storage.Storage*), 242
espressopp.storage.Storage.setFixedTuplesAdress() (*in module espressopp.storage.Storage*), 242
espressopp.System
module, 231
espressopp.System() (*in module espressopp.System*), 232
espressopp.System.addInteraction() (*in module espressopp.System*), 232
espressopp.System.getAllInteractions() (*in module espressopp.System*), 232
espressopp.System.getInteraction() (*in module espressopp.System*), 232
espressopp.System.getNumberOfInteractions() (*in module espressopp.System*), 232
espressopp.System.removeInteraction() (*in module espressopp.System*), 232
espressopp.System.removeInteractionByName() (*in module espressopp.System*), 232
espressopp.System.scaleVolume() (*in module espressopp.System*), 233
espressopp.System.setTrace() (*in module espressopp.System*), 233
espressopp.Tensor
module, 233
espressopp.toInt3D() (*in module espressopp.Int3D*), 219
espressopp.toInt3DFromVector() (*in module espressopp.Int3D*), 219
espressopp.tools.analyse
module, 242
espressopp.tools.applyBoreschRestraints() (*in module espressopp.tools.prepareComplexMolecules*), 253
espressopp.tools.decomp
module, 245
espressopp.tools.DumpConfigurations
module, 246
espressopp.tools.espresso_old
module, 248
espressopp.tools.findConstrainedBonds() (*in module espressopp.tools.prepareComplexMolecules*), 252
espressopp.tools.getInternalNonbondedInteractions() (*in module espressopp.tools.prepareComplexMolecules*), 253
espressopp.tools.gromacs
module, 248
espressopp.tools.info

module, 243
espressopp.tools.io_extended
module, 250
espressopp.tools.lammps
module, 251
espressopp.tools.lattice
module, 244
espressopp.tools.pathintegral
module, 251
espressopp.tools.pdb
module, 252
espressopp.tools.pdbread() (*in module espressopp.tools.pdb*), 252
espressopp.tools.pdbwrite() (*in module espressopp.tools.pdb*), 252
espressopp.tools.povwrite
module, 252
espressopp.tools.prepareAdress
module, 254
espressopp.tools.prepareComplexMolecules
module, 252
espressopp.tools.psf
module, 254
espressopp.tools.psfread() (*in module espressopp.tools.psf*), 255
espressopp.tools.psfwrite() (*in module espressopp.tools.psf*), 254
espressopp.tools.readSimpleSystem() (*in module espressopp.tools.prepareComplexMolecules*), 253
espressopp.tools.replicate
module, 244
espressopp.tools.tabulated
module, 255
espressopp.tools.timers
module, 243
espressopp.tools.topology
module, 244
espressopp.tools.topology_helper
module, 255
espressopp.tools.units
module, 255
espressopp.tools.velocities
module, 245
espressopp.tools.vmd
module, 243
espressopp.tools.warmup
module, 245
espressopp.toReal3D() (*in module espressopp.Real3D*), 231
espressopp.toReal3DFromVector() (*in module espressopp.Real3D*), 231
espressopp.toRealND() (*in module espressopp.RealND*), 231
espressopp.toRealNDFromVector() (*in module espressopp.RealND*), 231
espressopp.UnknownParticleProperty() (*in module espressopp.Exceptions*), 210
espressopp.VerletList
module, 233
espressopp.VerletList() (*in module espressopp.VerletList*), 233
espressopp.VerletList.exclude() (*in module espressopp.VerletList*), 233
espressopp.VerletList.getAllPairs() (*in module espressopp.VerletList*), 233
espressopp.VerletList.localSize() (*in module espressopp.VerletList*), 233
espressopp.VerletList.totalSize() (*in module espressopp.VerletList*), 233
espressopp.VerletListAdress
module, 234
espressopp.VerletListAdress() (*in module espressopp.VerletListAdress*), 235
espressopp.VerletListAdress.addAdrParticles() (*in module espressopp.VerletListAdress*), 235
espressopp.VerletListAdress.exclude() (*in module espressopp.VerletListAdress*), 235
espressopp.VerletListAdress.rebuild() (*in module espressopp.VerletListAdress*), 235
espressopp.VerletListAdress.totalSize() (*in module espressopp.VerletListAdress*), 235
espressopp.VerletListTriple
module, 235
espressopp.VerletListTriple() (*in module espressopp.VerletListTriple*), 235
espressopp.VerletListTriple.exclude() (*in module espressopp.VerletListTriple*), 235
espressopp.VerletListTriple.getAllTriples() (*in module espressopp.VerletListTriple*), 236
espressopp.VerletListTriple.localSize() (*in module espressopp.VerletListTriple*), 236
espressopp.VerletListTriple.totalSize() (*in module espressopp.VerletListTriple*), 236
espressopp.Version
module, 236
espressopp.Version() (*in module espressopp.Version*), 236
espressopppp.integrator.Rattle() (*in module espressopp.integrator.Rattle*), 106
espressopppp.integrator.Rattle.addConstrainedBonds() (*in module espressopp.integrator.Rattle*), 106
espressopppp.interaction.CoulombTruncated() (*in module espressopp.interaction.CoulombTruncated*), 129
espressopppp.interaction.FixedPairListTypesCoulombTruncated() (*in module espressopp.interaction.CoulombTruncated*), 129

espressopppp.interaction.FixedPairListTypesCoulombTruncated.setPotential() (*in module espressopp.interaction.CoulombTruncated*), 129
espressopppp.interaction.LennardJonesSoftcoreTI() (*in module espressopp.interaction.LennardJonesSoftcoreTI*), 173
espressopppp.interaction.LennardJonesSoftcoreTI.addPids() (*in module espressopp.interaction.LennardJonesSoftcoreTI*), 173
espressopppp.interaction.ReactionFieldGeneralizedTI() (*in module espressopp.interaction.ReactionFieldGeneralizedTI*), 199
espressopppp.interaction.ReactionFieldGeneralizedTI.addPids() (*in module espressopp.interaction.ReactionFieldGeneralizedTI*), 199
espressopppp.interaction.VerletListAdressLennardJones() (*in module espressopp.interaction.LennardJonesSoftcoreTI*), 173
espressopppp.interaction.VerletListAdressLennardJones.setPotentialAT() (*in module espressopp.interaction.LennardJonesSoftcoreTI*), 174
espressopppp.interaction.VerletListAdressLennardJones.setPotentialCG() (*in module espressopp.interaction.LennardJonesSoftcoreTI*), 174
espressopppp.interaction.VerletListAdressReactionFieldGeneralized() (*in module espressopp.interaction.ReactionFieldGeneralizedTI*), 199
espressopppp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialAT() (*in module espressopp.interaction.ReactionFieldGeneralizedTI*), 199
espressopppp.interaction.VerletListAdressReactionFieldGeneralized.setPotentialCG() (*in module espressopp.interaction.ReactionFieldGeneralizedTI*), 199
espressopppp.interaction.VerletListCoulombTruncated() (*in module espressopp.interaction.CoulombTruncated*), 129
espressopppp.interaction.VerletListCoulombTruncated.getPotential() (*in module espressopp.interaction.CoulombTruncated*), 129
espressopppp.interaction.VerletListCoulombTruncated.setPotential() (*in module espressopp.interaction.CoulombTruncated*), 129
euler_from_matrix() (*in module espressopp.external.transformations*), 64
euler_from_quaternion() (*in module espressopp.external.transformations*), 64
euler_matrix() (*in module espressopp.external.transformations*), 64
exec_() (*in module espressopp.pmi*), 226

F

f_max (*in module espressopp.integrator.MinimizeEnergy*), 95
fastreadxyz() (*in module espressopp.tools.DumpConfigurations*), 247
fastwritexyz() (*in module espressopp.tools.DumpConfigurations*), 247
fastwritexyz_standard() (*in module espressopp.tools.DumpConfigurations*), 247
final_info() (*in module espressopp.tools.analyse*), 242
finalizeWorkers() (*in module espressopp.pmi*), 227
FixedQuadrupleListDihedralHarmonicLocal (*class in espressopp.interaction.DihedralHarmonic*), 133
FixedQuadrupleListDihedralHarmonicNCosLocal (*class in espressopp.interaction.DihedralHarmonicNCos*), 135
FixedTripleListPIadressTabulatedAngularLocal (*class in espressopp.interaction.TabulatedAngular*), 189

G

GammaVariate (*class in espressopp.esutil.GammaVariate*), 57
gaussian() (*in module espressopp.tools.velocities*), 245
getAllBonds() (*in module espressopp.tools.info*), 243
getAllParticles() (*in module espressopp.tools.info*), 243
getCentroidThermostat() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
getClmassmultiplier() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
getCMDparameter() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 102
getConstKinMass() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getconstrain() (*espressopp.external.transformations.Arcball method*), 61
getFixedPairList() (*espressopp.interaction.FENE.espressopp.interaction.FixedPairListFEN method*), 114
getFixedPairList() (*espressopp.interaction.FENECapped.espressopp.interaction.FixedPairL*

method), 115
getFixedQuadrupleList() (*espressopp.interaction.DihedralHarmonic.FixedQuadrupleListDihedralHarmonic method*), 133
getFixedQuadrupleList() (*espressopp.interaction.DihedralHarmonicNCos.FixedQuadrupleListDihedralHarmonicNCos method*), 135
getFixedTripleList() (*espressopp.interaction.AngularCosineSquared.espressopp.interaction.FixedTripleListAngularCosineSquared method*), 111
getFixedTripleList() (*espressopp.interaction.AngularHarmonic.espressopp.interaction.FixedTripleListAngularHarmonic method*), 112
getFixedTripleList() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngular method*), 189
getFixedTupleList() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngular method*), 189
getFoldedPosition() (*espressopp.bc.BC.espressopp.bc.BC method*), 55
getGamma() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getKTI() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getLBMom() (*espressopp.integrator.LatticeBoltzmann.espressopp.integrator.LatticeBoltzmann method*), 88
getMinimumImageVector() (*espressopp.bc.BC.espressopp.bc.BC method*), 55
getmStep() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
getNtrotter() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getNTrotter() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngular method*), 189
getPILE() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getPILElambda() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getPotential() (*espressopp.interaction.FENE.espressopp.interaction.FixedPairListFENE method*), 114
getPotential() (*espressopp.interaction.FENECapped.espressopp.interaction.FixedPairListFENECapped method*), 116
getPotential() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngular method*), 189
getRandomPos() (*espressopp.bc.BC.espressopp.bc.BC method*), 56
getRealKinMass() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getSpeedup() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getSpeedup() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngular method*), 189
getSpeeL() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
getTemperature() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getTimeStep() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getUnfoldedPosition() (*espressopp.bc.BC.espressopp.bc.BC method*), 56
getVerletList() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
getVerletlistBuilds() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 103
Grid (*class in espressopp.esutil.Grid*), 58

H

Harmonic (*class in espressopp.interaction.Harmonic*), 120
HarmonicTrap (*class in espressopp.interaction.HarmonicTrap*), 121

I

identity_matrix() (*in module espressopp.external.transformations*), 65
import_() (*in module espressopp.pmi*), 227
info() (*in module espressopp.tools.analyse*), 242
inverse_matrix() (*in module espressopp.external.transformations*), 65
invoke() (*in module espressopp.pmi*), 227
is_same_transform() (*in module espressopp.external.transformations*), 65

K

keepLBDump() (*espressopp.integrator.LatticeBoltzmann.espressopp.integrator.LatticeBo method*), 88

L

LennardJones (*class in espressopp.interaction.LennardJones*), 158
LennardJones93Wall (*class in espressopp.interaction.LennardJones93Wall*), 192
LennardJonesAutoBonds (*class in espressopp.interaction.LennardJonesAutoBonds*), 160

LennardJonesCapped (*class in espressopp.interaction.LennardJonesCapped*), 163
LennardJonesEnergyCapped (*class in espressopp.interaction.LennardJonesEnergyCapped*), 166
LennardJonesExpand (*class in espressopp.interaction.LennardJonesExpand*), 167
LennardJonesGeneric (*class in espressopp.interaction.LennardJonesGeneric*), 170
LennardJonesGromacs (*class in espressopp.interaction.LennardJonesGromacs*), 171
LennardJonesSoftcoreTI (*class in espressopp.interaction.LennardJonesSoftcoreTI*), 174
LJcos (*class in espressopp.interaction.LJcos*), 176
locateItem() (*espressopp.esutil.Collectives.espressopp.esutil method*), 57

M

matrix() (*espressopp.external.transformations.Arcball method*), 61
MirrorLennardJones (*class in espressopp.interaction.MirrorLennardJones*), 177
MissingFixedPairList, 210
module
espressopp.analysis.AdressDensity, 46
espressopp.analysis.AllParticlePos, 33
espressopp.analysis.AnalysisBase, 39
espressopp.analysis.Autocorrelation, 40
espressopp.analysis.CenterOfMass, 46
espressopp.analysis.CMVelocity, 40
espressopp.analysis.ConfigsParticleDecomp, 41
espressopp.analysis.Configurations, 41
espressopp.analysis.ConfigurationsExt, 42
espressopp.analysis.Energy, 43
espressopp.analysis.IntraChainDistSq, 44
espressopp.analysis.LBOutput, 33
espressopp.analysis.LBOutputScreen, 34
espressopp.analysis.LBOutputVzInTime, 34
espressopp.analysis.LBOutputVzOfX, 35
espressopp.analysis.MaxPID, 47
espressopp.analysis.MeanSquareDispl, 44
espressopp.analysis.MeanSquareInternalDist, 45
espressopp.analysis.NeighborFluctuation, 47
espressopp.analysis.NPart, 47
espressopp.analysis.NPartSubregion, 45
espressopp.analysis.Observable, 51
espressopp.analysis.OrderParameter, 35
espressopp.analysis.ParticleRadiusDistribution, 35
espressopp.analysis.PotentialEnergy, 47
espressopp.analysis.Pressure, 47
espressopp.analysis.PressureTensor, 35
espressopp.analysis.PressureTensorLayer, 36
espressopp.analysis.PressureTensorMultiLayer, 37
espressopp.analysis.RadGyrXProfilePI, 51
espressopp.analysis.RadialDistrF, 48
espressopp.analysis.RDFatomistic, 48
espressopp.analysis.StaticStructF, 50
espressopp.analysis.SubregionTracking, 52
espressopp.analysis.SystemMonitor, 53
espressopp.analysis.Temperature, 39
espressopp.analysis.Test, 39
espressopp.analysis.Velocities, 54
espressopp.analysis.VelocityAutocorrelation, 54
espressopp.analysis.Viscosity, 54
espressopp.analysis.XDensity, 50
espressopp.analysis.XPressure, 51
espressopp.analysis.XTemperature, 51
espressopp.bc.BC, 55
espressopp.bc.OrthorhombicBC, 56
espressopp.bc.SlabBC, 56
espressopp.check.System, 57
espressopp.esutil.Collectives, 57
espressopp.esutil.GammaVariate, 57
espressopp.esutil.Grid, 57
espressopp.esutil.NormalVariate, 58
espressopp.esutil.RNG, 58
espressopp.esutil.UniformOnSphere, 58
espressopp.Exceptions, 210
espressopp.external.transformations, 58
espressopp.FixedLocalTupleList, 211
espressopp.FixedPairDistList, 211
espressopp.FixedPairList, 212
espressopp.FixedPairListAdress, 213
espressopp.FixedQuadrupleAngleList, 213
espressopp.FixedQuadrupleList, 214
espressopp.FixedQuadrupleListAdress, 215
espressopp.FixedSingleList, 215
espressopp.FixedTripleAngleList, 216
espressopp.FixedTripleList, 217
espressopp.FixedTripleListAdress, 217
espressopp.FixedTupleList, 218
espressopp.FixedTupleListAdress, 218
espressopp.Int3D, 218
espressopp.integrator.Adress, 74
espressopp.integrator.AssociationReaction, 75
espressopp.integrator.BerendsenBarostat, 75
espressopp.integrator.BerendsenBarostatAnisotropic, 76
espressopp.integrator.BerendsenThermostat, 78
espressopp.integrator.CapForce, 79
espressopp.integrator.DPDThermostat, 80

espressopp.integrator.EmptyExtension, 80
espressopp.integrator.ExtAnalyze, 80
espressopp.integrator.Extension, 80
espressopp.integrator.ExtForce, 80
espressopp.integrator.FixPositions, 81
espressopp.integrator.FreeEnergyCompensation, 81
espressopp.integrator.GeneralizedLangevinThermostat, 82
espressopp.integrator.Isokinetic, 82
espressopp.integrator.LangevinBarostat, 82
espressopp.integrator.LangevinThermostat, 84
espressopp.integrator.LangevinThermostat1D, 85
espressopp.integrator.LangevinThermostatHybrid, 85
espressopp.integrator.LangevinThermostatOnGroup, 86
espressopp.integrator.LangevinThermostatOnRadius, 86
espressopp.integrator.LatticeBoltzmann, 87
espressopp.integrator.LBInit, 91
espressopp.integrator.LBInitConstForce, 92
espressopp.integrator.LBInitPeriodicForce, 93
espressopp.integrator.LBInitPopUniform, 92
espressopp.integrator.LBInitPopWave, 92
espressopp.integrator.MDIntegrator, 93
espressopp.integrator.MinimizeEnergy, 93
espressopp.integrator.OnTheFlyFEC, 95
espressopp.integrator.PIAdressIntegrator, 95
espressopp.integrator.Rattle, 105
espressopp.integrator.Settle, 106
espressopp.integrator.StochasticVelocityRescaling, 107
espressopp.integrator.TDforce, 107
espressopp.integrator.VelocityVerlet, 108
espressopp.integrator.VelocityVerletOnGroup, 109
espressopp.integrator.VelocityVerletOnRadius, 109
espressopp.integrator.VelocityVerletRESPA, 108
espressopp.interaction.AngularCosineSquared, 110
espressopp.interaction.AngularHarmonic, 112
espressopp.interaction.AngularPotential, 110
espressopp.interaction.ConstrainCOM, 131
espressopp.interaction.ConstrainRG, 132
espressopp.interaction.Cosine, 113
espressopp.interaction.CoulombKSpaceEwald, 124
espressopp.interaction.CoulombKSpaceP3M, 126
espressopp.interaction.CoulombRSpace, 127
espressopp.interaction.CoulombTruncated, 129
espressopp.interaction.CoulombTruncatedUniqueCharge, 130
espressopp.interaction.DihedralHarmonic, 133
espressopp.interaction.DihedralHarmonicCos, 134
espressopp.interaction.DihedralHarmonicNCos, 134
espressopp.interaction.DihedralPotential, 135
espressopp.interaction.DihedralRB, 136
espressopp.interaction.FENE, 114
espressopp.interaction.FENECapped, 115
espressopp.interaction.GravityTruncated, 146
espressopp.interaction.Harmonic, 116
espressopp.interaction.HarmonicTrap, 120
espressopp.interaction.Interaction, 192
espressopp.interaction.LennardJones, 147
espressopp.interaction.LennardJones93Wall, 191
espressopp.interaction.LennardJonesAutoBonds, 158
espressopp.interaction.LennardJonesCapped, 160
espressopp.interaction.LennardJonesEnergyCapped, 163
espressopp.interaction.LennardJonesExpand, 166
espressopp.interaction.LennardJonesGeneric, 167
espressopp.interaction.LennardJonesGromacs, 170
espressopp.interaction.LennardJonesSoftcoreTI, 172
espressopp.interaction.LJcos, 174
espressopp.interaction.MirrorLennardJones, 176
espressopp.interaction.Morse, 121
espressopp.interaction.OPLS, 137
espressopp.interaction.Potential, 193
espressopp.interaction.PotentialVSpherePair, 193
espressopp.interaction.Quartic, 193
espressopp.interaction.ReactionFieldGeneralized, 194
espressopp.interaction.ReactionFieldGeneralizedTI, 197
espressopp.interaction.SingleParticlePotential, 199
espressopp.interaction.SmoothSquareWell, 203
espressopp.interaction.SoftCosine, 123
espressopp.interaction.StillingerWeberPairTerm, 138
espressopp.interaction.StillingerWeberPairTermCapped, 140
espressopp.interaction.StillingerWeberTripleTerm, 142
espressopp.interaction.Tabulated, 177

espressopp.interaction.TabulatedAngular, 186
espressopp.interaction.TabulatedDihedral, 190
espressopp.interaction.TersoffPairTerm, 143
espressopp.interaction.TersoffTripleTerm, 145
espressopp.interaction.VSpherePair, 200
espressopp.interaction.VSphereSelf, 201
espressopp.interaction.Zero, 201
espressopp.io.DumpGRO, 204
espressopp.io.DumpGROAdress, 205
espressopp.io.DumpH5MD, 208
espressopp.io.DumpXYZ, 206
espressopp.MultiSystem, 219
espressopp.ParallelTempering, 220
espressopp.Particle, 221
espressopp.ParticleAccess, 223
espressopp.ParticleGroup, 223
espressopp.pmi, 224
espressopp.Quaternion, 228
espressopp.Real3D, 230
espressopp.RealND, 231
espressopp.standard_system.Default, 236
espressopp.standard_system.KGMelt, 237
espressopp.standard_system.LennardJones, 237
espressopp.standard_system.Minimal, 237
espressopp.standard_system.PolymerMelt, 238
espressopp.storage.DomainDecomposition, 238
espressopp.storage.DomainDecompositionAdress, 239
espressopp.storage.DomainDecompositionNonBlocking, 239
espressopp.storage.Storage, 239
espressopp.System, 231
espressopp.Tensor, 233
espressopp.tools.analyse, 242
espressopp.tools.decomp, 245
espressopp.tools.DumpConfigurations, 246
espressopp.tools.espresso_old, 248
espressopp.tools.gromacs, 248
espressopp.tools.info, 243
espressopp.tools.io_extended, 250
espressopp.tools.lammps, 251
espressopp.tools.lattice, 244
espressopp.tools.pathintegral, 251
espressopp.tools.pdb, 252
espressopp.tools.povwrite, 252
espressopp.tools.prepareAdress, 254
espressopp.tools.prepareComplexMolecules, 252
espressopp.tools.psf, 254
espressopp.tools.replicate, 244
espressopp.tools.tabulated, 255
espressopp.tools.timers, 243
espressopp.tools.topology, 244
espressopp.tools.topology_helper, 255
espressopp.tools.units, 255
espressopp.tools.velocities, 245
espressopp.tools.vmd, 243
espressopp.tools.warmup, 245
espressopp.VerletList, 233
espressopp.VerletListAdress, 234
espressopp.VerletListTriple, 235
espressopp.Version, 236
Morse (*class in espressopp.interaction.Morse*), 123
MultiSystem (*class in espressopp.MultiSystem*), 220
MultiSystemLocal (*class in espressopp.MultiSystem*), 220

N

next() (*espressopp.external.transformations.Arcball method*), 61
NormalVariate (*class in espressopp.esutil.NormalVariate*), 58
NPartSubregionLocal (*class in espressopp.analysis.NPartSubregion*), 46

O

OPLS (*class in espressopp.interaction.OPLS*), 137
orthogonalization_matrix() (*in module espressopp.external.transformations*), 65
OrthorhombicBC() (*espressopp.bc.OrthorhombicBC.espressopp.bc method*), 56

P

ParticleDoesNotExistHere, 211
ParticleLocal (*class in espressopp.Particle*), 223
PIAdressIntegratorLocal (*class in espressopp.integrator.PIAdressIntegrator*), 102
place() (*espressopp.external.transformations.Arcball method*), 61
polymerRW() (*in module espressopp.tools.topology*), 244
projection_from_matrix() (*in module espressopp.external.transformations*), 65
projection_matrix() (*in module espressopp.external.transformations*), 66
Proxy (*class in espressopp.pmi*), 225

Q

Quartic (*class in espressopp.interaction.Quartic*), 194
quaternion_about_axis() (*in module espressopp.external.transformations*), 67
quaternion_conjugate() (*in module espressopp.external.transformations*), 67
quaternion_from_euler() (*in module espressopp.external.transformations*), 67
quaternion_from_matrix() (*in module espressopp.external.transformations*), 67

quaternion_imag() (*in module espressopp.external.transformations*), 68
quaternion_inverse() (*in module espressopp.external.transformations*), 68
quaternion_matrix() (*in module espressopp.external.transformations*), 68
quaternion_multiply() (*in module espressopp.external.transformations*), 68
quaternion_real() (*in module espressopp.external.transformations*), 68
quaternion_slerp() (*in module espressopp.external.transformations*), 68

R

RadGyrXProfilePILocal (*class in espressopp.analysis.RadGyrXProfilePI*), 52
random_quaternion() (*in module espressopp.external.transformations*), 69
random_rotation_matrix() (*in module espressopp.external.transformations*), 69
random_vector() (*in module espressopp.external.transformations*), 69
ReactionFieldGeneralized (*class in espressopp.interaction.ReactionFieldGeneralized*), 197
ReactionFieldGeneralizedTI (*class in espressopp.interaction.ReactionFieldGeneralizedTI*), 199
read() (*in module espressopp.tools.espresso_old*), 248
read() (*in module espressopp.tools.gromacs*), 250
readxyz() (*in module espressopp.tools.DumpConfigurations*), 247
readxyzr() (*in module espressopp.tools.DumpConfigurations*), 247
receive() (*in module espressopp.pmi*), 227
reduce() (*in module espressopp.pmi*), 227
reflection_from_matrix() (*in module espressopp.external.transformations*), 69
reflection_matrix() (*in module espressopp.external.transformations*), 70
registerAtExit() (*in module espressopp.pmi*), 228
replicate() (*in module espressopp.tools.replicate*), 244
ReplicateParallel (*class in espressopp.tools.replicate*), 244
RNG (*class in espressopp.esutil.RNG*), 58
rotation_from_matrix() (*in module espressopp.external.transformations*), 70
rotation_matrix() (*in module espressopp.external.transformations*), 70

S

saveLBConf() (*espressopp.integrator.LatticeBoltzmann.espressopp.integrator.LatticeBoltzmann method*), 88
scale_from_matrix() (*in module espressopp.external.transformations*), 71
scale_matrix() (*in module espressopp.external.transformations*), 71
setaxes() (*espressopp.external.transformations.Arcball method*), 61
setBoxL() (*espressopp.bc.OrthorhombicBC.espressopp.bc.OrthorhombicB method*), 56
setBoxL() (*espressopp.bc.SlabBC.espressopp.bc.SlabBC method*), 57
setCentroidThermostat() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setClmassmultiplier() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setCMDparameter() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setConstKinMass() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setconstrain() (*espressopp.external.transformations.Arcball method*), 62
setFixedPairList() (*espressopp.interaction.FENE.espressopp.interaction.FixedPairListFEN method*), 114
setFixedPairList() (*espressopp.interaction.FENECapped.espressopp.interaction.FixedPairL method*), 116
setFixedTripleList() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabul method*), 189
setFixedTupleList() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabul method*), 189
setForce() (*espressopp.integrator.LBInit.espressopp.integrator.LBInit method*), 91
setGamma() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegrator method*), 104
setKTI() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLoc method*), 104
setLBMom() (*espressopp.integrator.LatticeBoltzmann.espressopp.integrat method*), 88
setLennardJones14Interactions() (*in module espressopp.tools.gromacs*), 250
setLennardJonesInteractions() (*in module espressopp.tools.gromacs*), 250
setLennardJonesInteractionsTI() (*in module espressopp.tools.gromacs*), 250
setmStep() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorL method*), 105
setNtrotter() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegrator*

method), 104
setNTrotter() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngularLocal method*), 190
setPILE() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setPILElambda() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setPotential() (*espressopp.interaction.AngularCosineSquaredTripleListAngularCosineSquared method*), 111
setPotential() (*espressopp.interaction.AngularHarmonic.espressopp.interaction.FixedTripleListAngularHarmonic method*), 112
setPotential() (*espressopp.interaction.DihedralHarmonic.FixedQuadrupleListDihedralHarmonicLocal method*), 134
setPotential() (*espressopp.interaction.DihedralHarmonicNC.FixedQuadrupleListDihedralHarmonicNCLocal method*), 135
setPotential() (*espressopp.interaction.FENE.espressopp.interaction.FixedPairListFENE method*), 114
setPotential() (*espressopp.interaction.FENECapped.espressopp.interaction.FixedPairListFENECapped method*), 116
setPotential() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngularLocal method*), 190
setRealKinMass() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 104
setSpeedup() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 105
setSpeedup() (*espressopp.interaction.TabulatedAngular.FixedTripleListPIadressTabulatedAngularLocal method*), 190
setsStep() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 105
setTabulatedInteractions() (*in module espressopp.tools.gromacs*), 250
setTemperature() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 105
setTimeStep() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 105
setVerletList() (*espressopp.integrator.PIAdressIntegrator.PIAdressIntegratorLocal method*), 105
shear_from_matrix() (*in module espressopp.external.transformations*), 71
shear_matrix() (*in module espressopp.external.transformations*), 71
show() (*in module espressopp.tools.timers*), 243
SlabBC() (*espressopp.bc.SlabBC.espressopp.bc method*), 56
SmoothSquareWell (*class in espressopp.interaction.SmoothSquareWell*), 204
SoftCosine (*class in espressopp.interaction.SoftCosine*), 124
startWorkerLoop() (*in module espressopp.pmi*), 228
step (*in module espressopp.integrator.MinimizeEnergy*), 225
StillingerWeberPairTerm (*class in espressopp.interaction.StillingerWeberPairTerm*), 140
StillingerWeberPairTermCapped (*class in espressopp.interaction.StillingerWeberPairTermCapped*), 142
StillingerWeberTripleTerm (*class in espressopp.interaction.StillingerWeberTripleTerm*), 142
stopWorkerLoop() (*in module espressopp.pmi*), 228
SubregionTracking (*class in espressopp.analysis.SubregionTracking*), 53
SuperFixedQuadrupleListDihedralHarmonicNCLocal (*class in espressopp.external.transformations*), 72
sync() (*in module espressopp.pmi*), 228

T

Tabulated (*class in espressopp.interaction.Tabulated*), 180
TabulatedAngular (*class in espressopp.interaction.TabulatedAngular*), 190
TabulatedDihedral (*class in espressopp.interaction.TabulatedDihedral*), 191
TersoffPairTerm (*class in espressopp.interaction.TersoffPairTerm*), 145
to_dict() (*in module espressopp.tools.timers*), 243
toInt3D() (*in module espressopp.Int3D*), 219
toInt3DFromVector() (*in module espressopp.Int3D*), 219
toQuaternion() (*in module espressopp.Quaternion*), 230
toQuaternionFromVector() (*in module espressopp.Quaternion*), 230
toReal3D() (*in module espressopp.Real3D*), 231
toReal3DFromVector() (*in module espressopp.Real3D*), 231
toRealND() (*in module espressopp.RealND*), 231
toRealNDFromVector() (*in module espressopp.RealND*), 231
toTensor() (*in module espressopp.Tensor*), 233
toTensorFromVector() (*in module espressopp.Tensor*), 233
translation_from_matrix() (*in module espressopp.external.transformations*), 73
translation_matrix() (*in module espressopp.external.transformations*), 73

U

UniformOnSphere (*class in espressopp.esutil.UniformOnSphere*), 58
unit_vector() (*in module espressopp.external.transformations*), 73
UnknownParticleProperty, 211

UserError, 226

V

vector_norm() (*in module espressopp.external.transformations*), 73
vector_product() (*in module espressopp.external.transformations*), 74
VSpherePair (*class in espressopp.interaction.VSpherePair*), 201
VSphereSelf (*class in espressopp.interaction.VSphereSelf*), 201

W

warmup() (*in module espressopp.tools.warmup*), 245
writeTabFile() (*in module espressopp.tools.tabulated*), 255
writexyz() (*in module espressopp.tools.DumpConfigurations*), 247

X

xyzfilewrite() (*in module espressopp.tools.DumpConfigurations*), 247

Z

Zero (*class in espressopp.interaction.Zero*), 203