MOLSIM

version 6.4.7, v4.3.2

Documentation from October 30, 2018

Contents

1 Molsim

1.1 Introduction

MOLSIM is a modular MOLecular SIMulation software for simulation of all-atom and coarse-grained model systems with extensive static and dynamic analyses. It can be run in two different **modes**: generating configurations/trajectory and reading configurations/trajectory. The latter one is useful for subsequent analyses.

MOLSIM supports molecular dynamics (MD), Monte Carlo (MC), and Brownian dynamics (BD) simulation **methods**. Simulations can be performed the microcanonical (NVE), canonical (NVT), isobaric (NPT), and grand canonical (µVT) **ensembles** with a number of different boundary **conditions** applied. Initial **configurations** can be generated by random, on lattice, or using user- provided routines, as well as be read from text files or from binary files of previous simulations.

More specifically, several MD integrators are provided and quaternion description is used for rigid- body rotation. Regarding MC, a large selection of different types of trial displacements is available and cluster moves are supported being useful for system with strongly interacting particles. The total number of **time steps**, in case of MD or BD, or **passes** (one attempt to move each particle), in case of MC, are divided into blocks called **macrosteps**. Complete calculations of averages are done and written for each macrostep, and grand averages are given after the simulation. For some quantities fluctuations are also given. The **precision** of grand averages (i) of scalar variables are calculated by extrapolation of block averages to infinite block length and (ii) of distribution functions from the spread of macrostep averages. Simulations can easily be restarted after an interruption or **prolonged**. Binary storage are used for saved positions and velocities, saved intermediate data, and saved data for subequent analyses for maximal accuary.

MOLSIM is able to handle four types of **objects**. They are **atom**, **particle**, **chain**, and **hierarchical structure**, and they are related to each other as follows: a hierarchical structure is built up of one or several chains, a chain of one or several particles and a particle of one or several atoms. Interactions originate from atoms and particles constitute rigid entities. A particle can, e.g., represent a molecule, a monomer in a polymer joined by a bonding potential, or a colloidal particle; a chain a linear polymer; and a hierarchical structure a dendrimer. Identical **objects** belong to the same class of **object type**. Order numbers are assigned to objects and object types as well as to **pairs of objects** and to **pairs of object types**. Whenever objects or object types, say A, B, C, appear ordered as (A, B, C), the order of the corresponding pairs becomes (AA, AB, AC, BB, BC, CC). This ordering of object and object pairs is important for the assignment of values to input variables.

The **nonbonded** pair potentials, operating between pairs of atoms, can be specified either as (i) a sum of A∗/r∗∗n terms with different prefactors and exponents, or by special routines adapted for more complicated potentials, including the possibility of user-provided potential routines. The two- body interactions are calculated through a **neighbor list** technique, and the potentials and forces are evaluated through **lookup tables**. Different routines are used whether (i) charge, (ii) charge and dipole, or (ii) charge, dipole, and induced dipole interactions are present. In the last case, atom charges, atom dipole moments, and anisotropic polarizability tensors are explicitly used in an efficient prediction method to handle the **many-body polarization**. The long-range charge and dipolar interactions are handled by **Ewald** summation, including the Smooth Particle Mesh method, or by a **reaction-field** method.

The **static analysis** comprises energy distribution functions (df), radial and angular distribution functions, and structure factors. Moreover a large selection of df's describing polarization of dipolar fluids and, the structure of water are available. Also a range of df's charactering polymer shape, structure as well as cluster formation analyses are implemented. Finally, excess chemical potential and potential of mean forces can be evaluated. Implemented **dynamic analyses** are mean square displacements, time correlation functions of velocities, orientations, angular velocities, forces, torques, induced dipole moments, and potential energy. It is possible to restrict the static and dynamic analyses to groups of molecules specified by the user.

It is easy to add **user-provided routines** to MOLSIM to enhance the capability of it. They comprise additional (i) nonbonded potentials, (ii) preparations of start configuration, (iii) selections of data to be dumped, (iv) group divisions, (v) static analyses, and (iv) protocols of exporting coordinates and other information for later image rendering. The user-provided routines are collected in a special file and do not belong to the supported software. Existing input variables control the calls of these routines.

The specification of **input variables** characterizing the model and controlling the calculation is given in an input file. That makes it easy to make sequential calculations with only minor differences in the specification. Both names and values of the variables are given in input file using the Namelist protocol, making it easy to read and prepare input files. Blocks of input data are collected into separate groups.

Text books on molecular simulation:

- Allen, M. P.; Tildelsley, D. J. 'Computer Simulation of Liquids'; Clarendon Press: Oxford, 1987 (1989 in paperback).
- Frenkel D. and Smit, B. 'Understanding Molecular Simulation: From Algorithms to Applications'; Academic Press: San Diego, 1996.
- Rapaport, D. C. 'The Art of Molecular Dynamics Simulation'; Cambridge University Press: Cambridge, 1995.
- Field, M. J. 'A practical introduction to the simulation of molecular systems'; Cambridge University Press: Cambridge, 1999.

1.2 Installation

Obtaining the Code

There are two possibilities of how to obtain the code. You can either simply download the zip of the code, or clone the whole repository.

Downloading the Zip

Download the zip from `here` and save it in the directory of your choice. Afterwards, navigate to that directory and extract the source code with

```
unzip <name of the zip file>
```

You might want to rename the directory to some more reasonable name.

Clone the Molsim repository

Just run

```
git clone https://github.com/joakimstenhammar/molsim.git
```

If you `set up your ssh key` at `GitHub` you can run

```
git clone git@github.com:joakimstenhammar/molsim.git
```

Installation of Molsim

Navigate into the Molsim directroy and run the configure script. This will check some dependencies. Molsim requires FFTW 3.3.4. In can be install automatically within the configure script (Note: This might take some time). The configure script will also ask you for a version name. This version name will be appended to the executables of molsim (use `molsim_ser.ver` instead of `molsim_ser`). Leave it blank for no special version name.

```
cd Molsim
./configure.sh
```

The configure script tries to locate the FFTW libary. If you want to customize the path of the libary, modify the `Src/make.fftwpath` file. Additionally it will select which compiler to use. To costumize the compiler which is to be used change the `Src/make.arch` file.

Now go to the Src directory and make Molsim:

```
cd Molsim/Src
```

and compile the code with

```
make all
```

Alternatively you can compile only the serial code with `make ser`, only the parallel code with `make par`. You might be able to speed up the compilation, by letting the compilation run in parallel (add the option `-j <number of cores>` to the make call e.g. `make -j 4 all`).

To install Molsim run

```
make install
```

If you want to uninstall Molsim run

```
make uninstall
```

After completion, (i) additional files have been created in the source directory, (ii) a subdirectory Bin has been created and contains the executables molsim_ser.exe and molsim_par.exe , and (iii) the files molsim_ser and molsim_par have been copied from the source directory to `$HOME/bin`.

Note, the environmental variable PATH has to contain `$HOME/bin` for the software to function (usually this is handled by the configure script)

The installation of the software is now completed.

You can check the version of molsim by passing either the `-v`, `-V` or `--version` to molsim, like

```
molsim_ser --version
```

1.3 Getting Started

This chapter describes how to execute the software.

– Create a working directory where input and result files will be residing

This directory will in the following be referred to as the work directory. The location of the work directory is arbitrary

– Copy the input file test.in to the work directory

This input file could, e.g., be retrieved from `www.physchem.lu.se/sm`. At this moment we will not consider the content and the meaning of the variables in the input file. The input variables are the subject of the next chapter.

– Type 'molsim_ser test'

After the execution, the files test.out and test.list (among others) should now exist in the working directory.

The content of the output file test.out , should be essentially the same as that obtained by execution trough `www.physchem.lu.se/sm`. If not, probably some error has occurred in the transfer of the code or during the installation.

The test calculations are normally short. The results of them are normally not applicable for the equilibrium situation, but are nevertheless useful for testing the installation.

The parallel version of MOLSIM is invoked by typing 'molsim_par' instead of 'molsim_ser'.

If during the installation you gave a version name while running the configure script, the version name will be appended to the executables of molsim (`molsim_ser.ver` instead of `molsim_ser`).

2 Input

All input data read from the file FIN are done with the namelist procedure. Each namelist contains input related variables. Further information on the namelist procedure is given in Appendix A.

The following namelists are available. Compulsory nameslist are checked with X.

file	namelist		variables describing
molsim.F90	nmlSystem	X	general system variables
	nmlScale	X	scaling units
	nmlThermoInteg		thermodynamic integration
	nmlDist		distribution functions
particle.F90	nmlParticle	X	particle variables
	nmlCopolymerSequence		describe the sequence of copolymers
	nmlNetworkConfiguration		describe network configuration
	nmlRepeating		define the repeating block structure of copolymers
potential.F90	nmlPotential	X	potentials and forces
	nmlPotentialChain		bond and angle potential
	nmlPotentialExternal		external potential
	nmlPolarizationIter		many-body polarization calculation
coordinate.F90	nmlSetConfiguration	X	initial start configuration
md.F90	nmlMD		molecular dynamics simulation
mc.F90	nmlMC		Monte Carlo simulation
	nmlMCAll		Monte Carlo (all) simulation
	nmlMCWeight		applying weights for pmf calculations
	nmlUmbrella		umbrella potential sampling
	nmlMCPmf		direct pmf calculation using updated weights
sso.F90	nmlSPartSSO		optimization of the single particle move
bd.F90	nmlBD		Brownian dynamics simulation
nlist.F90	nmlIntList		calculation of list of nonbonded pairs
dump.F90	nmlDump		dumping/reading of data
group.F90	nmlGroup		dividing particles into groups
static.F90	nmlStatic		call of static analysis routines
	nmlSPDF		single particle distribution functions
	nmlRDF		radial distribution functions
	nmlRDFChain		radial distribution functions for chains
	nmlRDFSph		projected radial distribution functions
	nmlRDFCond		conditional radial distribution functions
	nmlG3Dist		normalized triplet correlation functions
	nmlSF		structure factors
	nmlScatIntens		scattering intensities

file	namelist		variables describing
	nmlAngDF		angular distribution functions
	nmlAngExtDF		angular distribution functions wrt external frame
	nmlOriDipDF		orientation/dipole distribution functions
	nmlRadAngDF		radial-angular 2d distribution functions
	nmlKirkwoodgk		Kirkwood's Gk factor
	nmlOriPolDF		orientaitional polarization distribution functions
	nmlNNHB		neighbors and hydrogen bonds
	nmlNNDF		nearest neighbor distribution functions
	nmlChainDF		chain distribution functions
	nmlChainTypeDF		chain type distribution functions
	nmlChainTypeExtDF		chain type distribution functions, external frame
	nmlCBPC		probability of bead-particle contact
	nmlLoopTailTrain		loop, tail, and train characteristics
	nmlCluster		cluster size distribution functions
	nmlMultipoleDF		electrostatic multipole moment distribution functions
	nmlEnergyDF		energy distribution functions
	nmlWidom1		chemical potentials using Widom's method
	nmlWidom2		chemical potentials using Widom's method
	nmlMeanForce1		mean force between two particles
	nmlMeanForce2		mean force between two particles
	nmlPotMeanForce		potential of mean force between two particles
	nmlSurfaceArea		surface area exposed by all particles of one type
	nmlTrajectory		write trajectory on unit FLIST
	nmlSubStructureDF		distribution function of substructures
	nmlNetworkDF		network distribution functions
	nmlNetworkRadialDF		radial network distribution functions
dynamic.F90	nmlDynamic		call of static analysis routines
	nmlMSD		mean square displacement
	nmlOriXTCF, nmlOriYTCF and nmlOriZTCF		orientational tcf of particle x'/y'/z'-axis
	nmlLinVelTCF and nmlAngVelTCF		linear / angular velocity tcf
	nmlForTCF and nmlTorTCF		force and torque tcf
	nmlIDMTCF		induced dipole moment tcf
	nmlUtotTCF		energy tcf
image.F90	nmlImage		call of image data writing routines
	nmlVRML		generation of VRML coordinate files
	nmlVTF		generation of VTF coordinate files
mixed.F90	nmlMixed		general Molmix variables
	nmlMixed1		generation of potential energy curves

file	namelist		variables describing
	nmlMixed2		calculation of potential energies on a lattice
	nmlMixed3		calculation of global potential energy minimum
	nmlMixed4		generation of random coordinates
	nmlMixed5		calculation of second virial coefficients
	nmlMixed6		calculation of orientational averaged potential energy
moluser.F90	nmlComplexation		analysis of interparticle complexation
	nmlComplexDist		calculation of complexation distribution functions

The following subsections contain all input variables that are used for reading data from file FIN. The variables are grouped together and listed below their namelist name. The first line contains the name, the type, the dimension, and, if any, the default value of the variable. The following lines briefly explain the use of the variable. If the variable only can attain a limited number of values, these are listed. If description about writing data is given, the output is made on file FOUT (if nothing else is stated).

A practical point. Often there is a need of reading a variable conditionally, i.e., if some condition is true the variable is assigned a value and used later. If the condition is false, the presence of the variable in the input file does no harm; thus it has not to be deleted. In the latter case the input file contains redundant data.

2.1 nmlSystem

The namelist nmlSystem contains variables that describe the main features of the simulation and flags that controls optionally calls of routines as well as output

- Variables
  - txtitle
  - txmethod
  - txmode
  - txensemb
  - txbc
  - txstart
  - txuser
  - boxlen
  - cellside
  - sphrad
  - ellrad
  - cylrad
  - cyllen

- lenscl
- temp
- prsr
- nstep1
- nstep2
- iseed
- luseXYseed
- ixseed
- iyseed
- maxcpu
- lcont
- laver
- lti
- ldist
- ldump
- lgroup
- lstatic
- ldynamic
- limage
- ltime
- itest
- ipart
- iatom
- iaver
- ishow
- iplot
- ilist
- ltrace
- lblockaver

txtitle

`character(90)`

- User-provided title.

txmethod

`character(5)`

- '`md`': Molecular dynamic simulation. Further specification is given in namelist nmlMD.
- '`mc`': Monte Carlo simulation. Further specification is given in namelist nmlMC.
- '`mcall`': Monte Carlo with simultaneous movement of all particles. Further specification is given in namelist nmlMCAll.
- 'bd': Brownian dynamic simulation (configuration space). Further specificati on is given in namelist nmlBD.

txmode

`character(10)` **default:** '`simulation`'

- '`simulation`': Simulation and analyses. Further specification of method, ensemble, boundary conditions, and initial configuration are specified by variables txmethod, txensemb, txbc, and txstart. On a top level, analyses are controlled by lgroup , lstatic , and ldynamic.
- '`analysis`': Reading of dump data from DUMP files and analyses. On a top level, analyses are controlled by lgroup , lstatic , and ldynamic.
- '`mixed`': Mixed activities controlled by namelist nmlMixed.

txensemb

`character(3)`

- `nve`: Microcanonical ensemble (only MD). This option should also be used for MD with temperature and/or volume scaling. Further specification is given in namelist nmlMD.
- '`ntv`': Canonical ensemble (only MC or BD).
- '`nvt`': Canonical ensemble (only MC or BD).
- 'npt': Isothermal-isobaric ensemble (only MC).
- '`ntp`': Isothermal-isobaric ensemble (only MC).
- '`mvt`': Grand canonical ensemble (only MC). Still not fully tested.
- '`mtv`': Grand canonical ensemble (only MC). Still not fully tested.

txbc

`character(3)`

- '`xyz`': Periodical boundary condition, x, y, and z-directions.
- '`xy`': Periodical boundary condition, x and y-directions.
- '`z`': Periodical boundary condition, z-direction.
- '`rd`': Periodical boundary condition, rhombic dodecahadral.
- '`to`': Periodical boundary condition, truncated octahedral.
- '`sph`': Hard sphere boundary (only MC).
- '`cyl`': Hard cylinder boundary (only MC).
- '`ell`': Hard ellipsoidal boundary (only MC).

txstart

`character(8)`

- '`setconf`' : Generation of a start configuration and accumulation variables are set to zero. This option is used to obtain a start configuration that should be equilibrated. Further specification is given in namelist nmlSetConfiguration.
- '`readfin`' : Read start configuration from file FIN and accumulation variables are set to zero. The format is(ro(1:3,ip),ori(1:3,1:3,ip),ip = 1,np), i.e., the same as the output of particle coordinates on FOUT.
- '`zero`' : Start configuration is read from FCNF and accumulation variables are set to zero. This option is used to start of a production run composed of nstep1$*$nstep2 steps/passes.
- '`continue`' : Start configuration and accumulated averages are read from FCNF. This option is used to continue an equilibration or production run if the execution was aborted due to exceeded time limit, system malfunction etc. To continue, change start to 'continue' and resubmit the job. This option may also be used to extend a completed production run. Then increase nstep1 to the new total number of macrosteps, change start to 'continue', and resubmit the job. The simulation will continue from the last run to a new total nstep1$*$ nstep2 steps/passes.

txuser

`character(80)`

- Character string for engaging user-provided code in the MOLSIM core.

boxlen

`real(1:3)`

- Box length in x-, y-, and z-directions.

cellside

`real`

- Side length of a rhobic dodecaheron (only txbc='rd') and truncated octahedron cell (only txbc='to').

sphrad

`real`

- Spherical cell radius (only txbc='sph').

ellrad

`real`(1:3)

- Ellipsoidal cell radii (only txbc='ell').

cylrad

`real`

- Cylindrical cell radius (only txbc='cyl').

cyllen

`real`

- Cylindrical cell length (only txbc='cyl').

lenscl

`real` **default:** `1.0`

- Scaling constant which scales box, and particle coordinates (only txstart='zero'). Useful for creating a system similar to a previous one, but with different box lengths.

temp

`real`

- System temperature (only txensemb='npt' or 'nvt'). Desired temperature (only txensemb='nve' ensemble and temperature scaling). Temp is also used if the velocities are set according to a Maxwell distribution.

prsr

`real`

- Desired pressure (only txensemb='npt' or 'nve' and with volume scaling).

nstep1

`integer`

- Number of macrosteps. The total number of steps/passes are nstep1$*$nstep2. After the simulation grand averages are calculated and written for the nstep1$*$nstep2 steps/passes.

nstep2

`integer`

- Number of steps (only MD or BD), or number of passes (only MC or MCALL) per macrostep. One pass is one attempt to move each particle in average (only MC) or one attempt to simultaneously move all particles (only MCALL). Averages are calculated and written for each set of nstep2 steps/passes. After each set the FCNF file is updated with new coordinates and accumulated averages.

iseed

`integer`

- Seed of random number generator. iseed $>$ 0 is required.

luseXYseed

`logical` **default:** `false`

- `.true.`: ixseed and iyseed from the input file are used.
- `.false.`: Nothing.

ixseed

`integer`

- Initial ix used for the random number generator.

iyseed

`integer`

- Initial iy used for the random number generator.

maxcpu

`integer` **default:** `0`

- `0`: Nothing.
- $<$`0`: The program stops before starting next macrostep if the total cpu time including the next one exceeds maxcpu (in seconds). The cpu time for the last macrostep is used as the estimate of the cpu time for the next macrostep. This is handy if batch queue installation is used. In such cases, if the job exceeds the maximum cpu time for the batch job, the job might stop abruptly (depending on system installation) without executing the remaining commands in the flink command file and possibly corrupting the file FCNF. If so, the entirely job is lost. The variable maxcpu allows the program to stop itself and promptly send the FCNF file which then can be used to continue the simulation by using the option txstart='continue'.

lcont

`logical` **default:** `.false.`

- Quantities which use is primary to check that the simulation is properly advancing may be calculated (by driver ControlAver). The quantities are averaged over particle types and they are average square force and torque (per particle), linear and angular moments (per particle), as well as translational and rotational temperatures (only MD). Fraction accepted and rejected attempts to move (only MC). Mean square displacement per step/pass. Orientational order. Defined as the scalar product of the direction of a molecular axis and its direction at start. The value is one for a perfect alignment and approximately zero for a fluid. Useful for monitoring the relaxation of an initial lattice start. Position and orientational means. The position of the center of mass denoted $\langle r0 \rangle$, and the projection of the molecular axes on the box axes denoted $<$`x'`$>$, $<$`y'`$>$, and $<$`z'`$>$.
- `.true.`: Control quantities are calculated.
- `.false.`: No calculation of control quantities.

laver

`logical` **default:** `.false.`

- Thermodynamic averages and their precision as well as fluctuations and their precision may be calculated (by routine MainAver). Consider the quantity Q. The precision its average $\langle Q \rangle$ and fluctuation $\sqrt{\langle Q^2 \rangle - \langle Q \rangle^2}$, both given as one standard deviation are evaluated by block averaging with variable block length and extrapolation to infinite block length. The quantities considered are:

1. Total energy (MD)
2. Kinetic energy (MD)
3. Potential energy, total
4. Potential energy, total two-body contribution (only txelec='dip' or 'pol' the electrostatic interaction is excluded) Potential energy, partial two-body contributions (only txelec='dip' or 'pol' the electrostatic interaction is excluded)
5. Potential energy from the reciprocal space (only txelec='charge' .and. lewald)
6. Electrostatic energy (only txelec='dip' or 'pol'), including the reciprocal space (only lewald)

7. Polarization energy (only txelec='pol'), including the reciprocal space (only lewald)
8. Enthalpy
9. Heat capacity
10. Temperature
11. Pressure
12. Volume
13. Induced dipole moment: total (only txelec='pol')
14. Induced dipole moment: particle (only txelec='pol')
15. Induced dipole moment: atom (only txelec='pol')

- `.true.`: Thermodynamic averages are calculated.
- `.false.`: No calculation of thermodynamic averages.

lti

`logical` **default:** `.false.`

- `.true.`: Handle thermodynamic integration (by routine ThermoInteg). Further specification is given in namelist nmlThermoInteg.
- `.false.`: No thermodynamic integration.

ldist

`logical` **default:** `.false.`

- `.true.` Distribution functions are calculated (by routine DistFunc). Further specification is given in namelist nmlDist.
- `.false.` No calculation of distribution functions.

ldump

`logical` **default:** `.false.`

- `.true.` Variables are dumped or read. Further specification is given in namelist nmlDump.
- `.false.` No dumping/readiing

lgroup

`logical` **default:** `.false.`

- `.true.` Particles are divided into groups. This division is used in routine static and optionally by moldyn. Further specification is given in namelist nmlGroup.
- `.false.` No division into groups

lstatic

`logical` **default:** `.false.`

- `.true.` Makes it possible to call several routines calculating static properties. Further specification is given in namelist nmlStatic (require lgroup=.true.).
- `.false.` No static analysis.

ldynamic

`logical` **default:** `.false.`

- `.true.` Makes it possible to call several routines calculating dynamic properties. Further specification is given in namelist nmlDynamic (require lgroup=.true.).
- `.false.` No dynamic analysis.

limage

`logical` **default:** `.false.`

- `.true.` Makes it possible to call several routines preparing files for generating images. Further specification is given in namelist nmlImage.
- `.false.` No image analysis.

ltime

`logical` **default:** `.true.`

- `.true.`: Timing statistics are carried out.
- `.false.`: No timing statistics.

itest

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- '1': Energy, pressure, particle and atom coordinates, forces, torques, dipole moments, polarizability tensor, linear and angular accelerations, linear and angular velocities, and particle distance matrix are written after each step/configuration.
- 3: Intermediate energies (energy.F90).
- 4: Neighbour list (nlist.F90).

ipart

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Initial and final particle coordinates and velocities are written for every ipart particle.

iatom

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Initial atomic coordinates are written for every iatom atom.

iaver

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Cumulative (for the macrostep) thermodynamic averages are written for every iaver steps/passes.

ishow

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Calculated functions are listed at every ishow bin. Compact list.

iplot

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Calculated functions are plotted.

ilist

`integer` **default:** `0`

- `0`: Nothing.
- $>$`0`: Calculated functions are listed at every ilist bin on file FLIST. Extended list.

ltrace

`logical` **default:** `.false.`

- `.true.`: Information upon entering and exiting subroutines on three levels are written on unit 40 for the master and unit 41 for slaves.
- `.false.`: Nothing.

lblockaver

`logical` **default:** `.false.`

- `.true.`: Intermediate data concerning the block averaging and its extrapolation to infinite block length are written on file blockaver.data.
- `.false`: Nothing.

2.2 nmlScale

The namelist nmlScale contains scaling variables that describe the relation between SI units and units used. The relation between the numerical value of quantity Q in SI units and units used is Q(SI) = S$*$Q, where S is the scaling factor. Below, the SI units are given in parentheses after the default scaling factors. The output list gives the scaling factor for some other derived units presented in the output. By a suitable choice of assignments, input variables could be interpreted as reduced variable. In particular, sclene = RT (ca. 2500 at ambient temperature), where R is the gas constant and T the provided temperature, energies are given units of kT.

- Variables:
  - scllen
  - sclmas
  - scltem
  - scltim
  - sclene
  - sclhca
  - sclpre
  - scldif
  - sclang

scllen

`real` **default:** `1.0d-10` m

- Scaling factor for length.

sclmas

`real` **default:** `1.0d-3` kg/mol

- Scaling factor for mass.

scltem

`real` **default:** `1.0` K

- Scaling factor for temperature

scltim

`real` **default:** `1.0d-12` s

- Scaling factor for time.

sclene

`real` **default:** `1.0d+3`J/mol

- Scaling factor for energy.

sclhca

`real` **default:** `1.0` J/(mol$*$K)

- Scaling factor for heat capacity.

sclpre

`real` **default:** `1.0d+6` Pa

- Scaling factor for pressure.

scldif

`real` **default:** `1.0d-9` m^2/s

- Scaling factor for diffusion coefficient.

sclang

`real` **default:** `PI/180` 1/rad

- Scaling factor for angle.

2.3 nmlThermoInteg

The namelist nmlThermoInteg contains variables that control thermodynamic integration. Presently, (i) hard-sphere, (ii) Coulomb, (iii) bond, and/or (iv) angular potentials can be changed through the coupling parameter.

- Variables:
  - lambda
  - powercharge
  - powerkbond
  - powerkangle

lambda

`real`

- Coupling parameter.

powercharge

`real` **default:** `1.0`

- Power of the coupling parameter scaling the charge.

powerkbond

`real` **default:** `9.0`

- Power of the coupling parameter scaling kbond.

powerkangle

`real` **default:** `6.0`

- Power of the coupling parameter scaling kangle.

type	label	distribution functions

2.4 nmlDist

The namelist nmlDist contains variables that control the calculation of distribution functions during the simulation. Any combination of the nine types of distribution functions listed below may be selected through vtype%l.

type	label	distribution functions
1	totu	total potential energy
2	paru	partial potential energy (only two-body contributions)
3	bindu	binding energy (only two-body contributions)
4	pairu	pair energy (only two-body contributions)
5	rdf	radial d.f., particle-particle (center of mass)
6	rdf	radial d.f., atom(mass $>$ maslim)-atom(mass $>$ maslim)
7	idm	induced dipole moment: total
8	idm	induced dipole moment: particle
9	idm	induced dipole moment: atom (polarization $>$ pollim)
10	zdens	z-density distribution function: particle

The uncertainty of a given value of the distribution functions is given as one standard deviation, and it is evaluated by block averaging where one macrostep constitutes one block.

- Variables:
  - vtype
  - idist
  - rcutdist
  - maslim
  - pollim
  - itestdist

vtype

`static1D_var(logical, real, real, integer, logical, character, real)(1:10)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.

idist

`integer` **default:** `10`

- Interval of sampling the distribution functions.

rcutdist

`real` **default:** rcut+racom(ipt)+racom(jpt)

- Cutoff distance for evaluating the distribution functions based on the center-ofmass of the particle. The value could be smaller but should not exceed the default one.

maslim

`real` **default:** `1d-4`

- Lower mass limit for calculating atom-atom rdf.

pollim

`real` **default:** `1d-4`

- Lower polarizability limit for calculating induced dipole moment distribution functions for atoms.

itestdist

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `1`: Forces and virial (routine DistFunc).

2.5 nmlParticle

The namelist nmlParticle contains variables that describe the number of particles, their geometries, masses, etc. Examples of namelist nmlParticle describing different types of particles are given in Appendix B.

- Variables:
  - txelec
  - lclink
  - lmultigraft
  - lhierarchical
  - maxnbondcl
  - ngen
  - ictgen
  - nbranch
  - ibranchpbeg

- ibranchpinc
- nnwt
- nct
- txct
- ncct
- npptct
- txcopolymer
- nblockict
- npt
- txpt
- nppt
- natpt
- txat
- massat
- radat
- zat
- zatalpha
- sigat
- epsat
- latweakcharge
- jatweakcharge
- pK
- pH
- naatpt
- txaat
- rain
- dipain
- polain
- lintsite
- raintin
- lradatbox
- itestpart

txelec

`character(11)` **default:** '`charge`'

- Describes the complexity of the electrostatic interactions. Available sources of electrostatic interactions are: (permanent) charges, weak charges, dipoles, and induced dipoles. Charges, dipole moments, and polarizability tensors employed are specified in namelist nmlParticle.
- '`charge`': Atoms possessing charges only.
- '`weakcharge`': Atoms possessing charges and weak charges only. Limited to hard-sphere monoatomic particles.
- '`dip`': Atoms possessing charges and static dipoles only. MD for all boundary conditions, and MC for all except Ewald summation.
- '`pol`': Atoms possessing charges, static dipoles, and induced dipoles only. MD or MCALL. Ref. Chem. Phys. 191, 195 (1995). Charges, dipole moments, and polarizability tensors employed are specified in namelist nmlParticle.
- '`dipsph`': Atoms possessing charges and/or dipoles in a spherical geometry, no neighbour list; special energy routines.
- '`dieldis`': Atoms possessing charges with the presence of a planar or spherical dielectric discontinuity.

lclink

`logical` **default:** `.false.`

- `.true.`: Enabling cross-links between particles and chains or between chains. Diamond- like network and bottle-brushes are currently supported.
- `.false.`: No cross-links.

lmultigraft

`logical` **default:** `.false.`

- `.true.`: Enabling multigrafted polymers made of chains.
- `.false.`: No multigrafting.

lhierarchical

`logical` **default:** `.false.`

- `.true.`: Enabling hierarchical structures made of chains.
- `.false.`: No hierarchical structures.

maxnbondcl

`integer(1:npt)` **default:** npt∗4

- Maximum number of cross-links of a particle (only txsetconf='hierarchical').

ngen

`integer` **default:** 1

- Number of generations of hierarchical structure (only lhierarchical=.true. and txsetconf ='hierarchical').

ictgen

`integer(1:ngen)` **default:** 1

- Generation number -> chain type (only txsetconf='hierarchical').

nbranch

`integer(0:ngen -1)` **default:** `0`

- Number of branches of a branching point of generation igen (only lhierarchical=.true. and txsetconf = 'hierarchical').

ibranchpbeg

`integer(0:ngen -1)` **default:** `0`

- Chain segment of first branching point of generation igen (only lhierarchical=.true. and txsetconf = 'hierarchical').

ibranchpinc

`integer(0:ngen -1)` **default:** 1

- Segment increment between branching point of generation igen (only lhierarchical=.true. and txsetconf = 'hierarchical').

nnwt

`integer` **default:** `0`

- Number of network types (only txsetconf = 'network').

nct

`integer` **default:** `0`

- Number of chain types.

txct

`character(10)`(1:nct)

- Text label for each chain type.

ncct

`integer`(1:nct)

- Number of chains of each chain type.

npptct

`integer`(1:npt ,1:nct)

- npptct (ipt,ict) is the number of particles of type ipt belonging to chain of type ict. Note, either non or all particles of a given type has to belong to chains, in the latter case sum(ncct (1:nct)∗npptct (ipt,1:nct)=nppt (ipt) is required.

txcopolymer

`character(30)`(1:nct) **default:** nct ∗'`block`'

- '`block`': Block copolymer.
- '`regular`': Regular copolymer (alternating as possible).
- '`sequence`': Copolymer with highly specific monomer distribution (the control is given in nmlCopolymerSequence).
- '`repeating`': Copolymers with a defined repeating block structure (the control is given in nmlRepeating).
- '`random`': Random Copolymer.

nblockict

`integer`(1:nct) **default:** nct∗`0`

- Number of blocks in each repeating of a chain of a chain type.

npt

`integer`

- Number of particle types.

txpt

`character(10)(1:npt)`

- Text label for each particle type.

nppt

`integer(1:npt)`

- Number of particles of each particle type.

natpt

`integer(1:npt)`

- Number of atom types of each particle type.

txat

`character(10)(1:nat)`

- Text label for each atom type (global atom type list).

massat

`real(1:nat)`

- Mass of atom type.

radat

`real(1:nat)`

- Hard-sphere radius of atom type. It is used to avoid an unreasonable start configuration (see namelist nmlSetConfiguration) and to check that atoms do not come too close to each other due to potential or program error. The check is performed after each macrostep by routine CheckHS↩ Overlap . The value of radat should be smaller than the van der Waals radius, approximately 75% of it. In the case of a hard-core potential and MC, radat should be equal to the hard-core radius of the atom.

zat

`real`(1:nat) **default:** nat$*$`0.0`

- Charge of atom type. The charge should be given in number of elementary charges.

zatalpha

`real`(1:nat) **default:** `0.0`

- $>$`0.0`: Gaussian charge distribution with width 1/(sqrt(2)$*$zatalpha)
- =`0.0`: Point charge

sigat

`real`(1:nat) **default:** nat$*$`0.0`

- Lennard-Jones σ parameter of atom type; $u(r) = 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6]$.

epsat

`real`(1:nat) **default:** (1:nat)

- Lennard-Jones ϵ parameter of atom type; $u(r) = 4\epsilon[(\sigma/r)^{12} - (\sigma/r)^6]$.

latweakcharge

`logical`(1:nat) **default:** nat$*$`.false.`

- `.true.`: Weak (titrating) charge of atom type (only txelec ='weakcharge').
- `.false.`: No weak charge.

jatweakcharge

`integer`(1:nat) **default:** nat$*$`0`

- type of atom carrying counter charge to weak charge iat (0 means no counter charge)

pK

`real`(1:nat) **default:** nat$*$`0.0`

- pK value of the weak charge.

pH

`real` **default:** `0.0`

- pH of the solution.

naatpt

`integer`(1:nat,1:npt)

- naatpt (ialoc,ipt) denotes the no of atoms of type ialoc (local list) on a particle of type ipt. ialoc runs from 1 to natpt (ipt), the no of atom types of particle type ipt.

txaat

`character(10)`(1:napt,1:npt)

- txaat (ialoc,ipt) is a text label for atom of no ialoc (local list) on particle of type ipt. ialoc runs from 1 to napt(ipt), the no of atoms of particle type ipt.

rain

`real`(1:3,1:napt,1:npt)

- rain (1:3,ialoc,ipt) is the x:y:z-coordinate of atom no ialoc (local list) in a particle of type ipt. ialoc runs from 1 to napt(ipt), the no of atoms of particle type ipt. rain need not necessarily be given in the principle frame axes.

dipain

`real`(1:3,1:napt,1:npt)

- dipain (1:3,ialoc,ipt) is the x:y:z-component of the dipole moment of atom no ialoc (local list) in a particle of type ipt. ialoc runs from 1 to napt(ipt), the no of atoms of particle type ipt. dipain has to be given in the same frame as rain.

polain

`real`(1:napt,1:npt)

- polain (1:6,ialoc,ipt) is the xx:yy:zz:xy:xz:yz-component of the symmetric polarizability of atom no ialoc (local list) in a particle of type ipt. ialoc runs from 1 to napt(ipt), the number of atoms of particle type ipt. polain has to be given in the same frame as rain.

lintsite

`logical` **default:** `.false.`

- `.true.`: Positions of interaction sites are given by raintin.
- `.false.`: Positions of interaction sites are given by rain.

raintin

`real`(1:3,1:napt,1:npt)

- raintin (1:3,ialoc,ipt) is the x:y:z-coordinate of interaction size no ialoc (local list) in a particle of type ipt. ialoc runs from 1 to napt(ipt), the no of atoms of particle type ipt.

lradatbox

`logical` **default:** `.false.`

- `.true.`: Check that atoms including their hard-sphere radii are inside the box.
- `.false.`: No such check.

itestpart

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `10`: Chain pointers

2.6 nmlCopolymerSequence

The namelist nmlCopolymerSequence contains variables that describe the sequence of copolymers.

- Variables:
  - iptsegct

iptsegct

`integer`(maxval(npct(1:nct)),1:nct) **default:** npct∗nct∗`0`

- Particle type ipt of segments iseg of chain type ict

2.7 nmlNetworkConfiguration

The namelist nmlNetworkConfiguration contains variables that describe the number of networks the particles and chains, which form the networks and the the network topology.

- Variables:
  - nnwnwt
  - ncctnwt
  - txnwt
  - txtoponwt
  - iptclnwt

nnwnwt

`integer`(1:nnwt) **default:** nnwt$*$`0`

- Number of networks of network type inwt (only txsetconf = 'network').

ncctnwt

`integer`(1:nct,1:nnwt) **default:** nct $*$nnwt $*$`0`

- Number of chains of chain type ict of network type inwt (only txsetconf = ‘network’).

txnwt

`character(10)`(1:nnwt) **default:** nnwt $*$“network”

- Text label for each network type inwt (only txsetconf = ‘network’).

txtoponwt

`character(30)`(1:nnwt) **default:** nnwt $*$'`default`'

- Controls the network topology to be set (currently only ‘default’ possible and only txsetconf = ‘network’).

iptclnwt

`integer`(1:nnwt) **default:** nnwt$*$`0`

- Type of particles forming cross-links of networks of network type inwt (only txsetconf = 'network').

2.8 nmlRepeating

The namelist nmlRepeating contains variables that define the repeating block structure of copolymers. The copolymers consist of repeating units, each consisting of blocks of one particle type.

- Variables:
  - rep_block_ict

rep_block_ict

`block_type`(1:pt,1:np) **default:** pt∗np∗`0`

- Particle type and number of particles in in each block and chain type.

2.9 nmlPotential

The namelist nmlPotential contains variables that describe the potentials between the particles. Tabulated potential energy and force evaluation, which often gives a fast code by avoiding square root and divisions. Procedure according to Andrea, Swope, and Andersen, JCP 79, 4576 (1983).

- Variables:
  - r2uminin
  - utoltab
  - ftoltab
  - umaxtab
  - fmaxtab
  - rcut
  - txpot
  - npot
  - ipot
  - ucoff
  - relpermitt
  - lscrc
  - scrlen
  - lscrhs
  - lewald
  - txewaldrec
  - iewaldopt
  - uewaldtol
  - ualpha

- ualphared
- ncut
- ncutregion
- lsurf
- lewald2dlc
- order
- nmesh
- lrf
- epsrf
- epsimg
- radimg
- epsi1
- epsi2
- boundaryrad
- lmaxdiel
- lbg
- lljcut
- ljrcut
- ljushift
- lambda_sw
- epsilon_sw
- alpha_sw
- lambda_ramp
- epsilon_ramp
- alpha_ramp
- rad_dep
- rho_dep
- factor_dep
- lellipsoid
- radellipsoid
- aellipsoid
- lsuperball
- radsuperball
- qsuperball

- txmethodsuperball
- nitersuperball
- tolsuperball
- meshdepthsuperball
- dl_damp
- dl_cut
- dr_damp
- dr_cut
- lstatsuperball
- luext
- lmonoatom
- itestpot

r2uminin

`real` **deault:** `0.1`

- Square of the lower end of the tabulated potential.

utoltab

`real` **default:** `10d-5`

- Energy tolerance of the tabulated potential.

ftoltab

`real` **default:** `10d-5`

- Force tolerance of the tabulated potential.

umaxtab

`real` **default:** `2*10d4`

- Energy at which the table is cut off at small separation.

fmaxtab

`real` **default:** `2*10d4`

- Force at which the table is cut off at small separation.

rcut

`real` **default:** `0.0`

Cutoff distance of the potential and forces based on the particle - particle center of mass distance. If rcut=0.0, the cutoff distance sets to

- `(boxlen2(1)**2 + boxlen2(2)**2 + boxlen2(3)**2)**1/2` (only txbc= ' xyz '),
- `(boxlen2(1)**2 + boxlen2(2)**2 + boxlen(3)**2)**1/2` (only txbc= ' xy '),
- `(boxlen(1)**2 + boxlen(2)**2 + boxlen2(3)**2)**1/2` (only txbc= ' z '),
- `2*rsph` (only txbc= ' sph '), or
- `(2*rcyl)**2+lcyl**2)**1/2` (only txbc= ' cyl'),

txpot

`character(20)`

- Text label used for selecting potential of each pair of particle type. The potentials may either be already existing in the program or supplied by the user. The available options are:
- '`(1,6,12)`': Charge plus Lennard-Jones interaction $u_{ij}(r) = q_i q_j / 4\pi\epsilon_0 r + 4\epsilon_{ij}((\sigma_{ij}/r)^{12}(\sigma_{ij}/r)^6)$. The parameters q_i, σ_{ii}, and ϵ_{ii} are given in namelist nmlParticle. Lorentz-Berthelot mixing rules are applied for cross terms.
- '`setx`': Spherical Ewald potential. As the default potential, but the 1/r term is multiplied with erfc(r*ualphared/rcut).
- '`mcy`': The MCY water potential, ref. JCP 64, 1351 (1976).
- '`nemo:xxx`': The two-body part of the Nemo potential 'xxx'. The potential form and the coefficients are read from file molsim.lib. Present installed potentials include nemo:ww (water-water) and nemo:uw(urea-water). Note, the specification of the many-body polarization interaction is given in namelist nmlParticle.
- '`sw`': Square-well potential. Parameters: lambda_sw, epsilon_sw and alpha_sw
- '`ramp`': Ramp potential. Parameters
- '`asakura-oosawa`': Asukura-Oosawa potential
- '`xxx`': Search for user-provided potential labeled 'xxx' called from routine PotentialUser in file moluser.F90.
- If there is no match, the default potential form sum[ucoff (1:npot)/r**ipot(1:npot)] and the variables npot, ipot, and ucoff, which are read in this namelist, are used. If zat /= 0, the Coulomb term need not to be specified.

npot

`integer(1:natat)`

- npot (iatjat) denotes the number of terms of atom type pair iatjat.

ipot

`integer`(1:npot ,1:natat)

- ipot (m,iatjat) denotes the exponent of term m of atom type pair iatjat.

ucoff

`real`(1:npot ,1:natat)

- ucoff (m,iatjat) denotes the coefficient of term m of atom type pair iatjat.

relpermitt

`real` **default:** `1.0`

- Relative permittivity.

lscrc

`logical` **default:** `.false.`

- `.true.`: Screened Coulomb potential. See also lscrhs.
- `.false.`: No screened Coulomb potential.

scrlen

`real`

- Screening length for the screened Coulomb potential.

lscrhs

`logical` **default:** `.false.`

- `.true.`: U(r) = zat (iat) ∗ jat)/r∗exp(-r/scrlen)∗FAC(iat)∗FAC(jat) with FAC(iat) = exp(-r∗radat (iat))/(1+radat (iat)/scrlen)
- `.false.`: U(r) = zat (iat)∗zat (jat)/r∗exp(-r/scrlen). If zat all are zero ucoff (1,iatjat) is used.

lewald

`logical` **default:** `.false.`

- `.true.`: Ewald summation. Implemented for txpot='(1,6,12)', txpot=nemo... with txelec='pol', and default potential. Periodic boundary conditions are required.
- `.false.`: No Ewald summation.

txewaldrec

`character` **default:** '`std`'

- '`std`': standard Ewald summation.
- '`spm`': smooth particle mesh Ewald summation (require installation of the FFTW library from `www.fftw.org`; see makefile). If only the standard Ewald summation is used, calls to subroutines fftw. . . in files energy.F90 and denergy.F90 can be comment away.

iewaldopt

`integer`

- Controls the choice of Ewald truncation analysis. ualphared = ualpha$*$rcut.

- For txewaldrec = 'std':
- `0`: ualphared, rcut, and ncut are used.
- `1`: ualphared and rcut are used to calculate uewaldtol and ncut. [1]
- `2`: uewaldtol and ualpha are used to calculate rcut and ncut. [1]
- `3`: uewaldtol and rcut are used to calculate ualpha and ncut. [1]
- `4`: uewaldtol and ncut are used to calculate ualpha and rcut. [1]
  [1] According to Kolafa and Perram (charges) and Wang and Holm (dipoles).

- For txewaldrec = 'spm':
- `0`: ualphared, rcut, order and nmesh are used.
- `3`: uewaldtol, rcut, order and nmesh are use.

uewaldtol

`real` **default:** `0.0`

- Potential energy tolerance in Ewald summation (see iewaldopt).

ualpha

`real`

- Ewald parameter (see iewaldopt).

ualphared

`real` **default:** `3.0`

- Reduced Ewald parameter used for the Ewald summation (see iewaldopt) and the Spherical Ewald potential.

ncut

`integer`

- Largest number of k-vectors in one direction in the reciprocal space (see iewaldopt).

ncutregion

`character(6)` **default:** '`sphere`'

- '`sphere`': Spherical k-space region.
- '`cube`': Cubic k-space region.

lsurf

`logical` **default:** `.true.`

- `.true.`: Inclusion of the surface term of the Ewald summation (corresponding to ϵ (surrounding) = 1).
- `.false.`: Exclusion of the surface term of the Ewald summation (corresponding to ϵ (surrounding) -> infinity).

lewald2dlc

`logical` **default:** `.false.`

- `.true.`: Correction for making a 3d-periodic system 2d-periodic according to Arnold et al. JCP 2002. Should only be applied with considerable insight on, e.g, the need of making the simulation box longer than the simulated system in the z-direction and the convergence.
- `.false.`: No such correction.

order

`integer` **default:** `5`

- Interpolation order in the reciprocal space (see iewaldopt).

nmesh

`integer` **default:** `48`

- Number of meshes used in the reciprocal space (see iewaldopt).

lrf

`logical` **default:** `.false.`

- `.true.`: Reaction field method (only txelec = 'pol').
- `.false.`: No reaction field method.

epsrf

`real` **default:** `78.0`

- Relative dielectric permittivity of the surrounding beyond rcut for reaction field method. Zero means infinity.

epsimg

`real`

- Relative dielectric permittivity of the surrounding medium (only txelec = 'dipsphimage').

radimg

`real`

- Radius of the surrounding medium (only txelec = 'dipsphimage').

epsi1

`real`

- Dielectric constant of the sphere (only txelec = 'dipdiel').

epsi2

`real`

- Dielectric constant outside the sphere (only txelec = 'dipdiel').

boundaryrad

`real`

- Radius of dielectric boundary (only txelec = 'dipdiel').

lmaxdiel

`integer`

- Truncation of l-sum (only txelec = 'dipdiel').

lbg

`logical`

- `true.`: Volume charge density of inside the sphere neutralizing the system (only txelec = 'dipdiel').

lljcut

`logical` **default:** `.false.`

- `.true.`: Apply a cutoff and shift of the Lennard-Jones potential (only txpot (ipt) = '(1,6,12)').
- `.false.`: No cut and shift.

ljrcut

`real` **default:** `2.0**(1.0/6.0)`

- Cutoff distance in LJ-sigma units.

ljushift

`real` **default:** `1.0`

- Shift of LJ potential in LJ-epsilon units.

lambda_sw

`real` **default:** `1.1`

- End of the square-well potential in units of hard-sphere diameter.

epsilon_sw

`real` **default:** `1.0`

- Depth of the square-well potential.

alpha_sw

`real` **default:** `1.0d-3`

- Regulate the soft slope change at lambda_sw . Higher alpha implies softer change. u(r) = slope(r-r_sw)(0.5+(1/Pi)∗tan(alpha_sw (r-r_sw), slope = -epsilon_sw /r1atat(lambda_sw -1), r_sw = r1atat∗lambda_sw for r1atat $<$ r1atat(lambda_sw -1).

lambda_ramp

`real` **default:** `1.1`

- End of the ramp potential in units of hard-sphere diameter.

epsilon_ramp

`real` **default:** `1.0`

- Depth of the ramp potential.

alpha_ramp

`real` **default:** `1.0d-3`

- Regulate the soft slope change at lambda_ramp. Higher alpha implies softer change. u(r) = slope(r-r_ramp)(0.5+(1/Pi)∗tan(alpha_ramp(r-r_ramp), slope = -epsilon_ramp /r1atat(lambda_ramp -1), r↩ _ramp = r1atat∗lambda_ramp for r1atat $<$ r1atat(lambda_ramp -1).

rad_dep

`real`

- Radius of penetrable hard sphere (Asakura-Oosawa model)

rho_dep

`real`

- number density of penetrable hard sphere (asakura-oosawa model)

factor_dep

`real` **default:** `1.0`

- depletion-thickness factor

lellipsoid

`logical` **default:** `.false.`

- `.true.`: Hard-core ellipsoidal (prolate) particles.
- `.false.`: No hard-core ellipoidal particles.

radellipsoid

`real` **default:** `1.0`

- Radius of degenerated axes.

aellipsoid

`real` **default:** `1.0`

- Aspect ratio: $>$1 prolate and $<$1 oblate.

lsuperball

`logical` **default:** `.false.`

- `.true.`: particles are superballs

radsuperball

`real` **default:** `One` radius of superballs

qsuperball

`real` **default:** `One`

- q parameter of superballs

txmethodsuperball

`character(4)` **default:** “nr‘

- 'nr', 'mesh', 'test'

nitersuperball

`integer` **default:** `10`

- maximal number of nr iterations

tolsuperball

`real` **default:** `1.0d-4`

- tolerance of nr iterations

meshdepthsuperball

`integer` **default:** `4`

- depth of mesh

dl_damp

`real` **default:** `One`

dl_cut

`real` **default:** `1d10`

dr_damp

`real` **default:** `One`

dr_cut

`real` **default:** `1d10`

lstatsuperball

`logical` **default:** `.false.`

- `.true`: engage time statistics

luext

`logical` **default:** `.false.`

- `.true.`: Application of external potentials on the particles.
- `.false.`: No external potential.

lmonoatom

`logical` **default:** `.true.`

- `.true.`: The program checks to see if all particles have only one atom each. If so, 1) sections involving orientational integration, orientational movement, and some orientational output are omitted and 2) simpler and faster potential and force routines are used.
- `.false.`: Forces the program to treat the particles as polyatomic. This option, which may make the program slower, is for maintaining and checking purposes.

itestpot

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- '1': Intermediate potential variables are written
- `2`: Examination of accuracy of two-body potentials
- `3`: Plot of two-body potentials
- `4`: Examination of truncation error of Ewald summation (routine PotTwoBodyTab).

2.10 nmlPotentialChain

The namelist nmlPotentialChain contains variables that describe the potentials involving particles belonging to the same chain. There is a bond potential between two consecutive particles in a chain and an angular potential between three consecutive particles in a chain. The potentials are of the type (k/p)(x-x0)∗∗p.

- Variables:
  - bond
  - angle
  - clink
  - itestpotchain

bond

`bond_var(real, integer, real)(1:`nct`)` **default:** nct∗`bond_var(0.0, 2, 0.0)`

- Force constant, power, and equilibrium separation of bond potential.

angle

`bond_var(real, integer, real)`(1:nct) **default:** nct ∗`bond_var(0.0, 2, 0.0)`

- Force constant, power, and equilibrium separation of angular potential.

clink

`bond_var(real, integer, real)` **default:** `bond_var(0.0, 2, 0.0)`

- Force constant, power, and equilibrium separation of crosslinks.

itestpotchain

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `1`: Bond length table and bond angle table (routines BondLengthTab and BondAngleTab).

2.11 nmlPotentialExternal

The namelist nmlPotentialExternal contains variables that describe the potentials involving atoms and an external potential.

- Variables:
  - txuext

txuext

`character(20)`(1:npt) **default:** npt∗`' '`

- Text label used for selecting external potential. The number of external potential is growing and a number of parameters specifying these parameters are available. See routine IOPotExternal for details.
- '`wall_z`': hard walls at abs(z) = wall_z-ext
- '`sw_wall_zlow`': square-well wall at z = -wall_z-ext
- '`ramp_wall_z`': ramp walls at abs(z) = wall_z-ext
- '`lj_wall_z`': LJ walls at abs(z) = wall_z-ext
- '`lj_wall_z_ts`': truncated and shifted LJ walls at abs(z) = wall_z-ext
- '`lj_wall_z_mod`': heterogeneous LJ walls at abs(z) = wall_z-ext
- '`lj_wall_zlow`': LJ + ramp wall at z = -wall_z-ext
- '`lj_wall_desorb`': truncated LJ + ramp wall at z = -wall_z-ext (Niklas)

- 'estat_field_z': homogeneous electrical field in z-direction
- 'hom_charged_walls': interaction with a charged surface
- 'i_soft_sphere': external, soft, and spherical wall
- 'Gunnar_soft_sphere': external, soft, and spherical wall (Gunnar)
- 'out_hard_ellipsoid': hard ellipsoidal wall
- 'capsid_shell'; hard spherical capsid shell
- 'uniform_shell': hard spherical capsid shell with a uniform surface charge density
- 'sphdielboundary_q': dielectric sphere (multipole expansion)
- 'sphdielboundary_p': dielectric sphere (pairwise interaction)
- 'sphdielboundary': spherical dielectric boundary (pairwise interaction)
- 'core_shell': hard inner and outer spherical walls (Steffi)
- 'insulating_sphere': penetrable uniformly charged sphere (Steffi)
- "hollow_sphere' hollow and charged sphere (Steffi)
- See code for further details.

2.12 nmlPolarizationIter

The namelist nmlPolarizationIter contains variables that control the calculation of the many-body polarization contribution to the potential energy and forces, ref. JPC 94, 1649 (1990).

- Variables:
  - tpolit
  - mpolit
  - npolit
  - ldamping

tpolit

`real` **default:** `10**-4`

- Relative tolerance of the induced dipole moment in the iteration.

mpolit

`integer` **default:** `15`

- Maximum number of iterations.

npolit

`integer` **default:** `5`

- Interval of iteration.

ldamping

`logical` **default:** `.false.`

- `.true.`: Short-distance damping of electrostatic force when calculating induced dipole moment.
- `.false.`: Nothing.

2.13 nmlSetConfiguration

The namelist nmlSetConfiguration contains variables that control the generation of the start configuration and is used only txstart='setconf'.

- Variables:
  - txsetconf
  - nucell
  - rclow
  - rcupp
  - roshift
  - radatset
  - lranori
  - bondscl
  - anglemin
  - iptnode
  - ictstrand
  - rnwt
  - shiftnwt
  - txoriginnwt
  - radlimit
  - ntrydef
  - itestcoordinate

txsetconf

`character(20)`(1:npt)

- Text label used for selecting start coordinates. Existing routines or user provided routines are used.
- If, nmlSetConfiguration ='. . . lattice', particles are placed in the corner with the lowest x, y, and z-coordinate and at the surfaces with the lowest x, y, and zcoordinates, respectively, of a unit cell.
- If nmlSetConfiguration ='. . . random', hard-core overlap test with already existing particles is made. If coordinates could not be generated after 100$*$nppt (ipt) attempts with particles of type ipt, the program is stopped.
- '`origin`': Set one particle in the center of the box.
- '`pclattice`': Generate a primitive cubic lattice (1 particle in the unit cell).
- '`bcclattice`': Generate a body centered cubic lattice (2 particles in the unit cell).
- '`fcclattice`': Generate a face centered cubic lattice (4 particles in the unit cell).
- '`sm2lattice`': Generate a sm2 lattice (4 particles in the unit cell).
- '`diamondlattice`': Generate a cubic diamond lattice (8 particles in the unit cell).
- '`h2olattice`': Generate a cubic lattice, im3m (ice VIII) (2 particles in the unit cell).
- '`n2lattice`': Generate a cubic lattice, pa3 (solid N2) (4 particles in the unit cell).
- '`benzenelattice`': Generate a cubic lattice, pbca (solid benzene) (4 particles in the unit cell).
- '`random`': Generate random positions and orientations
- '`randomfixori`': Gererate random positions and fixed orientations
- '`chainline`': Generate a straight configuration with chain particles on a line (x-dir)
- '`chaincircle`': Generate a straight configuration with chain particles on a circle (xy-plane)
- '`chainrandom`': Generate random positions and orientations for chain particles.
- '`chainrandomintori`': Generate random positions and int. fixed orientations for chain particles.
- '`sphbrushlattice`': Generate a lattice brush on a spherical surface, first segment placed on a lattice.
- '`sphbrushrandom`': Generate a random brush on a spherical surface, first segment randomly placed.
- '`planbrushrandom`': Generate a random brush on a planar surface.
- '`hierarchicallattice`': Generate a hierarchical polymer, first segment placed on a lattice.
- '`hierarchicalrandom`': Generate a hierarchical polymer, first segment randomly placed.
- '`perodicnetwork`': Generate a periodic network (diamond-like containing 8 nodes in a unit cell).
- '`network`': Generate a nonperiodic network (diamond-like containing 8 nodes in a unit cell).
- '`coreshell`': Generate particle positions in a spherical shell.
- '*xxx*': Search for user-provided routine labeled '*xxx*' called from routine SetParticleUser in file moluser.F90.
- If there is no match, the program stops.

nucell

`integer`(1:3,1:npt) **default:** 3npt∗`0`

- Number of unit cells in the x-, y-, and z-direction. The number of particles of type ipt has to be at most nucell (1,:)∗nucell (2,:)∗nucell (3,:)∗npl, where npl is the number of particles in the unit cell (1 if PC, 2 if BCC, and 4 if FCC).

rclow

`real`(1:3,1:npt) **default:** 3npt∗(`-0.5`∗`box(1)`)

- Lower x-, y-, and z-coordinate for the random positions or the set of unit cells. rclow may be unequal for different particle types to allow for a separation of particles of different types.

rcupp

`real`(1:3,1:npt) **default:** 3npt∗(`+0.5`∗`box(1)`)

- Upper x-, y-, and z-coordinate for the random positions or the set of unit cells.

roshift

`real`(1:3,1:npt) **default:** 3npt∗`0.0`

- Shift of the lattice points in a unit cell in fraction of the unit cell length.

radatset

`real`(1:npt) **default:** radat

- Hard-sphere radius used to create the start configuration.

lranori

`logical`(1:npt) **default:** npt∗`.false.`

- `.true.`: Random particle orientation (only txsetconf='origin' and '...lattice').
- `.false.`: Equal particle orientation (x' = x, y' = y, and z' = z).

bondscl

`real`(1:nct) **default:** nct∗`1.0`

- Bond length scaling factor for chains.

anglemin

`real`(1:nct) **default:** nct∗`0.0`

- Smallest allowed angle between consecutive particles in a chain (only nct>0).

iptnode

`integer` **default:** `0`

- Type of particles of nodes (only txsetconf = 'periodicnetwork').

ictstrand

`integer` **default:** `0`

- Type of chain of strands (only txsetconf = 'periodicnetwork').

rnwt

`real`(1:nnwt) **default:** nnwt∗`10.0`

- Cropping sphere radius of networks of network type inwt (only txsetconf = 'network').

shiftnwt

`real`(3,1:nnwt) **default:** 3∗nnwt∗`0.0`

- x-,y- and z-shift of the center of the cropping sphere of network type inwt in units of the unit cell

txoriginnwt

`character(8)`(1:nnwt) **default:** nnwt∗'`random`'

- Selecting center of networks of different types (only txsetconf='network').
- '`origin`': Set one network in the center of the box.
- '`random`': Generate random positions of the centers of the networks.

radlimit

`real`(2)

- Lower and upper radial limit for placing particles (only txsetconf='coreshell').

ntrydef

`integer` **default:** 100

- number of trials of setting the configuration per particle

itestcoordinate

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `1`: Write crosslink data.

2.14 nmlMD

The namelist nmlMD contains variables that control the MD simulation.

- Variables:
  - integ
  - nmlMDtstep
  - tvvite
  - nvvite
  - lsetvel
  - lzeromom
  - tvscl
  - tlscl
  - compre

integ

`character(6)`

- `verlver` Integration according to the velocity form of the Verlet algorithm.
- `gear3` Integration according to a third-order Gear algorithm.
- `gear4` Integration according to a fourth-order Gear algorithm.

nmlMDtstep

`real`

- Time step of the MD integration.

tvvite

`real` **default:** `0.0`

- Factor that determines the initial quaternion velocity for the iteration of the quaternion velocity (only integ='velver'; 0.0 is fine).

nvvite

`integer` **default:** 2

- Number of iterations for the quaternion velocities (only integ='velver'; 2 is preferred).

lsetvel

`logical` **default:** `.true.` (only txstart = 'setconf' .or. 'zero'); `.false.` (only txstart = 'continue')

- `.true.`: Linear and angular velocities are set according to a Maxwell distribution using the temperature temp.
- `.false.`: No set of linear and angular velocities.

lzeromom

`logical` **default:** `.false.`

- `.true.`: Modify linear and angular velocities (only possible) to get zero linear and angular moments while preserving the translational and rotational temperature (only lsetvel=.true.).
- `.false.`: Nothing.

tvscl

`real` **default:** `0.0`

- Time constant for the velocity scaling. The scaling is applied if tvscl>0.0 and is then performed every time step according to velocity(new)=velocity(old)∗ sqrt(1+(tstep/tvscl)∗(temp /t-1)), where t is the instantaneous temperature (Berendsen et al., 1984). tvscl=tstep scales the velocities to give t=temp.

tlscl

`real` **default:** `0.0`

- Time constant for the length scaling. The scaling is applied if tlscl >0.0 and is then performed every time step according to length(new) = length(old)∗(1+x)∗∗(-1/3), x=(tstep/tlscl)∗compre∗(p-prsr), where p is the instantaneously pressure (Berendsen et al., 1984).

compre

`real`

- Compressibility used for the length scaling.

2.15 nmlMC

The namelist nmlMC contains variables that control the MC simulation. Each trial move involves an attempt to move one or severalparticles.

After each macrostep the relative difference of the total potential energy calculated from scratch and that from the updated energy is calculated and written below the heading 'check'. Normally the relative difference is less than $10^(-10)$. This might not hold if 'the linear displacement' $\leq$ drnlist (see IONList).

- Variables:
  - isamp
  - pspart
  - dtran
  - drot
  - lcl1spart
  - lfixzcoord
  - lfixxycoord
  - lshiftzcom
  - lfixchainstartspart
  - ispart
  - pspartcl2
  - txmembcl2
  - radcl2
  - dtrancl2
  - ppivot
  - txpivot
  - drotpivot
  - drotminpivot
  - ipivotrotmode
  - lcl1pivot
  - pchain
  - dtranchain
  - drotchain

- lcl1chain
- pslither
- pbrush
- dtranbrush
- drotbrush
- lcl1brush
- pbrushcl2
- dtranbrushcl2
- drotbrushcl2
- phierarchical
- dtranhierarchical
- pnetwork
- dtrannetwork
- pvol
- dvol
- pnpart
- chempot
- pcharge
- radcl1
- pselectcl1
- pspartsso
- lmcweight
- lautumb
- lmcpmf
- lmcsep
- itestmc

isamp

`integer` **default:** 1

- `0`: Uniform sampling of particles, sequential selection.
- `1`: Uniform sampling of particles, random selection.

pspart

`real`(1:npt) **default:** npt∗`1.0`

- Relative weight of a single-particle move. Further control is given by dtran and drot.

dtran

`real`(1:npt) **default:** npt∗`0.0`

- 0.5∗dtran is the maximal translational displacement of a particle along one box axis. Displacements are made along all box axes. If dtran>0, square region, or if dtran<0, spherical displacement region.

drot

`real`(1:npt) **default:** npt∗`0.0`

- 0.5∗drot is the maximal rotational displacement (in degrees) of one axis. If drot>0, rotation around one randomly selected box axis, or if drot<0, rotation around one randomly selected particle frame axis.

lcl1spart

`logical`(1:npt) **default:** npt∗`.false.`

- `.true.`: Engage a single particle + cluster1 move involving a simultaneous trial move of (i) the selected particle (primary particle) and (ii) some or all particles located within the distance radcl1 from the selected one (secondary particles). The latter particles are selected with the probability pselectcl1.
- `.false.`: No single particle + cluster1 move.

lfixzcoord

`logical`(1:npt) **default:** npt∗`.false.`

- `.true.`: Fixed z-coordinate (restrict the trial move in the xy-plane) of particle. Applicable to spart and spart trial moves.
- `.false.`: No.

lfixxycoord

`logical`(1:npt) **default:** npt∗`.false.`

- `.true.`: Fixed xy-coordinate (restrict the trial move to the z-coordiante) of particle. Applicable to spart and spartcl2 trial moves.
- `.false.`: No.

lshiftzcom

`logical(1:npt)` **default:** npt∗`.false.`

- `.true.`: Shift the z-coordinate of all particles such that the center-of-mass of particles 1 and 2 is as close as possible to z = 0 (only txbc = 'cyl').
- `.false.`: No.

lfixchainstartspart

`logical(1:npt)` **default:** `.false.`

- `.true.`: No trail-displacement of first particle in a chain (for grafted chains).
- `.false.` No such restriction.

ispart

`integer` **default:** `0`

- `0`: Nothing, the normal use.
- $\neq$`0`: For development/special use

pspartcl2

`real(1:npt)` **default:** npt∗`0.0`

- Relative weight of a single particle + cluster2 move involving a simultaneous trial move of (i) the (original) cluster of type 1 as specified above and (ii) other similar clusters connected to the original one. The other clusters of type 1 consist of a central particle of the same type as the selected one and its neighboring particles in analogy with the original cluster. Clusters of type 1 are connected if their center-tocenter separation is directly or indirectly at most radcl2. Connected clusters of type 1 are called clusters of type 2. Further control is given by txmembcl2,dtrancl2 and radcl2.

txmembcl2

`character(3)(1:npt)`

- Selection of particle type of members belonging to clusters of type 2.
- `=ipt`: Search for members only of same particle type as that displaced.
- `=all`: Search for members among all particles.

radcl2

`real(1:npt)` **default:** npt∗`0.0`

- Largest separation between clusters of type 1 for belonging to same cluster of type 2.

dtrancl2

`real`(1:npt) **default:** npt∗`0.0`

- 0.5∗dtrancl2 is the maximal translational displacement of a particle along one box axis. Displacements are made along all box axes. If dtrancl2>0, square region, or if dtrancl2<0, spherical displacement region.

ppivot

`real`(1:nct) **default:** nct∗`0.0`

- Relative weight of end-pivot rotation. The shorter end of the chain is rotated. Further control is given by drotpivot.

txpivot

`character(5)`(1:nct) **default:** nct∗`short`

- `short`: Rotation of the shorter subchain.
- `lower`: Rotation of the subchain containing particles with low numbers.
- `upper`: Rotation of the subchain containing particles with high numbers.

drotpivot

`real`(1:nct) **default:** nct∗`360`

- Rotational displacement parameter of the end-pivot rotation.

drotminpivot

`real`(1:nct) **default:** nct∗`0.0`

- Smallest rotational displacement of the end-pivot rotation.

ipivotrotmode

`integer` **default:** 1

- `1`: Rotation around the bond joining pivot segment and the previous one. Bond angle is preserved.
- `2`: Rotation round the normal to the plane formed by the pivot segments and its two neighbors.
- `3`: Rotation around a random direction.

lcl1pivot

`logical(1:npt)` **default:** npt`*.false.`

- `.true.`: Engage an end-pivot + cluster1 rotation involving a simultaneous trial move of (i) the selected particle(s) (primary particle(s)) and (ii) some or all particles located within the distance radcl1 from the selected one (secondary particles). The latter particles are selected with the probability pselectcl1.
- `.false.`: No end-pivot + cluser1 move.

pchain

`real(1:npt)` **default:** npt`*0.0`

- Relative weight of chain move. Further control is given by dtranchain.

dtranchain

`real(1:npt)` **default:** npt`*0.0`

- Translational displacement parameter of the chain move.

drotchain

`real(1:npt)` **default:** npt`*0.0`

- Rotational displacement parameter of the chain move.

lcl1chain

`logical(1:npt)` **default:** npt`*.false.`

- `.true.`: Engage a chain + cluster1 move involving a simultaneous trial move of (i) the selected particles (primary particles) and (ii) some or all particles located within the distance radcl1 from the selected ones (secondary particles). The latter particles are selected with the probability pselectcl1.
- `.false.`: No chain + cluster1 move.

pslither

`real(1:npt)` **default:** npt`*0.0`

- Relative weight of slithering move. All particles of a selected chain are subjected to a trial displacement (even for a homopolymer).

pbrush

`real`(1:npt) **default:** npt∗`1.0`

- Relative weight of a brush move. Simultaneous move of a particle and its grafted chains. Requires some assumption of chain and bead labeling.

dtranbrush

`real`(1:npt) **default:** npt∗`1.0`

- Translational parameter for brush move.

drotbrush

`real`(1:npt) **default:** npt∗`1.0`

- Rotational parameter for brush move.

lcl1brush

`logical`(1:npt) **default:** npt∗`.false.`

- `.true.`: Engage brush + cluster1 move involving a simultaneous trial move of (i) the selected particles (primary particles) and (ii) some or all particles located within the distance radcl1 from the selected ones (secondary particles). The latter particles are selected with the probability pselectcl1.
- `.false.`: No brush + cluster1 move.

pbrushcl2

`real`(1.npt) **default:** npt∗`0.0`

- Relative weight of a brush + cluster2 move.

dtranbrushcl2

`real`(1:npt) **default:** npt∗`1.0` Translational parameter for brush + cluster2 move.

drotbrushcl2

`real`(1:npt) **default:** npt∗`1.0`

- Rotational parameter for brush + cluster2 move.

phierarchical

`real(1:npt)` **default:** `npt*0.0`

- Relative weight of a hierarchical move.

dtranhierarchical

`real(1:npt)` **default:** `npt*1.0`

- Translational parameter for a hierarchical move.

pnetwork

`real(1:npt)` **default:** `npt*0.0`

- Relative weight of a network move.

dtrannetwork

`real(1:npt)` **default:** `npt*0.0`

- Translational parameter for network move.

pvol

`real(1:npt)` **default:** `npt*0.0`

- Relative weight of volume-change move (only txensemb='npt' or 'ntp').

dvol

`real` **default** `0.0`

- Volume change parameter (only txensemb='npt' or 'ntp').

pnpart

`real(1:npt)` **default:** `npt*0.0`

- Relative weight of 'change of number of particle' move (only txensemb='mvt' or 'mtv').

chempot

`real(1:npt)` **default:** `npt*0.0`

- Chemical potential (only txensemb='mvt' or 'mtv').

pcharge

`real`(1:npt) **default:** npt∗`0.0`

- Relative weight of charge-change move.

radcl1

`real`(1:npt) **default:** npt∗`0.0`

- Largest separation between a primary and secondary particle for which the secondary particle can belong to the cluster.

pselectcl1

`real`(1:npt) **default:** npt∗`0.0`

- Probability that a particle with the separation at most radcl1 from the primary particle is selected as a secondary particle of the cluster. $0 \leq pselectcl1 \leq 1$.

pspartsso

`real`(1:npt) **default:** npt∗`0.0`

- Relative weight of sso move. Further control is given in nmlSPartSSO.

lmcweight

`logical` **default:** `.false.`

- `.true.`: Applying a weighting function for potential of mean force MC simulations. Further specification is given in namelist nmlMCWeight .
- `.false.`: No weighting function applied.

lautumb

`logical` **default:** `.false.`

- `.true.`: Automatic umbrella sampling according to O. Engkvist and G. Karlström. Further specification is given in namelist nmlUmbrella.
- `.false.`: No umbrella sampling.

lmcpmf

`logical` **default:** `.false.`

- `.true`: Direct pmf calculation using updated weights. Further specification is given in namelist nmlMCPmf.
- `.false.`: No direct pmf calculation calculation.

lmcsep

`logical` **default:** `.false.`

- `.true.`: Either local or non-local moves are carried out during one MC-Pass, not both. When only local moves are used the neighbour list is build using drnlist as set in nmlIntlist, if non-local moves are also used drnlist is set to four times the contour length.
- `.false.`: Local and non-local moves are carried out during one MC-Pass.

itestmc

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `1`: Probability data of different moves (routine TestIOMCProb)
- `2`: Data concerning MC trial move (routine TestMCMove)
- `3`: Data concerning volume change (routine TestVolChange, TestNPartCharge1, TestNPart↩ Charge2, TestChargeChange1, TestChargeChange2)

2.16 nmlMCAll

The namelist nmlMCAll contains variables that control the MCALL simulation. Each trial move involves an attempt to move all particles simultaneously.

- Variables:
  - dtranall
  - drotall
  - lautumb

dtranall

`real(1:npt)`

- 0.5∗dtranall is the maximal translational displacement of a particle along one box axis. Displacements are made along all box axes. If dtranall>0, square region, or if dtranall<0, spherical displacement region.

drotall

`real`(1:npt)

- 0.5∗drotall is the maximal rotational displacement (in degrees) of one axis. If drotall>0, rotation around one randomly selected box axis, or if drot<0, rotation around one randomly selected particle frame axis.

lautumb

`logical` **default:** `.false.`

- `.true.`: Automatic umbrella sampling according to O. Engkvist and G. Karlström. Further specification is given in namelist nmlUmbrella.
- `.false.`: No umbrella sampling.

2.17 nmlMCWeight

The namelist nmlMCWeight contains variables that control the application of a weighting function for potential of mean force MC simulations between two particles. Presently, only results from routine Pot↩ MeanForce are reweighted.

- Variables:
  - ipmcw1
  - ipmcw2
  - txpotmcw
  - npolmcw
  - acoeffmcw

ipmcw1

`integer`

- Identity of the particle number 1.

ipmcw2

`integer`

- Identity of the particle number 2.

txpotmcw

`character(11)`

- `polynomial`: Weighting function: a(0) + a(1)∗r + ...
- `exponential`: Weighting function: a(0)∗exp(-a(1)∗(r-a(2))).

npolmcw

`integer` **default:** `0`

- Degree of the polynomial

acoeffmcw

`real`(1:npolmcw) **default:** npolmcw∗`0.0`

- Coefficients of the weighting function.

2.18 nmlUmbrella

The namelist nmlUmbrella contains variables that control the automatic umbrella sampling.

- Variables:
  - typeumb
  - ipumb1
  - ipumb2
  - rminumbrella
  - delumb
  - numbgrid
  - cupdate
  - umbcoord
  - lreadumb

typeumb

Type of umbrella potential

`character(6)` **default:** `''`

- Type of umbrella potential (e.g. particle-particle, particle-wall ...)

ipumb1

`integer` **default:** `0`

- Particle dentifier for particle-particle or atom-atom umbrella sampling.

ipumb2

`integer` **default:** `0`

- Particle dentifier for particle-particle or atom-atom umbrella sampling.

rminumbrella

`real` **default:** `3.0`

- Minimum particle-particle distance for particle-particle or atom-atom umbrella sampling

delumb

`real` **default:** `1.0`

- Distance between two grid points in xumb

numbgrid

`integer` **default:** `0.0`

- Number of grid points in xumb

cupdate

`character(4)` **default:** `''`

- Type of update of the weighting function

umbcoord

`character(1)` **default:** 'r'

- if set, particles ipumb1 and ipumb2 can only move along the coordinate set by umbcoord with fixed orientation

lreadumb

`logical` **default:** `.false.`

- `.true.`: the initial umbrella potential is read from file.
- `.false.`: the potential is calculated using xumb.

2.19 nmlMCPmf

The namelist nmlMCPmf contains variables that control the calculation of potential of mean force by updating weights.

- Variables:
  - iptmcpmf
  - nbinmcpmf
  - rlowmcpmf
  - ruppmcpmf
  - termmcpmf

iptmcpmf

`integer` **default:** `1`

- Type of the two particles for which the potential of mean force should be calculated.

nbinmcpmf

`integer` **default:** `100`

- Number of bins of the potential of mean force.

rlowmcpmf

`rlow` **default:** `0.0`

- Lower limit of sampled potential of mean force.

ruppmcpmf

`real` **default:** `100.0`

- Upper limit of sampled potential of mean force.

termmcpmf

`real`

- Controls the update of weights.

2.20 nmlSPartSSO

The namelist nmlSPartSSO contains variables that handle the sso-functionality. The SSO algorithm allows for on the fly optimization of the displacement parameter of the single particle move during the equilibration run.

- Variables:
  - dtransso
  - maxtransso
  - nstepzero
  - nstepend
  - dtranfac
  - nssobin
  - ltestsso

dtransso

`real(1:npt)` **default:** `npt*1.0`

- Initial displacement parameter to be used during the sso move. The displacement is in a spherical volume with diameter dtransso

maxtransso

`real(1:npt)`

- Maximal allowed displacement parameter the SSO can use. The default value is set according to half the system size, depending on the geometry.

nstepzero

`integer` **default:** `ceiling(sqrt(real(nstep)))`

- number of MC-Passes when the displacement parameter is adapted the first time. See Phys. Procedia 2011, 15, 81-86.

nstepend

`integer` **default:** `max(`nstepzero `, int(0.1*nstep))`

- duration (in MC-Passes) of the last part of the SSO. This, and only this, part will be used to give the final value for the optimal displacement parameter. Do not set it shorter than nstepzero.

dtranfac

`real` **default:** `2`

- Factor by which the displacement parameter is increased if no optimal displacement parameter is found.

nssobin

`integer` **default:** `20`

- number of bins used by the sso move. The more bins the more accurately the displacement parameter can be determined.

ltestsso

`logical` **default:** `.false.`

- `.true.`: Print out intermediate values after each SSO-Part.
- `.false.`: Nothing.

2.21 nmlBD

The namelist nmlBD contains variables that control the BD simulation. The simulation is performed in the configurational space according to Ermark, 1975.

- Variables:
  - nmlBDtstep
  - dcoeff

nmlBDtstep

`real`

- Time step of the BD integration.

dcoeff

`real`

- Isotropic particle self-diffusion coefficient.

2.22 nmlIntList

The namelist nmlIntList contains variables that control the calculation of lists of nonbonded pairs to be considered in two-body energy evaluations.

- Variables:
  - txintlist
  - lvlistllist
  - inlist
  - drnlist
  - facnneigh

txintlist

`character(8)` **default:** '`vlist`'

- '`vlist`': Verlet neighbour lists. Further specification is given by inlist and drnlist.
- '`llist`': Linked lists.

lvlistllist

`logical` **default:** `.false.`

- `.true.`: Use of liked lists to crease Verlet neighbour lists (only txintlist='nlist').

inlist

`integer` **default:** `0`

- `0`: Automatic check whether a new neighbour list should be calculated or not. If the sum of the displacements of any two molecules, since last calculation, is larger than drnlist a new list is generated. The number of neighbour list calculations is written after each macrostep.
- $>$`0`: Interval of calculating a new neighbour list.

drnlist

`real` **default:** `0.0`

- Distance added to the potential cutoff for calculating the neighbour list.

facnneigh

`real` **default:** `2.0`

- `0`: Multiplicative constant of radius for calculation number of interacting particles.

2.23 nmlDump

The namelist nmlDump contains variables that control the interval and variables dumped or read. Dumping is performed when txmode = 'simulation' and reading when txmode = 'analysis'.

- Variables:
  - idump
  - txptdump
  - ldpos
  - ldori
  - ldliv
  - ldanv
  - ldfor
  - ldtor
  - ldidm
  - ldumpuser

idump

`integer` **default:** `10`

- Dumping/reading is performed every idump step/pass. The types of quantities dumped are controlled by the remaining variables. If txstart='continue' the dump files are properly positioned according to the step/configuration saved in file FCNF.

txptdump

`character(20)` **default:** '`all`'

- '`all`': Data for particles of all types are dumped.
- '`xxx`': Data for particles of type ipt is dumped when 'xxx'=txpt (ipt).

ldpos

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle positions.
- `.false.` No dumping/reading of particle positions.

ldori

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle orientations.
- `.false.`: No dumping/reading of particle orientations.

ldliv

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle linear velocities.
- `.false.`: No dumping/reading of particle linear velocities.

ldanv

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle angular velocities.
- `.false.`: No dumping/reading of particle positions.

ldfor

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle forces.
- `.false.`: No dumping/reading of particle forces.

ldtor

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle torques.
- `.false.`: No dumping/reading of particle torques.

ldidm

`logical` **default:** `.false.`

- `.true.`: Dumping/reading of particle induced dipole moments.
- `.false.`: No dumping/reading of induced dipole moments.

ldumpuser

`logical` **default:** `.false.`

- `.true.`: Call of DumpUser, driver for user-provided dumping routines.
- `.false.`: No such call.

2.24 nmlGroup

The namelist nmlGroup contains variables that control the division of particles into reference and field groups. Most single particle quantities calculated in static.F90 are averaged over particles belonging to same reference group. In case of two particle quantities, the field group specifies the particles with which the reference particle interacts or is close to. More details are given below.

- Variables:
  - ref
  - field
  - lwref

ref

`character(20)` **default:** `'type=all'`

- Text label used for selecting procedure of how to divide particles into reference groups. The procedure may either be already existing in the program or supplied by the user. The available options are:
- '`type=all`': Each particle type constitute one group.
- '`networkgenerations`': The different chain generations of networks will be assigned to different reference groups. Requires particle type of the strand particles to be ipt = 2.
- '`type=xxx`': Particles of type xxx (txpt (ipt)='xxx') constitute the only group.
- `xxx`: Search for user-provided section labeled 'xxx' called from routine GroupUser in file moluser.↩ F90.
- If there is no match, the program stops.

field

`character(20)` **default:** '`type=all`'

- Describes the field groups. Same options as for ref.

lwref

`logical` **default:** `.false.`

- `.true.`: Reference group data are written on file FGROUP.
- `.false.`: No writing.

2.25 nmlStatic

The namelist nmlStatic contains variables that control the interval of the analysis and static analysis routines used.

- Variables:
  - istatic
  - lspdf
  - lrdf
  - lrdfchain
  - lrdfsph
  - lg3
  - lrdfcond
  - lsf
  - langdf
  - langextdf
  - loridipdf
  - llsphharaver
  - lradangdf
  - lkirkwoodgk
  - loripoldf
  - lnnhb
  - lnndf
  - lchaindf
  - lchaintypedf
  - lchaintypeextdf

- lcbpc
- lltt
- lcluster
- lzerosecondmoment
- lmultipoledf
- lenergydf
- lwidom1
- lwidom2
- lmeanforce1
- lmeanforce2
- lpotmeanforce
- lsurfacearea
- lcrystalformat
- ltrajectory
- lsubstructuredf
- lnetworkdf
- lnetworkradialdf
- lstaticuser

istatic

`integer` **default:** 1

- Conformation sampling interval. Holds both for txmode = 'simulation' and 'analysis'.

lspdf

`logical` **default:** `.false.`

- `.true.`: Single particle distribution functions are calculated. Further specification is given in namelist nmlSPDF.
- `.false`: No calculation.

lrdf

`logical` **default:** `.false.`

- `.true.`: Radial distribution functions or running coordination numbers or are calculated. Further specification is given in namelist nmlRDF.
- `.false`: No calculation.

lrdfchain

`logical` **default:** `.false.`

- `.true.`: Radial distribution functions or running coordination numbers are calculated between center of masses of chains. Further specification is given in namelist nmlRDFChain.
- `.false`: No calculation.

lrdfsph

`logical` **default:** `.false.`

- `.true.`: Radial distribution functions or running coordination numbers are calculated for particles which positions are projected on a sphere. Further specification is given in namelist nmlRDFSph.
- `.false`: No calculation.

lg3

`logical` **default:** `.false.`

- `.true.`: Normalized triplet correlation functions are calculated. Further specification is given in namelist nmlG3Dist.
- `.false`: No calculation.

lrdfcond

`logical` **default:** `.false.`

- `.true.`: Conditional radial distribution functions are calculated. Futher specification is given in namelist nmlRDFCond.
- `.false.`: No calculation.

lsf

`logical` **default:** `.false.`

- `.true.`: Partial structure factors are calculated. Further specification is given in namelist nmlSF.
- `.false.`: No calculation.

langdf

`logical` **default:** `.false.`

- `.true.`: Angular distribution functions are calculated. Further specification is given in namelist nmlAngDF.
- `.false.`: No calculation.

langextdf

`logical` **default:** `.false.`

- `.true.`: Angular 2d distribution functions with respect to external frame are calculated. Further specification is given in namelist nmlAngExtDF.
- `.false.`: No calculation.

loridipdf

`logical` **default:** `.false.`

- `.true.`: Orientation/dipole distribution functions are calculated. Further specification is given in namelist nmlOriDipDF.
- `.false.`: No calculation.

llsphharaver

`logical` **default:** `.false.`

- `.true.`: Averages of unnormalized spherical harmonics are calculated.
- `.false.`: No calculation.

lradangdf

`logical` **default:** `.false.`

- `.true.`: Radial-angular 2d distribution functions are calculated. Further specification is given in namelist nmlRadAngDF.
- `.false.`: No calculation.

lkirkwoodgk

`logical` **default:** `.false.`

- `.true.`: Kirkwood gk-factors are calculated. Further specification is given in namelist nmlKirkwoodgk.
- `.false.`: No calculation.

loripoldf

`logical` **default:** `.false.`

- `.true.`: Orientation polarization distribution functions are calculated. Further specification is given in namelist nmlOriPolDF.
- `.false.`: No calculation.

lnnhb

`logical` **default:** `.false.`

- `.true.`: Number of nearest neighbours and no of hydrogen bonds are calculated. Further specification is given in namelist nmlNNHB.
- `.false.`: No calculation.

lnndf

`logical` **default:** `.false.`

- `.true.`: Nearest neighbour distribution functions are calculated. Further specification is given in namelist nmlNNDF.
- `.false.`: No calculation.

lchaindf

`logical` **default:** `.false.`

- `.true.`: Chain distribution functions are calculated. Further specification is given in namelist nmlChainDF.
- `.false.`: No calculation.

lchaintypedf

`logical` **default:** `.false.`

- `.true.`: Chain distribution functions are calculated. Further specification is given in namelist nmlChainTypeDF.
- `.false.`: No calculation.

lchaintypeextdf

`logical` **default:** `.false.`

- `.true.`: Chain type distribution functions with respect to the lab frame are calculated. Further specification is given in namelist nmlChainTypeExtDF.
- `.false.`: No calculation.

lcbpc

`logical` **default:** `.false.`

- `.true.`: Probabilities of chain particles to be near particles of another type are calculated. Further specification is given in namelist nmlCBPC.
- `.false.`: No calculation.

lltt

`logical` **default:** `.false.`

- `.true.`: Loop, tail, and train statistics are calculated. Further specification is given in namelist nmlLoopTailTrain.
- `.false.`: No calculation.

lcluster

`logical` **default:** `.false.`

- `.true.`: Cluster size distribution functions are calculated. Further specification is given in namelist nmlCluster.
- `.false.`: No calculation.

lzerosecondmoment

`logical` **default:** `.false.`

- `.true.`: Zero and second moment for an ionic, neutral system is calculated.
- `.false.`: No calculation.

lmultipoledf

`logical` **default:** `.false.`

- `.true.`: Electrostatic multipole moment distribution functions are calculated. Further specification is given in namelists nmlMultipoleDF.
- `.false.`: No preparation.

lenergydf

`logical` **default:** `.false.`

- `.true.`: Energy distribution functions are calculated. Further specification is given in namelist nmlEnergyDF.
- `.false.`: No calculation.

lwidom1

`logical` **default:** `.false.`

- `.true.`: Excess chemical potentials are prepared. Further specification is given in namelists nmlWidom1.
- `.false.`: No preparation.

lwidom2

`logical` **default:** `.false.`

- `.true.`: Excess chemical potentials are prepared. Further specification is given in namelists nmlWidom2.
- `.false.`: No preparation.

lmeanforce1

`logical` **default:** `.false.`

- `.true.`: Mean force between two particles is calculated. Further specification is given in namelist nmlMeanForce1.
- `.false.`: No calculation.

lmeanforce2

`logical` **default:** `.false.`

- `.true.`: Mean force between two particles is calculated. Further specification is given in namelist nmlMeanForce2.
- `.false.`: No calculation.

lpotmeanforce

`logical` **default:** `.false.`

- `.true.`: Potential of mean force between two particles is calculated. Further specification is given in namelist nmlPotMeanForce.
- `.false.`: No calculation.

lsurfacearea

`logical` **default:** `.false.`

- `.true.`: Surface area available around atoms residing in particles of a specified type is calculated. Further specification is given in namelist nmlSurfaceArea.
- `.false.`: No calculation.

lcrystalformat

`logical` **default:** `.false.`

- `.true.`: Write atoms in the crystallographic format stating with atoms belong to molecules closest to the origin. Hard-code limitation of spatial distance and number of particles are in action.
- `.false.`: No calculation.

ltrajectory

`logical` **default:** `.false.`

- `.true.`: Write trajectory on FLIST. Further specification is given in namelist nmlTrajectory.
- `.false.`: No calculation.

lsubstructuredf

`logical` **default:** `.false.`

- `.true.`: Substructures of chains, hierarchical structures and networks are analyses. Further specification is given in namelist nmlSustructureDF.
- `.false.`: No calculation.

lnetworkdf

`logical` **default:** `.false.`

- `.true.`: Network distribution functions are calculated. Further specification is given in nmlNetworkDF.
- `.false.`: No calculation.

lnetworkradialdf

`logical` **default:** `.false.`

- `.true.`: Radial network distribution functions are calculated. Further specification is given in nmlNetworkRadialDF.
- `.false.`: No calculation.

lstaticuser

`logical` **default:** `.false.`

- `.true.`: StaticUser is called and from where user-provided static analysis routines are called in file moluser.F90.
- `.false.`: No call.

2.26 nmlSPDF

The namelist nmlSPDF contains variables that control the calculation of single particle distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	Quantity
1	rrden	Reduced radial number density
2	rren	Reduced running coordination number
3	zden	Number density in the z-direction
4	zden2	Number density in the z-direction (special)
5	z'∗z	Projection of molecular z'-axis on the box z-axis
6	z'∗x	Projection of molecular z'-axis on the box x-axis
7	z'∗y	Projection of molecular z'-axis on the box y-axis
8	opt_d	Radial density projected on the z = 0 plane
9	elpot	radial electrostaic potential evaluated at particle positions
10	m∗E	cos(m(ia)∗E); m(ia) is the dipole moment of atom ia and E an external field

- Variables:
  - vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real) (1:10)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- min: /0.0,0.0,-X,-X,-1.0,Y/
- X = lcyl/2 (only txbc = 'cyl'), box(3)/2 (else)
- Y = rsph (only txbc = 'sph), rcyl (only txbc = 'cyl')
- max /10.0,10.0,X,X,1.0,0.0/

2.27 nmlRDF

The namelist nmlRDF contains variables that control the calculation of running coordination number or radial distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	com-com	particle-particle
2	com-xxx	particle-atom
3	xxx-xxx	atom-atom

- Variables:
  - vtype
  - rmax
  - ndim
  - nbin
  - func
  - l2dtwo

vtype

`static1D_var(logical, real, real, integer, logical, character, real)` (1:3)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /0.0,0.0,0.0/ Max: /10.0,10.0,10.0/

rmax

`real` **default:** `10.0`

- Upper distance for particle separation considered.

ndim

`integer` **default:** `3`

- `2`: Sample distribution function in the xy-plane.
- `3`: Sample distribution function in the xyz-space.

nbin

`integer` **default:** `100`

- Number of bins used to sample the distribution functions.

func

`character(3)` **default:** `rdf`

- `rdf`: Radial distribution functions are calculated.
- `rcn`: Running coordination numbers are calculated.

l2dtwo

`logical` **default:** `.false.`

- `.true.`: Sampling of 2d radial distribution functions separately for z>0 and z<0. Useful for a system with txbc='xy' and two equivalent surfaces at z=±box2(3) (only ndim = 2)
- `.false.`: Nothing.

2.28 nmlRDFChain

The namelist nmlRDFChain contains variables that control the calculation of running coordination number or radial distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	com-com	center of mass-center of mass

Only chains within a separation of rmax are considered.

- Variables:
  - vtype
  - rmax
  - func

vtype

`static1D_var(logical, real, real, integer, logical, character, real)` (1:3)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /0.0/ Max: /10.0/

rmax

`real` **default:** `10.0`

- Upper distance for particle separation considered.

func

`character(3)` **default:** `rdf`

- `rdf`: Radial distribution functions are calculated.
- `rcn`: Running coordination numbers are calculated.

2.29 nmlRDFSph

The namelist nmlRDFSph contains variables that control the calculation of running coordination number or radial distribution functions of particles which positions are projected on a sphere.

- Variables:
  - ipsph
  - iptrcnsph
  - jptrcnsph
  - nbin
  - func

ipsph

`integer`

- The identity of the particle on which surface the projection is made.

iptrcnsph

`integer`

- Type of particle for which distribution function should be calculated.

jptrcnsph

`integer`

- Type of particle for which distribution function should be calculated.

nbin

`integer` **default:** `100`

- Number of bins used to sample the distribution functions.

func

`character(3)` **default:** `rdf`

- `rdf`: Radial distribution functions are calculated.
- `rcn`: Running coordination numbers are calculated.

2.30 nmlRDFCond

The namelist nmlRDFCond contains variables that control the calculation of conditional radial distribution functions. The distribution functions are made for five different bins of arccos(theta) making up -1 to 1.

- Variables:
  - vtype
  - rmax

vtype

`static1D_var(logical, real, real, integer, logical, character, real)` Min: /0.0/ Max: /10.0/

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.

rmax

`real` **default:** `10.0`

- Upper distance for particle separation considered.

2.31 nmlG3Dist

The namelist nmlG3Dist contains variables that control the calculation of normalized triplet correlation functions g(r12, r13, r23)/g(r12). r23 is represented by the angle theta formed by the vectors r12 and r13.

- Variables:
  - ipt1
  - ipt2
  - ipt3
  - snbin
  - slow
  - supp

  - tnbin
  - tlow
  - tupp
  - anbin
  - alow
  - aupp
  - nskip

ipt1

`integer`

- Particle type of the first particle.

ipt2

`integer`

- Particle type of the second particle.

ipt3

`integer`

- Particle type of the third particle.

snbin

`integer`

- Number of bins to sample the distance r12.

slow

`real`

- Lower end of r12 to be sampled.

supp

`real`

- Upper end of r12 to be sampled.

tnbin

`integer`

- Number of bins to sample the distance r13.

tlow

`real`

- Lower end of r13 to be sampled.

tupp

`real`

- Upper end of r13 to be sampled.

anbin

`integer`

- Number of bins to sample the angle theta.

alow

`real`

- Lower end of theta to be sampled.

aupp

`real`

- Upper end of theta to be sampled.

nskip

`integer` **default:** 1

- Interval of listing the distance r12.

2.32 nmlSF

The namelist nmlSF contains variables that control the calculation of partial structure factors.

- (i.) txbc='sph' or txbc='cyl' Variables:
  - txkscale
  - klow
  - logklow
  - logkupp
  - nbin
- (ii.) Cubic box and txbc='xyz'. The largest k-vector is 2Pi/box. Variables:
  - ndim
  - nbin
  - lqsorted
  - lsi

txkscale

`character(3)` **default:** `lin`

- `'lin'`: Linear k-scale
- `'log'`: Logarithmic k-scale

klow

`real` **default:** if sphrad $\neq$ 0: `2Pi/(10*sphrad )`, else `0.01`

- Lower k-vector (linear scale)

logklow

`real` **default:** `-4`

- Lower k-vector (logarithmic scale)

logkupp

`real` **default:** `1`

- Upper k-vector (logarithmic scale)

nbin

`integer` **default:** `100`

- Number of bins used to sample the partial structure factors.

ndim

`integer` **default:** `3`

- `2`: Structure factor in the xy-plane. It is averaged over 2 100 and 2 110 directions.
- `3`: Structure factor in the xyz-space. It is averaged over 3 100, 6 110 directions, and 4 110 directions.

nbin

`integer` **default:** `100`

- Number of bins used to sample the partial structure factors.

lqsorted

`logical` **default:** `.true.`

- `.false.`: List separately the structure factors of the different types directions.
- `.true.`: Sort the structure factors of the different types of directions and list the sorted structure factor.

lsi

`logical` **default:** `.false.`

- `.false.`: Nothing.
- `.true.`: Scattering intensities are calculated. Further specification is given in nmlScatIntens.

2.33 nmlScatIntens

The namelist nmlScatIntens contains variables that control the calculation of the scattering intensities using a multi shell profile of constant scattering properties of each particle.

- Variables:
  - nshell
  - rshell
  - cshell

nshell

`integer(1:npt)` **default:** `nshell(1:npt) = 1`

- Number of shells

rshell

`real(1:nshell,1:npt)` ** default:** `rshell(1,1:npt) = [ (radat(ipt) , ipt = 1,npt) ]`

- Radius of the shells.

cshell

`real(1:nshell ,1:npt)` **default:** `cshell(1,1:npt) = One`

2.34 nmlAngDF

The namelist nmlAngDF contains variables that control the calculation of angular distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	z'∗z'	cos(z'(ip)∗z'(jp))
2	z'∗r	cos(z'(ip)∗r(ijp))
3	r∗z'	cos(r(ijp)∗z'(jp))
4	oh∗r	cos(oh(ip)∗r(ijp)) ip has to be water
5	r∗oh	cos(r(ijp)∗oh(jp)) jp has to be water
6	oh.o	cos(oh(ip).o(jp)) ip and jp have to be water
7	o.o.o	cos(o(jp).o(ip).o(kp))
8	m∗m	cos(m(ia)∗m(ja)) m(ia) is the dipole vector of atom ia = ianpn(ip) + iashift
9	m∗r	cos(m(ia)∗r(ijp)) (the value of iashift is currently hard coded)
10	r∗m	cos(r(ijp)∗m(ja))

h(ip) denotes the z'-axis of particle ip, r(ijp) the normalized vector between particle ip and jp defined as r(jp)-r(ip), oh(ip) the direction of a oh-bond in water, oh(ip).o(jp) the largest angle of the four possible formed by the oh(ip) direction and the h(ip)-o(jp) vector (hydrogen bond angle), and o(jp).o(ip).o(jp) the angle formed by the location of three particles. The sampling of types 4-6 assumes that the first three sites of water are oxygen, hydrogen, and hydrogen. Generally ip refers to reference particles and jp to field particles. Only field particles within a separation of rmax from a reference particle are considered.

- Variables:
  - vtype
  - rmin
  - rmax

vtype

`static2D_var(logical, 2*real, 2*real, 2*integer, logical, character, real)(1:10)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /(-1.0, 0.0)/ Max: /(1.0, 2Pi)/

rmin

`real` **default:** `0.0`

- Lower distance for particle separation considered.

rmax

`real` **default:** `10.0`

- Upper distance for particle separation considered.

2.35 nmlAngExtDF

The namelist nmlAngDF contains variables that control the calculation of 2D angular distribution functions with respect to an external frame.

type	label	quantity
1	z'-axis	cos(theta)-phi 2d distributin function

- Variables:
- vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real)(1:3)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /7*(-1.0)/ Max: /7*0.0/

2.36 nmlOriDipDF

The namelist nmlOriDipDF contains variables that control the calculation of orientation/dipole distribution functions.

type	label	distribution functions
1	dir	orientation df based on ori(1:3,1:3)

type	label	distribution functions
2	dir_aver	symmetry-averaged orientation df
3	dip	dipole df
4	dip_tot	total dipole df

- Variables:
  - vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real)`(1:3)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /4∗(-1)/ Max: /4∗1.0/

2.37 nmlRadAngDF

The namelist nmlRadAngDF contains variables that control the calculation of radial-angular 2d distribution functions.

type	label	quantity
1	x'y'-z'-plane	two particle radial - angular r2d distributin function

- Variables:
  - vtype
  - txCoordSys
  - txAngle

vtype

`static2D_var(logical, 2*real, 2*real, 2*integer, logical, character, real)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Default values depend on txCoordSys

txCoordSys

`character(5)` **default:** “polar”`

- Coordinate system of distribution functions.
- `'polar'`: r and cos(theta)
- `'cart'`: sqrt(x∗∗2+y∗∗2) and z

txAngle

`character(6)` **default:** “theta1“

- Choice of angle.
- `'theta1'`: theta 1
- `'theta2'`: tehta 2
- `'psi'`: psi

2.38 nmlKirkwoodgk

The namelist nmlKirkwoodgk contains variables that control the calculation of running orientation averages. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	h∗h	sum_jp h(ip)∗h(jp)

- Variables:
  - vtype
  - rmax

vtype

`static1D_var(logical, real, real, integer, logical, character, real)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization (by radius∗∗3/2), title, and number of variable of vtype.
- Min: /0.0/ Max: /10.0/

rmax

`real` **default:** `10.0`

- Upper distance for particle separation considered.

2.39 nmlOriPolDF

The namelist nmlOriPolDF contains variables that control the calculation of orientation polarization distribution functions of molecular dipole moments. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	distribution functions
1	tot	orientation polarization, total

type	label	distribution functions
2	par	orientation polarization, parallel to central dipole (related to Kirkwood's Gk factor)
3	per	orientation polarization, perpendicular to central dipole
4	pevpa	orientation polarization, perpendicular versus parallel

- The analysis is made for a set of different radii
- The orientation polarization is normalized by (dipole moment)∗∗2 or radius∗∗(3/2)∗(dipole moment)∗∗2
- Variables:
  - vtype
  - lnorm
  - nrad
  - radius

vtype

`static2D_var(logical, real, real, integer, logical, character, real)`(1:4)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization (by radius∗∗3/2), title, and number of variable of vtype.
- Min: /0.0, -10.0, 0.0, 0.0/ Max: /4∗(10.0)/

lnorm

`logical` **default:** `.false.`

- `.true.`: Invoking normalization.
- `.false.`: No normalization invoked.

nrad

`integer` **default:** `2`

- Number of radii to consider

radius

`real`(1:nrad) **default:** `20.0`

- Radii to consider

2.40 nmlNNHB

The namelist nmlNNHB contains variables that control the calculation of no of nearest neighbours and no of hydrogen bonds. Field particles within thnn are considered as nearest neighbours to a reference particle. Field particles within rmax and with an interaction energy below a threshold value are considered as hydrogen bonded to a reference particle.

- Variables:
  - nthnn
  - thnn
  - nthhb
  - thhb
  - nnnhb
  - rmax

nthnn

`integer` **default:** `2`

- Number of separations considered.

thnn

`real(1:nthnn)` **default:** [3.3, 3.5]

- Distances considered. The largest thnn should not be larger than rmax given below.

nthhb

`integer` **default:** `2`

- Number of energy thresholds.

thhb

`real(1:nthhb)` **default:** [-10.0,-16.0]

- Energy thresholds.

nnnhb

`integer` **default:** `10`

- Maximal number of nearest neighbours and hydrogen bonds considered in the analysis.

rmax

`real` **default:** `3.5`

- Upper separation distance for hydrogen bonded particles.

2.41 nmlNNDF

The namelist nmlNNDF contains variables that control the calculation of nearest neighbour distribution functions. The R-F nearest neighbour distribution function gives the distribution of distances between a reference particle R and the nearest field particle F.

- Variables:
  - vtype

vtype

`static2D_var(logical, real, real, integer, logical, character, real)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization title, and number of variable of vtype.
- Min: /0.0/ Max: /10.0/

2.42 nmlChainDF

The namelist nmlChainDF contains variables that control the calculation of chain distribution functions. Distribution functions are calculated for each chain. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	rbb	bead-bead separation
2	ree	end-to-end separation
3	rg	radius of gyration
4	angle	angle between consequtive beads
5	cos	cos(180-angle)
6	shape	ree∗∗2/rg∗∗2 ratio
7	asph	asphericity
8	torp	toroidicity

- Variables:
  - vtype

vtype

`static2D_var(logical, real, real, integer, logical, character, real)`(1:8)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization title, and number of variable of vtype.
- Min: /0.0, 0.0, -1.0, 0.0, 0.0, 0.0, 0.0, 0.0/ Max: /12.0, 100.0, 100.0, 180.0, 1.0, 12.0, 1.0, 1.0/

2.43 nmlChainTypeDF

The namelist nmlChainTypeDF contains variables that control the calculation of chain distribution functions. Distribution functions are calculated for each type of chains. Any combination of the types of distribution functions listed below may be selected through vtype%l. Same input variables as for nmlChainDF.

2.44 nmlChainTypeExtDF

The namelist nmlChainTypeExtDF contains variables that control the calculation of chain distribution functions where the distribution functions are with respect to the lab frame. Distribution functions are calculated for each type of chains. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	com_x	com, x-direction
2	com_y	com, y-direction
3	com_z	com, z-direction
4	rg_z	square of radius of gyration projected on the z-axis
5	rg_xy	square of radius of gyration projected on the xy-plane

- Variables:
  - vtype

vtype

`static2D_var(logical, real, real, integer, logical, character, real)(1:5)`

- Flag for engagement, lower end, upper end, number of bins, flag for normalization title, and number of variable of vtype.
- Min: /-box2(1),-box2(2),-box2(3),0.0,0.0/
- Max: /box2(1),box2(2),box2(3),100.0,100.0/

2.45 nmlCBPC

The namelist nmlCBPC contains variables that control the calculation of probabilities that chain particles are near particles of other types. Distribution functions are calculated for each pair of chain molecules and particles of type iptpart.

- Variables:

  - iptpart
  - rcontact

iptpart

```
integer
```

- Type of particle to be considered.

rcontact

```
real
```

- Largest separation between chain particles and particles of the other type for considering the chain particle near the other particle.

2.46 nmlLoopTailTrain

The namelist nmlLoopTailTrain contains variables that control the calculation loop, tail, and train characteristics. Properties are calculated for each type of chains separately.

- Variables:
  - adscond

adscond

```
adscond_var(character, character, real)
```

- Adsorption condition('xy-plane'), selection of box ends, threshold distance of adsorption.

2.47 nmlCluster

The namelist nmlCluster contains variables that control the calculation of cluster size distribution functions. Objects within the distance rcluster from a given object forms a virtual bond and all objects directly or indirectly bonded form a cluster.

- Variables:
  - txobj
  - l1d
  - l2d
  - lpercolation
  - nobjt
  - iobjt

- rcluster
- txweight
- itestcluster

txobj

`character(8)` **default:** "particle"

- Object for which cluster analysis should be performed.
- `'particle'`: Particles, rcluster applies between particles.
- `'chain'`: Chains, rcluster applies between chain particles.
- `'chaincom'`: Chains, rcluster applies between chain com's.

l1d

`logical` **default:** `.true.`

- `.true.`: Calculation of 1d cluster size distribution functions.
- `.false.`: No such calculation.

l2d

`logical` **default:** `.false.`

- `.true.`: Calculation of 2d cluster size distribution functions.nptcluster=2 is required.
- `.false.`: No such calculation.

lpercolation

`logical` **default:** `.false.`

- `.true.`: Percolation analysis is made.
- `.false.`: No such analysis.

nobjt

`integer` **default:** 1

- Number of object types to be considered.

iobjt

`integer(1:npt)` **default:** `npt*0`

- Object types to be considered.

rcluster

`real`

- Upper separation for bonded objects (particles if txobj='particle' or 'chain' and chain com if txobj='chaincom').

txweight

`character(6)` **default:** "mass"`

- `'mass'`: Mass-weighted distribution.
- `'number'`: Number-weighted distribution.

itestcluster

`integer` **default:** `0`

- Flag for test output. This possibility is for maintenance purposes.
- `0`: Nothing. The normal option.
- `1`: Intermediate cluster data (several routines in static.F90). Type of particles to be inserted in the different sets.

2.48 nmlMultipoleDF

The namelist nmlMultipoleDF contains variables that control the calculation of electrostatic multipole moment distribution functions.

- Variables:
  - lmax
  - vmin
  - vmax
  - nbin
  - nrad
  - radius

lmax

`integer` **default:** `2`

- The largest multipole moment considered is 2∗∗lmax Real and imaginary components are analyzed separately and only for nonnegative m.

vmin

`real`(1:lmax) **default:** `0.0`

- Lower end of the sampled distribution function of each type.

vmax

`real`(1:lmax) **default:** `10.0`

- Upper end of the sampled distribution function of each type.

nbin

`integer` **default:** `100`

- Number of bins used to sample the distribution functions.

nrad

`integer` **default:** `1`

- Number of radii for which multipole moment df are calculated. If txbc='sph', then nrad=1 and radius=rsph are forced; otherwise multipole moment are calculated over spheres centered on all particles.

radius

`real`(1:nrad) **default:** nrad∗`20.0`

- Radii.

type	label	quantity

2.49 nmlEnergyDF

The namelist nmlEnergyDF contains variables that control the calculation of energy distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype. Only contributions from two-body potential terms are included.

type	label	quantity
1	bindu	binding energy, total
2	bindu	binding energy with particles in group jgr
3	pairu	pair energy with particles in group jgr within rmax

- Variables:
  - vtype
  - rmax

vtype

`static1D_var(logical, real, real, integer, logical, character, real)`(1:3)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /-150.0,-150.0,-25.0/ Max: /0.0,0.0,0.0/

rmax

`real` **default:** `1.0d10`

- Upper distance for particle separation for pair energy distribution function.

2.50 nmlWidom1

The namelist nmlWidom1 contains variables that control the calculation of excess chemical potentials using Widom's insertion method. Only charge neutral combinations of particles should be inserted.

- Variables:
  - ntimes
  - nset
  - nptset
  - iptset

ntimes

`integer` **default:** 1

- Number of samplings per occasion.

nset

`integer` **default:** 1

- Number of sets to evaluate.

nptset

`integer(`1:nset`)`

- Number of particles to be inserted in a set.

iptset

`integer(`1:nptset,1:nset`)`

- Types of particles to be inserted in a sets

2.51 nmlWidom2

The namelist nmlWidom2 contains variables that control the calculation of excess chemical potentials using Widom's insertion. Single particles are inserted and charge integration according to Svensson and Jönsson.

- Variables:
  - ntimes
  - nset
  - iptset

ntimes

`integer` **default:** 1

- Number of samplings per occasion.

nset

`integer` **default:** 1

- Number of sets to evaluate.

iptset

`integer(1:nptset,1:nset)`

- Types of particles to be inserted in a sets

2.52 nmlMeanForce1

The namelist nmlMeanForce1 contains variables that control the calculation of mean force between two particles (txbc='cyl' is required) according to F = Fmean + Fhs (the surface approach).

- Variables:
  - dr

dr

`real`

- Displacement for evaluation of Fhs

2.53 nmlMeanForce2

The namelist nmlMeanForce2 contains variables that control the calculation of mean force between two particles (only txbc='cyl') according to F = Fcorr + Fideal(midplane and an end) + Fhs(midplane) (midplane approach).

- Variables:
  - thickness
  - dz

thickness

`real`

- Thickness of volume for sampling Fideal

dz

`real`

- Displacement for evaluation of fhs

2.54 nmlPotMeanForce

The namelist nmlPotMeanForce contains variables that control the calculation of the potential mean force between two particles (only txbc='cyl'). The two particles of type iptpmf are assumed to be located on the z-axis. The pmf is set to zero at the separation cyllen /2.

- Variables:
  - iptpmf
  - rpmfzero

iptpmf

`integer` **default:** `1`

- Types of particles for which the potential of mean force is calculated. nppt(iptpmf) = 2 is required.

rpmfzero

`real` **default:** cyllen `/2`

- Separation at which pmf is set to zero.

2.55 nmlSurfaceArea

The namelist nmlSurfaceArea contains variables that control the calculation of the surface area available for a spherical prob.

- Variables:
  - ipt
  - rprobe
  - wradat
  - nrandom

ipt

`integer`

- Type of particles of interest for area determination.

rprobe

`real` **default:** `1.7`

- Radius of probe particle

wradat

`real`(1:nat) **default:** nat∗`0.0`

- van der Waal radius of atoms.

nrandom

`integer` **default:** `10000`

- Number of random numbers.

2.56 nmlTrajectory

The namelist nmlTrajectory contains variables that control the output of a trajectory.

- Variables:
  - iskip
  - rmin
  - rmax

iskip

`integer` **default:** `10`

- Write every trajectory data for every iskip timestep.

rmin

`real`(1:3) **default:** `-boxlen2(1:3)`

- Lower, left, and front corner of box bounding the particles.

rmax

`real`(1:3) **default:** `boxlen2(1:3)`

- Upper, right, and back corner of box bounding the particles.

type	label	quantity

2.57 nmlSubStructureDF

The namelist nmlSustructureDF contains variables that control the calculation of substructures.

type	label	quantity
1	rgsub	radius of gyration of substructure
2	rden	radial number density
3	rrden	reduced radial number density
4	com-sub	distance of com of chain to center of mass of substructure
5	rg-sub	average rg of a chain a a function of the distance to the center of mass of substructure
6	z(r)	sum of all charges as a function of the distance to the center of mass of substructure
7	zcum(r)	cumulative charge as a function of the distance to the center of mass of substructure
8	alpha	global degree of ionization distribution
9	a(r)	reduced radial degree of ionization

- Variables:
  - vtype
  - lptinsub

vtype

`static1D_var(logical, real, real, integer, logical, character, real)`(1:3)

- Flag for engagement, lower end, upper end, number of bins, flag for normalization, title, and number of variable of vtype.
- Min: /0.0/ Max: /100.0/

lptinsub

`logical`(1:npt)

- `.true.`: mass of particles of type ipt are used to evaluate mass center of substructure.

2.58 nmlNetworkDF

The namelist nmlNetworkDF contains variables that control the calculation of network distribution functions. Distribution functions are calculated for each network. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity

type	label	quantity
1	rg	radius of gyration
2	asph	asphericity
3	alpha	degree of ionization

- Variables:
  - vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real)(1:3)`

- Flag for engagement, lower end, upper end, number of bins. Other flags are not used.
- Min: /0.0,0.0,0.0/ Max: /100.0,1.0,1.0/

2.59 nmlNetworkRadialDF

The namelist nmlNetworkRadialDF contains variables that control the calculation of radial network distribution functions. Distribution functions are calculated for each network. Any combination of the types of radial distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	rpart(r)	radial particle number distribution
2	rdens(r)	radial particle density distribution
3	rgchain(r)	radial chain radius of gyration
4	q(r)	radial sum of all charges
5	qcum(r)	cumulative radial sum of all charges
6	alpha(r)	reduced degree of ionization
7	rchain(r)	radial chain number distribution

- Variables:
  - vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real)(1:7)`

- Flag for engagement, lower end, upper end, number of bins. Other flags are not used.
- Min: /0.0,0.0,0.0/ Max: /100.0,100.0,100.0/

2.60 nmlDynamic

- The namelist nmlDynamic contains variables that control the dynamic analyses. The detailed description of the use of stochastic data and evaluation of the correlation functions are given by the the data structures sf_var and cf_input_var, see chapter 8.
- Variables:
  - sfnplow
  - sfnpupp
  - lmsd
  - lorix
  - loriy
  - loriz
  - lliv
  - lanv
  - lfor
  - ltor
  - lidm
  - lutot
  - itestdyn

sfnplow

`integer` **default:** 1

- Lower id of particle to be used in the dynamic analysis.

sfnpupp

`integer` **default:** `np`

- Upper id of particle to be used in the dynamic analysis.

lmsd

`logical` **default:** `.false.`

- `.true.`: Mean square displacements are calculated. Further specification is given in namelist nmlMSD .
- `.false.`: Nothing.

lorix

`logical` **default:** `.false.`

•

`logical` **default:** `.false.`

- `.true.`: Orientational tcf for the molecular x'-axis is calculated. Further specification is given in namelist nmlOriXTCF .
- `.false.`: Nothing.

loriy

`logical` **default:** `.false.`

- `.true.`: Orientational tcf for the molecular y'-axis is calculated. Further specification is given in namelist nmlOriYTCF .
- `.false.`: Nothing.

loriz

`logical` **default:** `.false.`

- `.true.`: Orientational tcf for the molecular z'-axis is calculated. Further specification is given in namelist nmlOriZTCF .
- `.false.`: Nothing.

lliv

`logical` **default:** `.false.`

- `.true.`: Velocity tcfs for each molecular axis (molecular frame) and total velocity tcf are calculated. Further specification is given in namelist nmlLinVelCF.
- `.false.`: Nothing.

lanv

`logical` **default:** `.false.`

- `.true.`: Angular velocity tcfs for each molecular axis are calculated (molecular frame). Further specification is given in namelist nmlAnvVelCF.
- `.false.`: Nothing.

lfor

`logical` **default:** `.false.`

- `.true.`: Force tcfs for each molecular axis and total force tcf are calculated (box frame). Further specification is given in namelist nmlForCF.
- `.false.`: Nothing.

ltor

`logical` **default:** `.false.`

- `.true.`: Torque tcfs for each molecular axis and total torque tcf are calculated (box frame). Further specification is given in namelist nmlTorCF.
- `.false.`: Nothing.

lidm

`logical` **default:** `.false.`

- `.true.`: Induced dipole moment tcfs for each molecular axis and total induced dipole moment tcf are calculated (molecular frame). Further specification is given in namelist nmlIDPMCF.
- `.false.`: Nothing.

lutot

`logical` **default:** `.false.`

- `.true.`: Total energy tcf are calculated. Further specification is given in namelist nmlUtotCF.
- `.false.`: Nothing.

itestdyn

`integer` **default:** `0`

- Flag for test output.

2.61 nmlMSD

The namelist nmlMSD contains variables that control the calculation of the mean square displacement.

- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** sfnplow, sfnpupp, 3, 1, -

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, .false., .false.`

2.62 nmlOriXTCF, nmlOriYTCF and nmlOriZTCF

- The namelists nmlOriXTCF , nmlOriYTCF , and nmlOriZTCF contain variables which control the calculation of the three orientational tcf:s of the molecular axes.
- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** sfnplow, sfnpupp, 3, 3, -

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, -, -`

2.63 nmlLinVelTCF and nmlAngVelTCF

The namelists nmlLinVelTCF and nmlAngVelTCF contain variables that control the calculation of the velocity and angular velocity, respectively, tcf:s for the molecular axes.

- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** sfnplow, sfnpupp, 3, 1, -

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, -, -`

2.64 nmlForTCF and nmlTorTCF

The namelists nmlForTCF and nmlTorTCF contain variables that control the calculation of the force and torque, respectively, tcf:s about the box axes.

- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** sfnplow, sfnpupp, 3, 1, -

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, -, -`

2.65 nmlIDMTCF

The namelist nmlIDMTCF contains variables that control the calculation of the induced dipole moment tcf for the molecular axes.

- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** sfnplow, sfnpupp, 3, 1, -

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, -, -`

2.66 nmlUtotTCF

The namelists nmlUtotTCF contain variables that control the calculation of tcf of the total potential energy.

- Variables:
  - sf
  - cfin

sf

`sf_var(int, int, int, int, int)` **default:** `1, 1, 1, 1, -`

cfin

`cf_input_var(int, int, int, int, logical, logical, logical)` **default:** `-, -, -, 1, -, -, -`

2.67 nmlImage

The namelist nmlImage contains variables that control the interval of the calls and the calls to image date writing routines.

- Variables:
  - iimage
  - lvrml
  - lvtf
  - limageuser
  - lgr

iimage

`integer` **default::** 1

lvrml

`logical` **default:** `.false.`

- `.true.`: Coordinate data file for VRML images is prepared. Further specification is given in namelist nmlVRML.
- `.false.`: No preparation.

lvtf

`logical` **default:** `.false.`

- `.true.`:oordinate data file for VTF images is prepared. Further specification is given in namelist nmlVTF.
- `.false.`: No preparation.

limageuser

`logical` **default:** `.false.`

- `.true.`: Call of ImageUser, driver for user-provided image data writing routines.
- `.false.`: No such call.

lgr

`logical` **default:** `.false.`

- `.true.`: coloring is following the group assignment.
- `.false.`: coloring is following the atom types.

2.68 nmlVRML

The namelist nmlVRML contains variables that control the preparation of coordinate data files for VRML drawing.

- Variables:
  - txfile
  - txwhen
  - atsize
  - rgbcolor
  - blmax
  - bondr
  - tximage

txfile

`character(9)` **default:** “merged”`

- Controls the generation of Vrml files.
- `'merged'`: Coordinates for each frame are merged into one Vrml file separated by a wrapper.
- `'separated'`: Coordinates are written on separated files.

txwhen

`character(12)` **default:** “after_run‘

- Controls the frequency of generation of Vrml files.
- `'after_run'`: After the run.
- `'after_macro'`: After each macrostep.
- `'after_iimage'`: At an interval of iimage steps (iimage is given in nmlImage).

atsize

`real`(1:nat) **default:** radat `(1:nat)`

- Size of the atoms.

rgbcolor

`real`(3,1:nat) **default:** some values

- Each triple defines the RGB-color of the atoms.

blmax

`real` **default:** `1.5`

- Maximal separation between two atoms for drawing a bond between them.

bondr

`real` **default:** `0.3`

- Radius of the bonds.

tximage

`character(20)`(1:3) **default:** ['frame',' ',' ']

- Controls additional aspects and features of the snapshots
- (1)“frame’: `Make a frame around the box.`
- `(2)`'one_plane': `Draw a plane at -box(3)/2.`
- `(2)`'two_plane': `Draw planes at -box(3)/2 and box(3)/2.`
- `(3)`'undopbc': `Periodic boundary conditions are not applied when drawing chains.`
- `(3)`'nopbc“: Same as 'undopbc’.

2.69 nmlVTF

The namelist nmlVTF contains variables that control the preparation of coordinate data files for VTF drawing.

- Variables:
  - txfile
  - txwhen
  - tximage
  - atsize
  - rgbcolor
  - blmax
  - bondr
  - bondres
  - sphres
  - lframezero

txfile

`character(5)` **default:** "merged"`

- Controls the generation of Vtf files. Options are:
- `'split'`: Coordinates are written in separated files.
- `'merge'`: All frames are written in one single vtf file.

txwhen

`character(12)` **default:** "after_run"`

- Controls the frequency of generation of Vrml files.
- `'after_run'`: After the run.
- `'after_macro'`: After each macrostep.
- `'after_iimage'`: At an interval of iimage steps (iimage is given in nmlImage).

tximage

`character(20)`(1:3)

- Controls additional aspects and features of the snapshot. Options are
- (1) "frame': `Make a frame around the box`
- `(2)`'undopbc': `Undo periodic boundary conditions for bonded and cross-linked systems`
- `(3)`'center': `Move center of mass of all particles to origin of the simulation cell`
- `(3)`'xycenter'`: Move center of mass of all particles to origin of the simulation cell (consider x- and y-direction only).

atsize

`real`(1:nat) **default:** radat `(1:nat)`

- Size of the atoms.

rgbcolor

`real`(3,1:nat) **default:** some values

- Each triple defines the RGB-color of the atoms.

blmax

`real` **default:** `0.0`

- Maximal separation between two atoms for drawing a bond between them.

bondr

`real` **default:** `0.3`

- Radius of the bonds.

bondres

`real` **default:** `12.0`

- Bond resolution: Number of prisms bonds are being drawn of

sphres

`real` **default:** `12.0`

- Atom resolution: Number of prisms atoms are being drawn of

lframezero

`logical` **default:** `.true.`

- `.true.`: Frame of the initial configuration is being stored.
- `.false.`: Frame of the initial configurations is not stored.
- lframezero = `.true.` does not work with lgr = `.true.`

2.70 nmlMixed

The namelist nmlMolmix contains variables that control the choice of miscellaneous calculations In all cases two particles are involved, and their types are specified by txpt1 and txpt2.

- Variables:
  - mode
  - txpt1
  - txpt2
  - coord1
  - coord2
  - ip

mode

`integer(1:10)` **default:** `0`

- Select type of calculation. Several calculations, at most 10, of either same or different types may be performed by listing the corresponding values of mixmode. The correct number and order of namelists have to follow namelist nmlMolmix.
- `1`: Potential energy curve is generated. Further specification is given in namelist nmlMolmix1.
- `2`: Potential energies on a lattice is calculated. Further specification is given in namelist nml↩ Molmix2.
- `3`: The global potential energy minimum is calculated. Further specification is given in namelist nmlMolmix3.
- `4`: Random coordinates are generated, and corresponding potential energies are calculated. Further specification is given in namelist nmlMolmix4.
- `5`: The second virial coefficient is calculated. Further specification is given in namelist nmlMolmix5.
- `6`: Orientational averaged potential energy is calculated. Further specification is given in namelist nmlMolmix6.

txpt1

`character(10)`

- Particle type of particle one.

txpt2

`character(10)`

- Particle type of particle two.

coord1

`real(1:12)`

- Initial coordinate of particle of type txpt1 in the sequence ro(1), ro(2), ro(3) ori(1,1), ori(1,2), ..., ori(3,3). ro is the center-of-mass location. ori(1,1:3) is projection of the particle frame axis z' on the x, y, and z-lab axes, and similarly with ori(2,1:3) and ori(3,1:3).

coord2

`real(1:12)`

- As for coord1 but for particle of type txpt2.

ip

`integer`

- Particle to be moved or which coordinates should be integrated, either 1 or 2.

2.71 nmlMixed1

The namelist nmlMixed1 contains variables that control the generation of the potential energy curve.

- Variables:
  - dxx
  - dyy
  - dzz
  - iaxis
  - dr
  - mstep
  - umax

dxx

`real`

- Translational step in x-direction.

dyy

`real`

- Translational step in y-direction.

dzz

`real`

- Translational step in z-direction.

iaxis

`integer`

- `1`: Rotation about the x-axis.
- `2`: Rotation about the y-axis.
- `3`: Rotation about the z-axis.

dr

`real`

- Rotational step (in degrees).

mstep

`integer`

- Maximum number of steps.

umax

`real`

- Upper potential energy limit where the generation of the potential energy curve is stopped.

2.72 nmlMixed2

The namelist nmlMixed2 contains variables that control the calculation of potential energies on a lattice. The energies are calculated on a 2D grid in position space and at each point the potential energy is minimized as a function of the orientation.

- Variables:
  - itmax
  - udel
  - dr
  - iaxis
  - rdist
  - rlow1
  - rupp1
  - nrdiv1
  - rlow2
  - rupp2
  - nrdiv2

itmax

`integer`

- Maximum number of iterations of the orientation optimization.

udel

`real`

- Potential energy tolerance for the iteration of the orientation optimization.

dr

`real`

- Rotation step angle for the iteration of the orientation optimization.

iaxis

`integer`

- `1`: Grid in the yz-plane.
- `2`: Grid in the zx-plane.
- `3`: Grid in the xy-plane.

rdist

`real`

- Value of the constant coordinate, either x, y, or z depending on iaxis.

rlow1

`real`

- Lower value of either the y, z, or x-coordinate.

rupp1

`real`

- Upper value of either the y, z, or x-coordinate.

nrdiv1

`integer`

- Number of grid points in either the y, z, or x-coordinate.

rlow2

`real`

- Lower value of either the y, z, or x-coordinate.

rupp2

`real`

- Upper value of either the y, z, or x-coordinate.

nrdiv2

`integer`

- Number of grid points in either the y, z, or x-coordinate.

2.73 nmlMixed3

The namelist nmlMixed3 contains variables that control the calculation of the global potential energy minimum.

- Variables:
  - itmax
  - udel
  - dxx
  - dyy
  - dzz
  - dr

itmax

`integer`

- Maximum number of iterations.

udel

`real`

- Potential energy tolerance for the iteration.

dxx

`real`

- Translational step in x-direction.

dyy

`real`

- Translational step in y-direction.

dzz

`real`

- Translational step in z-direction.

dr

`real`

- Rotational step (in degrees).

2.74 nmlMixed4

The namelist nmlMixed4 contains variables that control the calculation of random coordinates. The random coordinates are restricted to a volume, given in spherical polar coordinates, and to a potential energy range.

- Variables:
  - iwr
  - rlow
  - rupp
  - thlow
  - thupp
  - filow
  - fiupp
  - ulow
  - uupp
  - no
  - ntry

iwr

`integer`

- `0`: Evenly distributed random numbers in the r direction.
- >0: Higher probability for larger r coordinates.

rlow

`real`

- Lower limit in the r-direction.

rupp

`real`

- Upper limit in the r-direction.

thlow

`real`

- Lower limit in the theta-direction (in degrees).

thupp

`real`

- Upper limit in the theta-direction (in degrees).

filow

`real`

- Lower limit in the phi-direction (in degrees).

fiupp

`real`

- Upper limit in the phi-direction (in degrees).

ulow

`real`

- Lower limit of potential energy.

uupp

`real`

- Upper limit of potential energy.

no

`integer`

- Number of desired random coordinates.

ntry

`integer`

- Maximum number of attempts.

2.75 nmlMixed5

The namelist nmlMixed5 contains variables that control the calculation of the second virial coefficient. The integration is done by generation of random coordinates restricted to a specified volume given in spherical polar coordinates. The uncertainty is given by one standard deviation obtained from ten sets of random coordinates.

- Variables:
  - iwr
  - rlow
  - rupp
  - thlow
  - thupp
  - filow
  - fiupp
  - no
  - nblock

iwr

`integer`

- `0`: Evenly distributed random numbers in the r direction.
- $>$0: Higher probability for larger r coordinates.

rlow

`real`

- Lower limit in the r-direction.

rupp

`real`

- Upper limit in the r-direction.

thlow

`real`

- Lower limit in the theta-direction (in degrees).

thupp

`real`

- Upper limit in the theta-direction (in degrees).

filow

`real`

- Lower limit in the phi-direction (in degrees).

fiupp

`real`

- Upper limit in the phi-direction (in degrees).

no

`integer`

- Number of desired random coordinates in one set.

nblock

`integer` **default:** `10`

2.76 nmlMixed6

The namelist nmlMixed6 contains variables that control the calculation of orientation averaged potential energy at a given separation. The orientational integration is done by generation of random coordinates restricted to a specified domain, given in spherical polar coordinates. The potential energy distribution of the potential energy is plotted.

- Variables:
  - rlow
  - rupp
  - thlow

  - thupp
  - filow
  - fiupp
  - ulow
  - uupp
  - no
  - nbin

rlow

`real`

- Lower limit in the r-direction.

rupp

`real`

- Upper limit in the r-direction.

thlow

`real`

- Lower limit in the theta-direction (in degrees).

thupp

`real`

- Upper limit in the theta-direction (in degrees).

filow

`real`

- Lower limit in the phi-direction (in degrees).

fiupp

`real`

- Upper limit in the phi-direction (in degrees).

ulow

`real`

- Lower value of the potential energy distribution function.

uupp

`real`

- Upper value of the potential energy distribution function.

no

`integer`

- Number of random orientations.

nbin

`integer`

- Number of bins used to sample the distribution functions.

2.77 nmlComplexation

The namelist nmlComplexation contains variables that can be used for the analysis of interparticle complexation.

- Variables:
  - rcut_complexation
  - lClusterDF
  - lComplexFraction
  - lComplexDist
  - lSegmentComplex

rcut_complexation

`real` **default:** `0.0`

- Cut-off distance for complexation.

lClusterDF

`logical` **default:** `.false.`

- `.true.`: The cluster size distribution will be calculated.
- `.false.`: Nothing.

lComplexFraction

`logical` **default:** `.false.`

- `.true.`: The fraction of complexation will be calculated.
- `.false.`: Nothing.

lComplexDist

`logical` **default:** `.false.`

- *documentation missing*

lSegmentComplex

`logical` **default:** `.false.`

- *documentation missing*

2.78 nmlComplexDist

The namelist nmlComplexDist contains variables that control the calculation of complexation distribution functions. Any combination of the types of distribution functions listed below may be selected through vtype%l.

type	label	quantity
1	w	fraction of complexation

- Variables:
  - vtype

vtype

`static1D_var(logical, real, real, integer, logical, character, real)`

- Flag for engagement, lower end, upper end, number of bins. Other flags are not used.
- Min: /-0.005,-0.005,-0.005/ Max: /1,005,1.005,1.005/

3 User-provided routines

The file moluser.F90 contains routines that are conditionally called and which might be modified by the user. This allows the software to be extended without having to modify the core of it.

The existing code in the distributed file moluser.F90 reflects the current use of it. The procedure to insert a new routine is as follows:

– At the appropriate position in driver moluser, extend the test of txUser and write the appropriate call to the new subroutine

– Add your new subroutine in moluser.F90

– Compile

The file moluser.F90 contains six drivers and several routines specifying different types of tasks. The six drivers are PotentialUser , SetParticleUser , DumpUser , GroupUser , StaticUser , and ImageUser .

3.1 PotentialUser

Routine PotentialUser is the driver of user-provided routines for two-body potentials. It calls routines specifying various potentials on the basis of the value of txpot(iptjpt), which is specified in namelist nml↩ Potential. The procedure to install a new user-provided potential is as follows:

– Extend the test of txpot(iptjpt) , and write the appropriate call to a new routine in routine PotentialUser.

– Add your new routine to file moluser.F90.

3.2 SetParticleUser

Routine SetParticleUser is the driver of user-provided routines for generating a start configuration. It calls routines specifying types of initial particle positions on the basis of the value of txsetconf(ipt) , which is specified in namelist nmlSetConfiguration. The procedure to install a new user-provided start configuration is as follows:

– Extend the test of txsetconf(ipt) and write the appropriate call to a new routine in the routine Set↩ ParticleUser.

– Add your new routine to file moluser.F90.

3.3 DumpUser

Routine DumpUser is the driver of user-provided routines for dumping data. It calls routines specifying different types of dumping. Generally, all these user-provided routines are called if variable ldumpuser , specified in namelist nmlDump , is true. The procedure to install a new dump analysis is as follows:

– Write the appropriate call to the new routine in the routine DumpUser.

– Add your new routine to file moluser.F90

3.4 GroupUser

Routine GroupUser is the driver of user-provided routines for particle group division. It calls routines specifying different division of particles into groups on the basis of the values of variables ref or field [later stored in txappl(m) (m = 1 for ref and m = 2 for field particles)], which are specified in namelist nmlGroup . The procedure to install a new group division procedure is as follows:

– Extend the test of txappl(m) (m = 1 for ref and m = 2 for field particles) and write the appropriate call to the new routine in driver GroupUser.

– Add your new routine to the file moluser.F90.

Follow closely the pattern of one of the other user-provided routines for particle group division. After the statement 'if (iStage = iWriteInput) then', variables ngr , igrpt , and txgrr have to be specified. ngr denotes the number of groups, igrpt the order number of the particle type of particles that might belong to this group, and txgrr a text label of the group. After 'if (iStage = iSimStep) then', the variable igr (within the particle loop) has to be assigned the order number of the group to which the particle should belong (zero if the particle does not belong to any group). The last two lines of the particle loop should be the same as in the other cases.

3.5 StaticUser

Routine StaticUser is the driver of user-provided routines for static analysis. It calls routines specifying different types of analysis. Generally, all these user-provided routines are called if variable lstaticuser, specified in namelist nmlStatic , is true. The procedure to install a new static analysis routine is as follows:

– Write the appropriate call to the new routine in the routine StaticUser.

– Add your new routine to the file moluser.F90.

3.6 ImageUser

Routine ImageUser is the driver of user-provided image data writing routines. It calls routines specifying different types of such routines. Generally, all these user-provided routines are called if variable limageuser, specified in namelist nmlImage , is true. The procedure to install a new image data writing routine is as follows:

– Write the appropriate call to the new routine in the routine ImageUser.

– Add your new routine to the file moluser.F90.

4 Distributed files

This chapter contains a complete list of the distributed source files.

Filename	Description
`molsim.F90`	Routines specific for MOLSIM
`particle.F90`	Routines for particles
`potential.F90`	Routines for potential and force tables
`coordinate.F90`	Routines for coordinates including start configuration
`md.F90`	Routines for MD integration
`mc.F90`	Routines for MC integration
`bd.F90`	Routines for BD integration
`nlist.F90`	Routines for neighbour lists
`energy.F90`	Routines calculating energy and forces
`dump.F90`	Routines for dumping and reading variables
`group.F90`	Routines for particle group division
`static.F90`	Routines for static analysis
`dynamic.F90`	Routines for dynamic analysis
`image.F90`	Routines for preparing image data files
`mixed.F90`	Routines specific for mixed tasks
`molaux.F90`	Auxiliary routines
`mesh.F90`	Routines for making meshes
`statistics.F90`	Routines for statistical analyses
`mollib.F90`	General library routines
`parallel.F90`	Parallel interface between MOLSIM and MPI
`moluser.F90`	User-provided routines
`molsim.lib`	Potential library

Command	Description
molsim_ser	Command file for execution of serial version of MOLSIM
molsim_par	Command file for execution of parallel version of MOLSIM
makefile	Makefile for compiling and linking
archive	Command file for filing

5 Subroutines and Functions

This chapter lists the routines and functions in the MOLSIM software. The routines are listed according to the Fortran files they are located in and in the order they appear in the files.

- mol.F90
- molsim.90
- particle.F90
- potential.F90
- coordinate.F90
- md.F90
- mc.F90
- bd.F90
- nlist.F90
- energy.F90
- denergy.F90
- dump.F90
- group.F90
- static.F90
- dynamic.F90
- image.F90
- mixed.F90
- molaux.F90
- mesh.F90
- statistics.F90
- mollib.F90
- parallel.F90
- moluser.F90
- sso.F90
- gc.f90
- celllist.F90

5.1 mol.F90

MolModule *main module for the Molsim software*

5.2 molsim.90

Molsimdriver *driver of the Molsim program*

IOMolsim(iStage) *perform mixed tasks, mainly i/o*

IOSystem *perform i/o on general system variables*

IOScale *perform i/o on scaling variables*

IOCnf *perform i/o on the configuration file*

ControlAver *caluclate control quantities*

MDAver *calculate averages of md scific variables*

DispAver *calculate averages of displacements*

OriOrderAver *calculate averages of orientational order*

PosOriAver *calculate averages of positions and orientation*

MainAver *calculate various quantities*

ThermoAver *calculate averages of thermodynamic quantities*

ChargeAver *calculate averages of related to weak charges*

IndDipMomAver *calculate averages of induced dipole moment*

ChainAver *calculate averages of chain quantities*

NetworkAver *calculate averages of network quantities*

HierarchicalAver *calculate averages of chain quantities*

ThermoInteg *handle thermodynamic integration*

DistFunc *calculate distribution functions*

PressureContact *calculate contract contribution to the reduced pressure*

GContact *calculate rdf contract value*

TestSimulation *write test output*

5.3 particle.F90

ParticleModule *module for particle*

Particle(iStage) *particle variables*

SetObjectParam1 *setobject parameters*

SetObjectParam2 *set object parameters*

PAxesSystem *transform from input to principal axes system*

5.4 potential.F90

PotentialModule *module for potential*

PotentialDriver *driver of potential routines*

IOPotTwoBody *perform i/o on potential variables for two body interactions*

PotTwoBodyTab1 *set tables for two-body interactions*

PotTwoBodyTab2 *set tables for two-body interactions for particle pair iptjpt*

PotTwoBodyTab3 *set table for two-body interaction iatjat*

SetUBuffer $*$transform to r^2, calculate coefficients, and store them in a table$*$

CheckUBuffer *check the accuracy of the tabulation of one interval*

PotDefault *default potential*

Pot_1_6_12 *1, 6, 12-potential*

MCY *MCY water potential*

SphEwaldTrunc *spherical ewald truncated potential*

Ramp *ramp potential (with soft slope change)*

SquareWell *Square-well potential (with a soft rise)*

AsakuraOosawa *Asakura-Oosawa depletion potential*

LekkerkerkerTuinier *Lekkerkerker-Tuinier depletion potential*

calc_LT_depletion *calculate Lekkerkerker-Tuinier depletion potential (in kT units)*

Nemo *nemo potentials*

NemoType1 *nemo potential of type 1*

NemoType2 *nemo potential of type 2*

NemoType3 *nemo potential of type 3*

NemoType4 *nemo potential of type 4*

NemoType5 *nemo potential of type 5*

NemoType6 *nemo potential of type 6*

NemoType7 *nemo potential of type 7*

WriteUBuffer *write the content of ubuf for pair iatjat*

TestUBuffer *calculate the accuracy of the table for pair iatjat*

PlotPotTwoBodyTab *plot two-body potential*

TestPotTwoBodyTab *test of utwobodytab and utwobody*

CalcScrCUfac $*$calculate exp(r1/scrlen) and two related derivatives$*$

CalcGCDUfac *calculate erfc(1/sqrt(1/a1∗∗2+1/a2∗∗2)∗r1) and two related derivatives*

CalcEwaldUfac *calculate erfc(ualpha∗r1) and two related derivatives*

CalcRFUfac *calculate modified interactions according to nymand and linse, eqs(55)*

EwaldErrorUReal *estimate error of real space potential energy*

EwaldErrorURec *estimate error of reciprocal space potential energy*

TestEwald *examine truncation error of ewald summation (energies in kT)*

TestEwaldStd *examine truncation error of standard ewald summation*

TestEwaldSPM *examine truncation error of the reciprocal space; smooth particle mesh ewald summation*

SetImageSph *Setup of image charges and dipoles according to Friedman*

IOPotChain *perform i/o on bond and angle interaction variables*

ChainTab *call routines setting up potential tables for chains*

BondLengthTab *make and use of lookup table to get random bond lengths*

BondAngleTab *make and use of lookup table to get random bond angles*

IOPotExternal *perform i/o on external potential variables*

IOPolarizationIter *perform i/o on polarization iteration variables*

5.5 coordinate.F90

CoordinateModule *module for coordinate*

Coordinate *handle coordinates and velocities*

SetChargeWeakChargeCase *initiate charge of for the weak-charge case*

SetConfiguration *generate a start configuration*

SetLattice *generate a lattice configuration*

SetOrigin *generate a point in the center of the unit cell*

SetSq2D *generate a 2d square lattice*

SetHex2D *generate a 2d hexagonal lattice*

SetPC *generate a primitive cubic lattice*

SetBCC *generate a body-centered cubic lattice*

SetFCC *generate a face-centered cubic lattice*

SetSM2 *generate a sm2 lattice*

SetDiamond *generate a cubic diamond lattice*

SetH2O *generate a cubic lattice, im3m (ice viii)*

SetN2 *generate a cubic lattice, pa3 (solid n2)*

SetBenzene *generate a cubic lattice, pbca (solid benzene)*

SetRandom *generate random positions and orientations*

SetRandomFixOri *generate random positions and fixed orientations*

SetChainLine *generate a linear configuration for chain particles (only for one chain)*

SetChainCircle *generate a circular configuration for chain particles (only for one chain)*

SetChainRandom *generate random positions and orientations for chain particles*

SetChainRandomIntOri *generate random positions and internally fixed orientations for chain particles*

SetSphBrush *generate a configuration for a spherical brush*

SetPlanarBrush *generate a configuration for a planar brush*

SetHierarchical *generate a configuration for hierarchical polymers*

SetPeriodicNetwork *generate a periodic network*

SetNetwork *generate a nonperiodic network*

SetCoreShell *generate a configuration with particles located in a spherical shell*

SetPartPosRandom *generate a random particle position*

SetPartPosRandomN *generate a random position at a given distance to a previously set particle*

CheckPartOutsideBox *check if a particle is outside the box*

CheckTooFoldedChain *check if a too folded chain*

MakeCrossLink *make crosslinks between particles labeled clbeg and particles labeled clend*

TestCoordinate *write test coordinates*

StoreInteger *store integer*

5.6 md.F90

MDModule *module for md*

MDDriver *molecular dynamics driver*

IOMD *perform i/o on molecular dynamics variables*

VelVer *perform one step according to the velocity form of the Verlet algorithm*

Gear *perform one step according to the Gear algorithm*

GetLinAcc *calculate new linear acceleration*

GetAngAcc *calculate new quaternion accelerations*

ScaleLength *scale lengths through a pressure coupling to an external bath*

ScaleVel *scale velocities and calculate new kinetic energies*

ScaleLinVel *scale linear velocities through a thermal coupling to an external bath*

ScaleAngVel *scale angular velocities through a thermal coupling to an external bath*

SetVel *set linear and angular velocities*

SetLinVel *set linear velocities according to the maxwell distribution*

SetAngVel *set angular velocities according to the maxwell distribution*

SetZeroMom *set zero linear and angular moments*

SetZeroLinMom *set zero linear moments*

SetZeroAngMom *set zero angular moments*

GetLinMom *calculate total linear moment*

GetAngMom *calculate total angular moment*

WriteLinMom *write total linear moment*

WriteAngMom *write total angular moment*

GetKinEnergy *calculate total kinetic energies and temperatures*

GetTStep *return the time step*

GetTime *return the time*

GetISetVel ∗return the value of lsetvel∗

GetIZeroMom ∗return the value of lzeromom∗

TestSimulationMD *write MD test output*

5.7 mc.F90

MCModule *module for mc*

MCDriver *monte carlo driver*

IOMC *perform i/o on monte carlo variables*

MCPass *perform one mc pass (np trial moves)*

SPartMove *perform one single-particle trial move*

SPartCl2Move *perform one single-particle + cluster2 trial move*

PivotMove *perform pivot rotation trial move*

ChainMove *perform one chain trial move*

SlitherMove *perform one chain slithering trial move*

BrushMove *perform one brush trial move*

BrushCl2Move *perform one brush + cluster2 trial move*

HierarchicalMove *perform one hierarchical trial move*

NetworkMove *perform one network trial move*

VolChange *perform one volume change trial move*

NPartChange *perform one number of particle change trial move*

ChargeChange *perform a charge-change trial move*

SetPartPosRandomMC *generate a random position*

MCAllDriver *mcall driver*

IOMCAll *performing i/o on monte carlo all variables*

MCAllPass *perform one mc pass (simultaneous move of all particles)*

ClusterMember *calculate secondary cluster members using Verlet neighbour list*

ClusterMemberLList *calculate secondary cluster members using linked lists*

GetRandomTrialPos *calculate translational displacement and trial position*

GetRandomTrialOri *calculate a trail orientation*

GetFixedTrialOri *get fixed trial orientation*

GetRotatedTrialOri *get rotated trial orientation*

RotateSetPart *rigid-body rotation of a set of particles*

FizedZCoord *no displacement in z-direction*

FizedXYCoord *no displacement in the xy-plane*

FixedChainStart *put rotm = ro and drotm to zero if particle is the first segment in a chain*

ShiftZDirection *center the system at z = 0 as much as possible (only for lbccyl)*

CheckPartBCTM *check boundary conditions of trial particle coordinate*

AddNeighbours *adding neighbours*

UpdateOri *get fixed trial orientation*

SetTrialAtomProp *set trial atom properties from trial particles properties*

CheckAtomBCTM *check boundary conditions of trial atom coordinate*

Metropolis *Metropolis test*

MCUpdate *update energies and coordinates after accepted mc trial move*

Restoreiptm *restore lptm*

MCAver *calculate means of mc specific variables*

MCAllAver *calculate means of mc specific variables for mcall*

MCWeightIO *perform i/o for an umbrella sampling with a distance dependent weight*

MCWeight *calculate weight based on trial and current position for mcweight*

MCWeightInverse *calculate the inverse weight based on current position for mcweight*

UmbrellaIO *perform i/o of the umbrella potential sampling*

UmbrellaWeight *calculate weight for umbrella sampling*

UmbrellaBin *calculate bin for umbrella potential*

UmbrellaUpdate *update weight function for updated umbrella potential*

MCPmfIO *perform i/o of the mc pmf sampling*

MCPmfWeight *calculate weight for mcpmf*

MCPmfBin *calculate bin for umbrella potential*

MCPmfUpdate *update weight function for mc pmf*

TestMCMove *test output for MC move*

5.8 bd.F90

BDDriver *Brownian dynamics driver*

IOBD *perform i/o on Brownian dynamics variables*

BDStep *perform one bd step (configuration space)*

5.9 nlist.F90

NListModule *module for neighbour (verlet and liked) list*

NListDriver *driver of calls for making lists of interacting particles*

IONList *perform i/o on neighbour list variables*

LoadBalance *control the calls of LoadBalanceRealSpace and LoadBalanceRecSpace*

LoadBalanceRealSpace *set iobjmyid for load balancing of single loop with equal load*

LoadBalanceRecSpace *set vbounds for load balancing of reciprocal space loop over nz and ny*

NList *control the calls of SetVList, VListAver, SetLList, and LListAver*

SetVList *set neighbour (verlet och liked) lists*

SetVListMD *set verlet lists for md*

SetVListMDLList *set verlet lists for md using linked lists*

SetVListMC *set verlet lists for mc*

SetVListMCLList *set verlet lists for mc using linked lists*

VListAver *calculate statistics involving verlet lists and particle partitioning*

TestVList *write verlet list and node decomposition test output*

SetLList *set linked lists*

SetNCell *set list of cells to be used for a position*

LListAver *calculate statistics involving linked cell lists and particle partitioning*

TestLList *write linked list*

CalcTotNeighbourPair *calculate number of neigbouring pairs*

Getdrnlist ∗return the value of drnlist∗

Getnpmyid *return the value of npmyid*

Getdrnlist *return the value of ncellllist*

5.10 energy.F90

EnergyModule *module for energy*

UTotal *calculate energies, forces, torques, virial, and pressure*

UTwoBodyA *calculate two-body potential energy; only monoatomic particles*

UTwoBodyALList *calculate two-body potential energy; only monoatomic particles*

UTwoBodyACellList *calculate two-body potential energy*

UTwoBodyP *calculate two-body potential energy; general particles*

UEwald *calculate potential energy, forces, and virial from charges; k-space*

EwaldSetArray *calculate eikx, eiky, and eikz arrays used for ewald summation; k-space*

EwaldSetArray2d *calculate sinkx, coskx, sinky, and cosky arrays used for ewald summation; k-space*

UDipole *calculate potential energy from charges and dipoles*

UDipoleP *calculate potential, field, and field gradient from charges and dipoles*

UDipoleEwald *calculate potential, field, and field gradient from charges and dipoles; k-space*

UManyBodyP *calculate many-body potential energy; general particles*

FieldStat *calculate potential and field from charges and static dipoles*

FieldStatEwald *calculate potential and field from charges and static dipoles; k-space*

IterIdm *iterate induced dipole moments self-consistently*

FieldIdm *calculate field from induced dipoles*

FieldIdmEwald *calculate field from induced dipoles; k-space*

FieldTot *calculate field, field gradient, and virial from charges and total dipoles*

FieldTotEwald *calculate field, field gradient, and virial from charges and total dipoles; k-space*

UIntraReac *calculate intramolecular contribution to energy and force in rf*

PackReduceU *pack the different energy contributions and perform all_reduce*

UDipoleSph *calculate charge and dipole potential energy using image charge approximation*

UDielDis *calculate energy for charge and in a system with dielectric discontinuities*

UDielDisPlane *calculate coulomb energy in a system with dielectric discontinuity at z = 0*

UDielDisSph *calculate coulomb energy in a system with spherical dielectric discontinuity*

ImageIntSph *get the image interaction for spherical geometry*

UTwoBodyPair *calculate two-body potential energy and force between two particles*

UBond *calculate bond potential energy, forces, and virial*

UAngle *calculate angle potential energy, forces, and virial*

UCrossLink *calculate crosslink potential energy, forces, and virial*

UExternal *calculate external potential energy*

UreactionSphere *calculate the reaction energy of an ion and a dielectric sphere*

UExternalUpdate *update arrays for external energy*

xCCLM *return the spherical harmonics*

xPLM *return the associate legendre polynomial p(l,m) at x*

Longrangecontr *calculate long-range contribution of the electrostatic energy*

SPMFFTRec *make Fourier transformation, reciprocal space operations, and back FFT*

EwaldSetup *setup for ewald summation*

Getnkvec *return the value of nkvec*

Getnkvec2d *return the value of nkvec2d*

Gettime_ewald *return the value of ncut2d*

UWeakChargeA *calculate two-body potential energy; only monoatomic particles*

UWeakChargeP *calculate two-body potential energy; general particles*

5.11 denergy.F90

DUTotal *calculate energy difference between two configurations*

DUTwoBody *calculate two-body potential energy difference*

UTwoBodyANew *calculate two-body potential energy for new configuration*

UTwoBodyAOld *calculate two-body potential energy for old configuration*

UTwoBodyANewLList *calculate two-body potential energy for new configuration*

UTwoBodyAOldLList *calculate two-body potential energy for old configuration*

UTwoBodyANewCellList *calculate two-body potential energy for old configuration*

UTwoBodyAOldCellList *calculate two-body potential energy for old configuration*

UTwoBodyPNew *calculate two-body potential energy for new configuration*

UTwoBodyPOld *calculate two-body potential energy for old configuration*

DUTwoBodyEwald *calculate two-body potential energy difference; k-space*

DUWeakChargeEwald *calculate two-body potential energy difference for titrating systems; k-space*

EwaldSetArrayTM *calculate eikxtm, eikytm, and eikztm arrays for moving particles*

EwaldSetArray2dTM *calculate sinkxtm, coskxtm, sinkytm, and sinkytm arrays for moving particles*

EwaldUpdateArray *update eikx, eiky, eikz, and sumeikr for moving particles*

DUDipole *calculate potential energy difference from charges and dipoles*

DUDipolePNew *calculate potential energy from charges and dipoles for new configuration*

DUDipolePOld *calculate potential energy from charges and dipoles for old configuration*

DUDipoleEwald *calculate potential energy change; k-space*

DUDipoleSph *calculate charge and dipole potential energy difference*

DUDipoleSphNew *calculate charge and dipole potential energy for new configuration*

DUDipoleSphOld *calculate charge and dipole potential energy for old configuration*

DUDielDis *calculate coulomb energy in a system with dielectric discontinuities*

DUDielDisPlane *calculate coulomb energy in a system with dielectric discontinuity at z = 0*

DUDielDisSph *calculate coulomb energy in a system with a radial dielectric discontinuity*

DUBond *calculate bond potential energy difference*

DUAngle *calculate angle potential energy difference*

DUCrossLink *calculate crosslink potential energy difference*

DUExternal *calculate external potential difference*

UWeakChargeANew *calculate two-body potential energy for new configuration*

UWeakChargeAOld *calculate two-body potential energy for old configuration*

UWeakChargePNew *calculate two-body potential energy for new configuration*

UWeakChargePOld *calculate two-body potential energy for old configuration*

5.12 dump.F90

DumpModule *module for dumping*

DumpDriver *dump driver*

IODump *performing i/o on dump variables*

DoDump *performing dumping matters*

5.13 group.F90

Group *classify reference and field particles into groups and make average*

GroupAll *group division: each particle type forms one group*

GroupType *group division: select particle type forms one group*

5.14 static.F90

StaticDriver *driver of static analysis routines*

SPDF *calculate single particle distribution functions*

RDF *calculate running coordination number or radial distribution function*

RDFChain *calculate running coordination number or radial distribution function*

RDFSph *calculate rcn or rdf of particles which positions are projected on a sphere*

RDFCond *calculate conditional radial distribution function*

G3Dist *calculate g(r12,r13,r23)/g(r12)*

SFDriver *calculate partial structure factors*

SFPBC *calculate partial structure factors*

ScatIntens *calculate scattering intensity, form factor, and structure factor*

SFNoPBC *calculate partial structure factors*

AngDF *calculate angular distribution functions*

AngExtDF *calculate angular 2d distribution functions with respect to external frame*

OriDipDF *calculate orientation/dipole distribution function*

SphHarAver *calculate averages of unnormalized spherical harmonics*

RadAngDF *calculate radial-angular 2d distribution functions*

Kirkwoodgk *calculate Kirkwood gk-factor*

OriPolDF *calculate orientation polarization distribution functions*

NNHB *calculate number of nearest neighbours and number of hydrogen bonds*

NNDF *calculate the nearest neighbour distribution*

ChainDF *calculate chain distribution functions*

ChainTypeDF *calculate chain type distribution functions*

ChainTypeExtDF *calculate chain type distribution function relative external frame*

ChainBeadPartContact *calculate probability of chain bead-particle contact*

LoopTailTrain *calculate loop, tail, and tail characteristics for chains*

CheckAdsChainSeg *check if a chain and its segments are adsorbed*

ClusterSD *calculate cluster size distribution*

ClusterSD1D *calculate cluster size distribution*

ClusterSD2D *calculate 2d cluster size distribution*

ZeroSecondMoment *calculate zero and second moment for an ionic, neutral system*

MultipoleDF *calculate multipole moment distribution function and mean square average*

CalcMultipole *calculate multipole moments of spherical volumes arising from dipoles*

EnergyDF *calculate energy distribution functions*

Widom1 *calculate excess chemical potential using Widom's method (neutral set)*

Widom2 *calculate excess chemical potential using Widom's method (charge integration)*

DUWidomx *calculate the change of the potential energy of the insertion*

UTwoBodyWidom *calculate two-body potential between two particles; monoatomic particles*

UTwoBodyWidomP *calculate two-body potential between two particles, polyatom particles*

MeanForce1 *calculate mean energies and forces between two particles*

MeanForce2 *calculate mean energies and forces between two particles*

UBondAcrossZ *calculate potential energy from bonds across z = 0*

PotMeanForce *calculate potential of mean force*

SurfaceArea *calculate surface area exposed by all particles of one type*

Crystalformat *write atoms in the crystalographic format starting with atoms*

Trajectory *write the trajectory on flist using every iskip timestep*

ExcessAmount *calculate excess amount*

SubStructureDF *calculate distribution functions of properties of a SubStructure*

NetworkDF *calculate network distribution functions*

NetworkRadialDF *calculate radial distribution functions of properties of networks*

5.15 dynamic.F90

DynamicModule *module for dynamic*

DynamicDriver *driver of dynamic analysis routines*

MSD *mean square displacement*

OrixTCF *orientation time correlation function of paricle x'-axis*

OriyTCF *orientation time correlation function of paricle y'-axis*

OrizTCF *orientation time correlation function of paricle z'-axis*

LinVelTCF *linear velocity time correlation function*

AngVelTCF *angular velocity time correlation function*

ForTCF *force time correlation function*

TorTCF *torque time correlation function*

IDMTCF *induced dipole moment time correlation function*

UtotTCF *total potential time correlation function*

PrepareDynamic *prepare for dynamic analyse*

MemoryDynamic *allocate memory*

CFCalc *calculate correlation function*

CFCalcSub *calculate time correlation function, subroutine*

CFWrite *write time correlation function*

5.16 image.F90

ImageDriver *driver of image preparation routines*

ImageVRML *generate input files for vrml 97 viewers*

VRMLSub *writes a vrml-97 file*

ImageVTF *generate vtf file(s) and tcl-script for VMD*

UpdateVTFFileName *update vtf file name with respect to a given frame iframe*

WriteVTFHeader *write header of the vtf file*

WriteVTFCoordinates *writes current atom coordinates to vtf-file*

WriteTCLScript *TCL: "tool command language"*

UndoPBC *undo periodic boundary conditions*

MakeBondList *make bond list*

5.17 mixed.F90

MixedDriver *mixed task driver*

Mixed1 *generate a potential energy curve*

Mixed2 *calculate potential energies on a 2d lattice*

Mixed3 *calculate the global potential energy minimum*

Mixed4 *generate random coordinates*

Mixed5 *calculate the second virial coefficient*

Mixed6 *calculate orientational averaged potential energy*

LineMove *perform one movement, rotation or translation, to a new minimum*

NewCoord *translate and rotate a particle and update its atom coordinates*

RandomPos *generate a random position*

CopyCoord *copy coordinates*

5.18 molaux.F90

MolauxModule *molaux module*

CalcCOM *Calculate the center of mass of given coordinates*

PBC *apply periodic boundary conditions*

PBCr2 *apply periodic boundary conditions and calculate r**2*

PBC2 *calculate periodic boundary condition displacement*

FileOpen *open external files*

FileFlush *flush external files (contains a system call)*

SetBoxParam *set box related parameters*

SetPartOriRandom *set particle orientation matrix corresponding to a random orientation*

SetPartOriLab *set particle orientation matrix to match lab frame orientation*

SetPartOriBond *set particle orientation matrix to match bond directions of the particle*

SetAtomPos *set atom positions in the laboratory frame*

SetAtomDipMom *set atom dipole moments in the laboratory frame*

SetAtomPolTens *set atom polarization tensors in the laboratory frame*

CalcIndDipMom *calculate induced dipole moment: total and particle*

CalcSysDipMom *calculate system dipole moment from charges, static, and induced dipoles*

CalcPartDipMom *calculate particle dipole moment from charges, static, and induced dipoles*

dvol *return the volume between two concentric shells, 4*pi/3 is not included*

darea *return the area between two concentric spheres, pi is not included*

angcos *return the angle cosine between two 3d vectors*

angle_rad *return the angle between two 3d vectors*

angle_deg *return the angle between two 3d vectors*

WritePartAtomProp *write particle and atom properties*

IWarnHCOverlap *warn if hard-core overlap between particle ip and other particles*

WarnHCOverlap *warn if hard-core overlap among atoms*

WarnAtomOutsideBox *warn if atoms are outside box*

WarnPartOutsideBox *warn if particles are outside box*

UndoPBCChain *undo the periodical boundary conditions for a chain molecule*

CalcChainProperty *calculate properties of a single chain*

CalcNetworkProperty *calculate properties of network number inw*

Asphericity *calculate the asphericity based on the 2:nd moments in the principal frame*

CalcLTT *calculate loop, tail, and tail characteristics for a single chain*

CalcChainPairListGeneral *calculate a list of chain pairs based on particle-particle separation*

CalcChainComPairListGeneral *calculate a list of chain pairs based on com-com separation*

CalcPartPairListAll *calculate a list of particle pairs based on their separations*

CalcPartPairListGeneral *calculate a list of particle pairs*

CalcPartPairListIpt *calculate pair list for particles*

Perculation *determine if a system of bonded objects is perculated*

CorrAnalysis *make correlation analysis of a single variable*

BlockAverAnalysis *read data and calculate average and precision for different block lengths*

WriteTrace *write trace information*

WriteVecAppend *write real array at the end of an external unit*

EllipsoidOverlap *check overlap between two ellipsoids with equal semi-axes*

SuperballOverlap *superball overlap check*

SuperballOverlapOF *superball overlap check, return overlap function*

SuperballOverlap_NR *check for overlap between two superballs*

SuperballStatNR *superball check statistics, NR overlap method*

SuperballStatMesh *superball check statistics, mesh overlap check*

SuperballAver *superball overlap-check average*

SuperballDF *superball overlap-check distribution functions*

5.19 mesh.F90

MeshModule *module for mesh*

BuildSuperball *mesh a superball*

BuildDopTree *build a DOP tree*

BuildNode *build a node*

TriTri *check if two triangles intersect*

Cint *tries to find interval on the line intersection between the triangle planes*

Coplanar *check for overlap in coplanar triangles*

PointInTriangle *check if point in a triangle*

Cross2 *cross product of two 2d vectors*

Cross *Cross product of two 3d vectors*

Sort2 *sort two elements in ascending order*

MidPoint *find midpoint of two 3d arrays, midpoint may be modified*

l2i *change a logic variable to an integer*

findExtrema *documentation_missing*

union *documentation_missing*

volume *documentation_missing*

idSort *documentation_missing*

swap *documentation_missing*

transformation *documentation_missing*

rotation *documentation_missing*

TestOverlap *documentation_missing*

OverlapMesh *translate and rotate a mesh and check overlap of two trees*

OverlapTree *check overlap of two trees*

OverlapDop *check overlap between two bounding boxes*

OverlapLeaf *check overlap of two leafs*

5.20 statistics.F90

StatisticsModule *module for statistics*

SetlBlockAver ∗set variable lblockaver∗

ScalarSample *sample scalar quantities*

ScalarSampleBlock *sample scalar quantities; fixed block length*

ScalarSampleExtrap *sample scalar quantities; extrapolate to large blocklen*

ScalarNorm *normalize scalar quantities*

ScalarWrite *write scalar quantities*

ScalarAllReduce *make all_reduce of scalar quantities*

DistFuncSample *sample distribution functions*

DistFuncNorm *normalize distribution functions*

DistFuncHead *write heading of distribution functions*

DistFuncWrite *control the calls of DistFuncShow, DistFuncPlot, and DistFuncList*

DistFuncShow *list bin numbers, x, y, and dy of distribution functions*

DistFuncPlot *plot distribution functions*

DistFuncList *list x, y, and dy values of distribution functions*

DistFuncAverValue *calculate and write average of distribution functions*

DistFuncAverDist *calculate average and spread among distribution functions*

DistFuncAllReduce *make all_reduce of df quantities*

DistFunc2DSample *sample two-dimensional distribution functions*

DistFunc2DNorm *normalize two-dimensinal distribution functions*

DistFunc2DHead *write heading of two-dimensional distribution functions*

DistFunc2DShow ∗list bin numbers and z as well as bin numbers and dz of two-dimensional distribution functions∗

DistFunc2DList ∗list bin numbers and z as well as bin numbers and dz of two-dimensional distribution functions∗

5.21 mollib.F90

MollibModule *mollib module*

InvInt *return the inverse of an integer*

Center *center a string*

SpaceOut *space out a string*

ReplaceText *Replace text in a string*

ErfLocal *return the error function, erf(x)*

Erf1 *return the error function, erf(x)*

Erf2 *return the error function, erf(x)*

GammaLn *return the logarithm of the gamma function*

PL *return the legendre polynomial p(l) at x*

PLM *return the associate legendre polynomial p(l,m) at x*

CCLM *return the spherical harmonics*

DDJKM *return one wigner's rotation matrix element*

SetW3J *precalculate wigner's 3j-symbols*

WW3J *return one wigner 3j-symbol*

Trap *integration by the trapezoidal rule*

TrapNe *integration by the trapezoidal rule with varying steplength*

IntegCirPoints *get points and weight for numerical integration over a circle*

PolFit *calculate least square fit to a polynomial*

PolVal *calculate the value of the polynomial for a given x-value*

MatVecMul *multiply a matrix with a vector, c = a∗b*

VecMatMul *multiply a vector with a matrix, c = b∗a*

MatInv2 *matrix inversion of a 2x2 matrix*

MatInv3 *matrix inversion of a 3x3 matrix*

MatInv *matrix inversion with solution of linear equations*

Diag *diagonalize a real matrix and calculate eigenvectors*

Eigensort *Given the eigenvalues d and eigenvectors v as output from Diag this*

Smooth *smooth by using spline functions argonne routine e350s*

CsEval *return the value and the first two derivatives of a spline func*

VInter *interpolate in a 1d array*

LinInter *linear interpolation between (x0,y0) and (x1,y1)*

CardinalBSpline *calculate values and derivatives of the n:th order cardinal B-spline*

HeapSort *sort an array into ascending order, Heapsort algorithm*

HeapSortIndex *generate an index array such that vec(index(j)) is ascending, Heapsort algorithm*

MakeCluster *make clusters*

MakeCluster1 *make clusters*

MakeCluster2 *make clusters*

CalcClusterMember *get members of the cluster to which object iobj belongs to*

CarToSph *transform a vector from a cartensian to a spherical polar coordinate system*

SphToCar *transform a vector from a spherical polar to a cartesian coordinate system*

CarToStd1 *transform a vector from a cartensian coordinate system to standard form*

Std1ToCar *transform a vector from standard form to a cartensian coordinate system*

OriToEuler *transforming orientation matrix ori to Euler angles a, b, c*

EulerToQua *transform Euler angles a, b, c to quaternions*

OriToQua *transform orientation matrix to quaternions*

QuaToOri *transform quaternions qua to orientation matrix*

AxisAngToOri *transform rotation axis and rotation angle to a rotation matrix*

LabToPri *transform a vector from lab to principal frame*

PriToLab *transform a vector from principal to laboratory frame*

EulerRot *rotation of a column vector (x,y,z) using the Euler angles*

EulerRotStd *rotate tensor of standard form using Euler angles (NOTE CONVENTION NOT CLEAR)*

OrthoOri *orthogonalize particle frame*

CheckOriOrtho *check that a matrix is orthogonal*

QuaNorm *normalize quaternions*

AngVelToQuaVel *calculate quaternion velocities from quaternions and angular velocities (principal frame)*

QuaVelToAngVel *calculate angular velocities (principal frame) from quaternions and quaternion velocities*

Random *return a random number in the range of 0 $<$ ran $<$ 1*

GauRandom *return a normally distributed stochastic variable*

CirRandom *return two random numbers x,y on a unit circle*

SphRandom *return three random numbers x,y,z on a unit sphere*

InvFlt *return the inverse of a real variable*

GetRelDiff *calculate relative differences*

CpuAdd *add and write total cpu time elapsed since start*

CpuLeft *check to see if the execution should be stopped*

CpuTot *write total cpu time elapsed since start*

SecondsSinceStart *return cpu time used since start in seconds*

WriteVec *write a vector*

WriteMat *write a matrix*

WriteStd *write a variable of standard form*

WriteFront *write a front*

WriteHead *write a heading*

WriteDateTime *write date and time*

WriteIOStat *write value of iostat and take appropriate action*

Warn *write a warning message*

Stop *write a stop message and stop process*

LowerCase *change a string to lower case*

UpperCase *change a string to upper case*

SubStr *determine the number of substrings of a string*

SignMagn *get sign and magnitude of a real number*

Advance *read from an external unit until a string match is found*

Plot *plot a function*

BrentMod *get the minimum of a function*

KnuthShuffle *Shuffle 1-Dimensional List if Integers !Pascal Hebbeker*

5.22 parallel.F90

ParallelModule *module to be used in routines invoking mpi*

par_initialize *initialize parallel matter*

par_finalize *finalize parallel matter*

par_comm_size *get number of processes*

par_comm_rank *get my id and set master and slave*

par_barrier *barrier synchronisation*

par_bc_characters *broadcast character variables*

par_bc_logicals *broadcast logicals*

par_bc_logical *broadcast logical*

par_bc_ints *broadcast integers*

par_bc_int *broadcast of scalar integer*

par_bc_ints8 *broadcast integers of kind 8*

par_bc_reals *broadcast double precision reals*

par_bc_real *broadcast double precision real*

par_bc_comps *broadcast double precision complex variables*

par_allreduce_logicals *perform global logical or and redistribution of logical variables*

par_allreduce_logical *perform global logical or and redistribution of logical variables*

par_allreduce_ints *perform global sum and redistribution of integer variables*

par_allreduce_int *perform global sum and redistribution of integer variable*

par_allreduce_reals *perform global sum and redistribution of double precision variables*

par_allreduce_real *perform global sum and redistribution of double precision variable*

par_allreduce_comps *perform global sum and redistribution of double precision complex variables*

par_max_ints *find maximum value and redistribute integer variables*

par_min_reals *find minimum value and redistribute double precision variables*

par_max_reals *find maximum value and redistribute double precision variables*

par_reduce_reals *perform global sum and redistribution of double precision variables*

par_allgather_shortints *perform a distribution of array of integer(2)*

par_allgather_ints *perform a distribution of array of integer(4)*

par_scatter_reals *scatter double precision data from root to the other processes*

par_gather_reals *gather double precision data from all processes except root to root*

par_handshake *hand shaking between master and slaves*

par_timing *parallel timing*

par_error *error handler for par routines*

5.23 moluser.F90

PotentialUser *driver of user-provided routines for potentials*

SodiumChloride *na-na, na-cl, and cl-cl potentials*

EwaldSquareWell *real space ewald plus square-well potential (with a soft rise)*

ElstatScreen *potential where the first term is screened if it is a 1/r term*

SetConfigurationUser *driver of user-provided routines for generating a start configuration*

SetCNF *generate a configuration by reading ro and ori from the begining of fcnf*

SetOneHelix *generate a helical conformation*

SetCubic2D1Surf *generate a cubic configuration with particles at z = -(boxlenz/2 + delta)*

SetCubic2D2Surf *generate a cubic configuration with particles at z = +-(boxlenz/2 + delta)*

SetChainRandomTrans *generate a random start configurations for chain particles + chain translation*

SetCapsid *generate a uniform distribution on the surface of a sphere of rcap+2*

SetCapsid_250 *generate a random configuration with 250 particles on a spherical surface*

SetCapsidRandom *generate a random start configurations for paticles of type ipt*

SetInsideCapsid *generate a random configuration inside a capsid*

SetChainInsideCapsid *generate a random chain configuration inside a capsid*

SetSpool *generate a spool-like structure inside a capsid*

SetSpoolm *generate a multilayered spool-like structure inside a capsid*

SetSpoolm1 *generate a multilayered spool-like structure inside a capsid*

setsheet *set bcc 'sheet' or 'tube'*

SetRandomCylinderShell *generate a random configuration inside a cylinder with a hard cylinder within*

SetChainLinePMA *generate a line configuration for PMA chain particles*

DumpUser *driver of user-provided routines for dumping*

BBDump *dumps three angles and two energies*

ChainCOMDump *dump center of mass of chains*

ChainReeDump *dump end-to-end separations of chains*

DipMomTotDump *dump the total dipole moment*

xy_coordinates_Jasper *read x,y coordinates*

GroupUser *driver of user-provided routines for particle group division*

GroupBW1 *group division for one benzene molecule in water*

GroupBW2 *group division for one benzene molecule in water*

GroupBW3 *group division for one benzene molecule in water*

GroupBBW1 *group division for two benzene molecules in water*

GroupBBW2 *group division for two benzene molecules in water*

GroupBBW3 *group division for two benzene molecules in water*

GroupBBW4 *group division for two benzene molecules in water*

GroupSurface *group division for particles of type ipt = 2 at surface and in bulk*

groupsurface2 *group division for particles only at surface*

GroupAds1 *group division for particles belonging to adsorbed chains*

GroupAds_layer1_ramp *group division for adsorbed (layer 1) particles*

GroupNetworkGenerations *group division according to generations of non-periodic network*

GroupWeakCharge *group division according to titratable species - respective division in*

StaticUser *driver of user-provided routines for static analysis*

ScalarDemo1 *calculate an average of a scalar quantity*

ScalarDemo2 *calculate average of scalar quantities*

BondOrder *calculate an average of a scalar quantity*

TabulationQl *calculate an average of a scalar quantity*

ChainBeadCylContact *calculate probability of chain bead-cylinder contact*

Min *calculate minimum energy configuration of a bcc "sheet" or "tube"*

S1 *calculate orientational order parameter S1 in all dimensions*

S2 *calculate orientational order parameter S2 in all dimensions*

C2 *calculate C2 nematic order parameter of projection on xy-plane*

Q *calculate nematic order parameter of projection on xy-plane with x*

SFPBC2D *documentation_missing*

SFPBC2DNoAv *Does not average over x and y, compatible with boxlength in x \= y*

PosMeanSph *calculate means of positions*

MacroionOneSph *calculate various df for system of one macroion + ions, sph geometry*

MacroionTwoCyl *calculate various df for system of two macroions + ions, cyl geometry*

MeanElFieldZCyl *calculate mean electrostatic field in the z-direction*

DomainModule *module for domain analysis*

DomainIO *domain analyse based on Kirkwood gk-factor*

CalcDomain *calculate dipole domains based on Kirkwoods gk-factor (G Karlstrom)*

DomainDominating *sample gk for the dominating domain of each configuration*

DomainDistFunc *Domain distribution functions*

UDomain *domain-domain dipole-dipole interaction energy*

TCFDipDomain *sample gk for the dominating domain of each configuration*

SCDF *calculate single chain distribution functions as a function of z coordinate*

AdsRadGyr *calculate radius of gyration distribution functions of adsorbed and adsorbing chains*

AdsBondOrder *calculate bond order distribution function for adsorbed chains*

AdsPropDyn *write chain adsorption dynamic data on external unit*

CalcAvBondOrder *calculation of the average bond order among provided chains*

CalcBondOrder *calculation of the bond order among bonds in chains of type ict*

AdsEventDyn *write time and occurrence of adsorption events on FUSER*

AdsModule *module for adsorption module*

AdsExam *call of AdsExam routines*

AdsExam1 *generate curvz file using the primary adsorption data*

AdsExam2 *analysis of primary adsorption data; adsorption events*

AdsExam3 *analysis of primary adsorption data; adsorption length*

AdsExam4 *analysis of primary adsorption data; adsorption length*

ReadPrimAdsData *read user-provided primary adsorption data*

WritePrimAdsData *write user-provided primary adsorption data*

Z_DF_Slit *distribution function based on z-coordinates, planar geometry*

ElMom *electrostatic moment of a particle*

XYProjectDF *calculate on 'z' = 0 projected normalized density df*

SPDF_COMB *calculate single particle distribution functions for comb polymers (coarse_grained)*

COMB_DF *calculate type distribution functions of comb polymer type (coarse model)*

COMBAver *calculate averages of comb chain quantities*

SFPBC_COMB *calculate partial structure factors*

OCF *calculate bond-bond orientational correlation function and where particle i belongs to group igr*

OCF_DF *calculate type distribution functions of comb polymer type (coarse model)*

ChainBeadBeadContact *documentation_missing*

ElPot *calculate electrostatic potential as a function of distance from the center of lab frame*

ImageUser *driver of user-provided routines for generating image input files*

JosUser *auxillary routine for syncronizing input data for Jos' project*

ComplexationModule *Module for analysing the Complexation*

DoComplexation *documentation_missing*

5.24 sso.F90

SSOModule *Module for the SSO simulation*

5.25 gc.f90

NPartChange *perform one number of particle change trial move*

5.26 celllist.F90

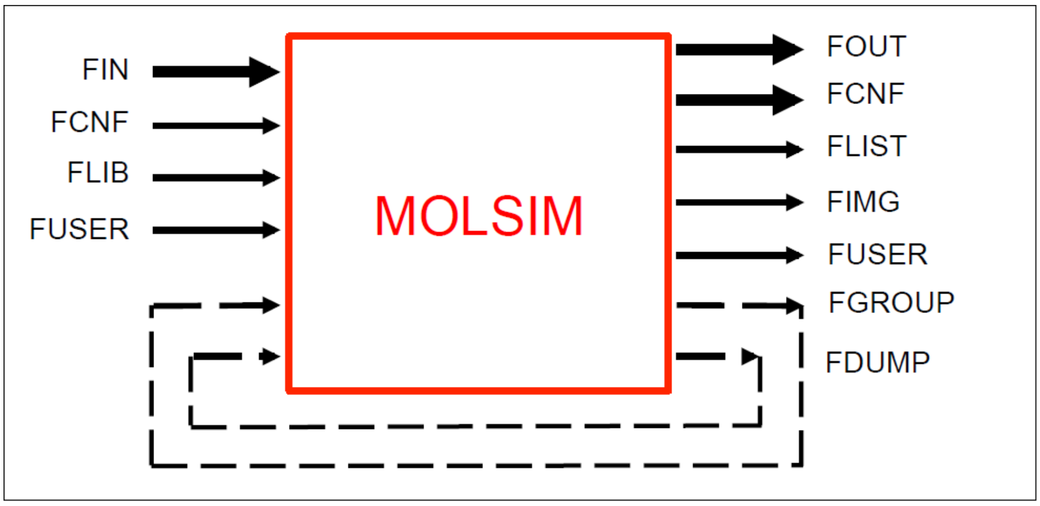
CellListModule *a new improved implementation of the linked list*

6 External units

A file is specified by its name and type, which are separated by a dot. The file types are used in a predetermined way to describe the use of the file and leaves file name to label the project. Within a project the file name is independent of program.

FPOS, FORI, FLIV, FANV, FFOR, FTOR, and FIDM are collectively called dump files.

File type	Generic name	Content	binary
in	FIN	Input data	
out	FOUT	Output data	
cnf	FCNF	Configuration and average data	binary
lib	FLIB	Potential library	
list	FLIST	List data	
img	FIMG	Image data	
user	FUSER	At the disposal to the user	
group	FGROUP	Particle group data	
pos	FPOS	Box lengths and positions of particles	binary
ori	FORI	Orientations of particles	binary
liv	FLIV	Linear velocities (box frame) of particles	binary
anv	FANV	Angular velocities (principal frame) of particles	binary
for	FFOR	Forces (box frame) of particles	binary
tor	FTOR	Torques (box frame) of particles	binary
idm	FIDM	Induced dipole moments (box frame) of particles	binary



Flow chart of the use of external files. Bold arrows denote files that are always required or generated, while thin arrows denote files that might be required or generated, depending on input variables.

7 Data structures

The following data structures are used for input variables:

bonds etc

```
type bond_var
   real(8) :: k ! force constant
   integer(4) :: p ! power
   real(8) :: eq ! equilibrium value
end type bond_var
```

adsorption conditions

```
type adscond_var
   character(8) :: txplane !'xy-plane' is yet the only option
   character(3) :: txend ! '-','+','-|+' for negative, positive, both ends
   real(8) :: offset ! threshold distance of adsorption for the com of a particle
end type adscond_var
```

stochastic data

```
type sf_var
   integer(4) :: nplow ! lower particle number
   integer(4) :: npupp ! upper particle number
   integer(4) :: ndim ! number of dimension of the stochastic variable
   integer(4) :: fac ! factor describing the separation of stoch. data
   integer(4) :: ngr ! logical flag if for single sf value
end type sf_var
```

correlation functions (input)

```
type cf_input_var
   integer(4) :: nmean ! length (number of values) forming a mean
   integer(4) :: nlevel ! length (number of values) forming a level
   integer(4) :: nolevel ! number of levels
   integer(4) :: legendre ! order of Legendre polynomial (1, 2, or 3)
   logical :: lsvalue ! logical flag if for single sf value
   logical :: lsubmean ! logical flag if subtraction of mean of cf
   logical :: lnorm ! logical flag if normalization of cf
end type cf_input_var
```

correlation functions

```
type cf_var
  integer(4)    :: nmean                        ! length (number of values) forming a mean
  integer(4)    :: nlevel                      ! length (number of values) forming a level
   integer(4)     :: ratio                                  ! nlevel/nmean
   integer(4)     :: nolevel                                ! number of levels
  integer(4)    :: legendre                    ! order of Legendre polynomial (1, 2, or 3)
   logical       :: lsvalue                          ! logical flag if for single sf value
  logical      :: lsubmean                     ! logical flag if subtraction of mean of cf
   logical       :: lnorm                            ! logical flag if normalisation of cf

   real(8), allocatable    :: sf(:,:,:,:)                   ! stochastic function
  real(8), allocatable   :: sf_aver(:,:,:)         ! stochastic function, local average (of particle)
   real(8), allocatable    :: sf_mean(:,:)               ! stochastic function, gobal mean
   real(8), allocatable    :: cf(:,:,:)                     ! time correlation function
  real(8), allocatable   :: cf2(:,:,:)               ! time correlation function squared
  integer(4), allocatable :: Np(:,:,:)            ! counter of correlation sampling (of particle)
  integer(4), allocatable :: Ngr(:,:,:)           ! counter of correlation sampling (of group)
   integer(8), allocatable :: Nlev(:,:)                ! counter of level (of particle)
end type cf_var
```

1D static variables

```
type static1d_var
   logical :: l ! logical flag for engagement
   real(8) :: min ! minimum value
   real(8) :: max ! maximum value
   integer(4) :: nbin ! number of bins
   logical :: lnorm ! logical flag for normalization
   character(10):: label ! title
   real(8) :: nvar ! expanded into nvar variables (not read)
end type static1D_var
```

2D static variables

```
type static2d_var
   logical :: l ! logical flag for engeagement
   real(8) :: min(2) ! minimum value
   real(8) :: max(2) ! maximum value
   integer(4) :: nbin(2) ! number of bins
   logical :: lnorm ! logical flag for normalization
   character(10):: label ! title
   real(8) :: nvar ! expanded into nvar variables (not read)
end type static2D_var
```

potential energy

```
   type potenergy_var
real(8), allocatable :: twob(:)     ! two-body contribution (excluding ewald contribution)
  real(8), allocatable :: oneb(:)     ! one-body contribution (dielectric discontinuity)
   real(8)                :: tot           ! total
  real(8)          :: rec        ! reciprocal electrostatic space contribution (UEwald)
   real(8)              :: stat          ! static electrostatic contribution (umbodyp)
  real(8)           :: pol         ! polarization electrostatic contribution (umbodyp)
   real(8)                :: bond
   real(8)                :: angle
   real(8)                :: crosslink
   real(8)                :: external      ! external contribution
end type potenergy_var
```

chain properties

```
type chainprop_var
   real(8)    :: ro(3)                      ! center of mass
   real(8)    :: rbb2                       ! bead-to-bead distance squared
   real(8)    :: angle                      ! angle between consecutive beads
   real(8)    :: cos                       ! cos(180 - angle between consecutive beads)
   real(8)    :: ree(3)                     ! end-to-end vector
   real(8)    :: ree2                       ! end-to-end distance squared
   real(8)    :: rg2                        ! radius of gyration squared
  real(8)   :: rg2s                  ! square extention along principal axes (smallest)
  real(8)   :: rg2m                    ! square extention along principal axes (middle)
  real(8)   :: rg2l                   ! square extention along principal axes (largest)
  real(8)   :: rg2z                 ! radius of gyration squared projected on the z-axis
  real(8)   :: rg2xy              ! radius of gyration squared projected on the xy-plane
  real(8)   :: lpree                 ! persistence length based on end-to-end separation
  real(8)   :: lprg                    ! persistence length based on radius of gyration
  real(8)   :: shape              ! square end-to-end distance / square of radius of gyration
   real(8)    :: asph                       ! asphericity (JCP 100, 636 (1994))
   real(8)    :: torp                       ! toroidicity
end type chainprop_var
```

network properties

```
type networkprop_var
   real(8)    :: ro(3)                 ! center of mass
   real(8)    :: rg2                   ! radius of gyration squared
  real(8)   :: rg2x              ! radius of gyration squared projected on the x-axis
  real(8)   :: rg2y              ! radius of gyration squared projected on the y-axis
  real(8)   :: rg2z              ! radius of gyration squared projected on the z-axis
  real(8)   :: rg2s               ! square extention along principal axes (smallest)
  real(8)    :: rg2m                ! square extention along principal axes (middle)
  real(8)   :: rg2l                ! square extention along principal axes (largest)
  real(8)   :: eivr(3,3)            ! normalized eigenvectors of the principal frame
  real(8)   :: theta(3)          ! angles of axes of largest extension and x-, y-, and z-axes of
       main frame
   real(8)    :: asph                  ! asphericity (JCP 100, 636 (1994))
   real(8)    :: alpha              ! degree of ionization (for titrating systems)
end type networkprop_var
```

simple averaging

```
type aver_var
   real(8)      :: s2                  ! summation/averaging over steps
   real(8)      :: s1                  ! summation/averaging over macrosteps
end type aver_var
```

complexation

```
type cluster_var
   integer(4), allocatable :: ip
   integer(4), allocatable :: np
end type cluster_var
```

blocks in chains

```
type :: block_type
   integer(4)  :: pt  !particle type
   integer(4)  :: np  !number of particles
end type block_type
```

scalar quantities

```
type scalar_var
   character(40) :: label                                  ! label
   real(8)       :: norm                                   ! normalization factor
   integer(4)    :: nsamp1                                ! number of macrosteps sampled
  integer(4)   :: nsamp2                         ! number of values sampled per macrostep
   real(8)       :: avs1                                   ! average of the run
   real(8)      :: avsd                              ! precision of average of the run
   real(8)       :: avs2                                   ! average of a macrostep
   real(8)       :: fls1                                   ! fluctuation of the run
  real(8)      :: flsd                          ! precision of fluctuation of the run
   real(8)       :: fls2                                   ! fluctuation of a macrostep
   real(8)       :: value                                  ! value of a configuration
   integer(4)    :: nsamp                                  ! number of samplings
  integer(4)    :: nblocklen                    ! for sampling with variable blocklen
  integer(4)   :: nblock(mnblocklen)             ! for sampling with variable blocklen
  real(8)      :: av_s1(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: av_sd(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: av_s2(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: fl_s1(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: fl_sd(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: fl_s2(mnblocklen)              ! for sampling with variable blocklen
  real(8)      :: av_sd_extrap                   ! for sampling with variable blocklen
  real(8)      :: av_sd_stateff                  ! for sampling with variable blocklen
  real(8)      :: fl_sd_extrap                   ! for sampling with variable blocklen
  real(8)      :: fl_sd_stateff                  ! for sampling with variable blocklen
end type scalar_var
```

1D distribution functions

```
type df_var
   character(27) :: label                                  ! label
   real(8)       :: norm                                   ! normalization factor
   integer(4)    :: nsamp1                                ! number of macrosteps sampled
  integer(4)   :: nsamp2                         ! number of values sampled per macrostep
   real(8)       :: min                                    ! minimum value of df
   real(8)       :: max                                    ! maximum value of df
   integer(4)    :: nbin                                   ! number of grid points
   real(8)       :: bin                                    ! grid length of df
   real(8)       :: bini                                   ! inverse of bin
  real(8)     :: nsampbin(-1:mnbin_df)        ! number of values sampled in each bin during
      macrostep
   real(8)       :: avs1(-1:mnbin_df)                      ! average of the run
   real(8)      :: avsd(-1:mnbin_df)                 ! precision of average of the run
   real(8)       :: avs2(-1:mnbin_df)                      ! average of a macrostep
end type df_var
```

2D distribution function

```
type df2d_var
   character(27) :: label                            ! label of df
   real(8)       :: norm                             ! normalization factor
   integer(4)    :: nsamp1                          ! number of macrosteps sampled
  integer(4)   :: nsamp2                   ! number of values sampled per macrostep
   real(8)       :: min(2)                           ! minimum value of df
   real(8)       :: max(2)                           ! maximum value of df
   integer(4)    :: nbin(2)                          ! number of grid points
   real(8)       :: bin(2)                           ! grid length of df
   real(8)       :: bini(2)                          ! 1/bini
   real(8)       :: avs1(-1:mnbin_df2d,-1:mnbin_df2d) ! average of the run
  real(8)     :: avsd(-1:mnbin_df2d,-1:mnbin_df2d) ! precision of average of the run
   real(8)       :: avs2(-1:mnbin_df2d,-1:mnbin_df2d) ! average of a macrostep
end type df2d_var
```

cluster1 trial move

```
type cluster1_tm_var
   logical      :: l                      ! flag for cluster1 tm
   real(8)      :: p                      ! relative probability of cluster1 move
   real(8)      :: rad                    ! radius of region for cluster members
   real(8)     :: psel                  ! probability to select a particle within rad
   real(8)      :: dtran                  ! maximal translational trial displacement
   real(8)      :: drot                   ! maximal rotational trial displacement
end type cluster1_tm_var
```

cluster 2 trial move

```
type cluster2_tm_var
   logical      :: l                      ! flag for cluster1 tm
   real(8)      :: p                      ! relative probability of cluster2 move
   real(8)      :: dtran                  ! maximal translational trial displacement
   real(8)      :: drot                   ! maximal rotational trial displacement
   integer(4)   :: mode                   ! =0 : search members only of type iptmove
                                        ! =1 : search members across all particle types
end type cluster2_tm_var
```

trial move

```
type trialmove_var
   logical      :: l                        ! flag for type of trial move
   real(8)      :: p                       ! relative probability of type of trial move
   real(8)      :: dtran                    ! maximal translational trial displacement
   real(8)      :: drot                     ! maximal rotational trial displacement
   integer(4)   :: mode                     ! specific for type of trial displacment
   logical      :: lcl1                     ! flag for cluster1 trial move
   real(8)      :: radcl1                   ! radius of region for cluster members
   real(8)      :: pselcl1                ! probability to select a particle within rad
end type trialmove_var
```

node in the DOP-tree

```
type node                                 ! node in the DOP-tree
  !private
  real(8) :: dop(6)                     ! bounding box in object coordinate system
  integer(4) :: c(2)                    ! id:s of children
  logical :: leaf                       ! leaf node: children is triangles
end type
```

triangle mesh, with DOP-tree

```
type trimesh  ! triangle mesh, with DOP-tree
  !private
  real(8), allocatable :: c(:,:)      ! c(3,np) coordinates of triangle verticies
  integer(4), allocatable :: t(:,:) ! t(3,nt) triangles as index as verticies into c
  type(node), allocatable :: n(:)    ! nodes in tree
  integer(4) :: levels                ! levels of subdivisions of triangles
end type
```

affine transformation

```
type affinetrans                          ! affine transformation
  real(8) :: trans(3)                   ! location of object origin in lab system
  real(8) :: rot(3,3)                   ! rotation matrix applied to object
 integer(4) :: sel(3,6)         ! for each directed axis in lab system which 3 axises in object system
      contribute positively
end type
```

sso step

```
type :: step
   integer(8)  :: n  !number of steps
   real(8)     :: d2  !squared displacement
   real(8)     :: d4 !displacement**4
end type step
```

SSOPart

```
type  :: ssopart_var
   real(8)     :: fac      !increment of part length
   integer(4)  :: nextstep !step at which next part starts
   integer(4)  :: i        !current part
   integer(4)  :: n        !number of parts
end type ssopart_var
type(ssopart_var), save  :: SSOPart
```

SSOParameters

```
type  :: ssoparam_var
   real(8)     :: used     ! used dtran
   real(8)     :: opt      ! dtran with the highest mobility
   real(8)     :: err   ! accuracy of opt
end type ssoparam_var
type(ssoparam_var), save, allocatable  :: SSOParameters(:,:)
```

sso mobility

```
type  :: mobility_var
   real(8)     :: val         !value
   real(8)     :: error       !error
   real(8)     :: smooth      !smooth
end type mobility_var
type(mobility_var), allocatable, save :: Mobility(:)
```

celllist cell-pointer-array

```
type cell_pointer_array
  type(cell_type), pointer         :: p => null()     ! pointer to a cell, usefull to create an
       array of pointers
end type cell_pointer_array
```

celllist cell-type

```
type cell_type
   integer(4)                          :: id                 ! for easy recognition
   integer(4)                     :: npart           ! number of particles per cell
   integer(4)                     :: nneighcell       ! number of neighbouring cells
  type(cell_pointer_array), allocatable :: neighcell(:)      ! pointer to the neighbouring cells
  integer(4)                    :: iphead          ! first particle in the linked list
end type cell_type
```

8 Variables

A brief explanation of most global variables is found in file mol.F90. Beside that, the following convention of indices is used.

Indices	Description
ic, jc	Chain number
ip, jp	Particle number
ia, ja	Atom number
ict, jct	Number of chain type
ipt, jpt	Number of particle type
iat, jat	Number of atom type
ictjct	Chain type-chain type pair number
iptjpt	Particle type-particle type pair number
iatjat	Atom type-atom type pair number
iploc	Particle number (local)
ialoc	Atom number (local)

9 Additional Descriptions

9.1 Networks

Overview

Non-periodic network structures in Molsim

The aim of this branch is the "proper" implementation of polymer networks as an integral part of the simulation software Molsim. These networks are called non-periodic networks in order to distinguish them from their macroscopic counter parts (i.e. macrogels). This follows Molsim's philosophy of having different objects in a hierarchy of different levels.

Object type	Hierarchy Level	Object type	Hierarchy Level
Atoms	1		
Particles	2		
Chains	3		
Hierarchical Structures	4	Non-periodic Network Structures	4

Within this scheme, atoms are part of particles, which in turn may be connected to form a chain. These can be interconnected by cross-links to form more complex structures, as e.g. hierarchical structures or network-like structures. Note, that for all object types on the left hand side of the table, a programming infrastructure in form of various characteristic parameters and (pseudo-)pointers is established already, thus, leading to a high programming flexibility, whenever one wishes to expand the functionality of the software with respect to one of these object types. On the other hand, non-periodic network structures have been introduced to Molsim later and therefore a programming infrastructure is not given. Besides the mentioned characteristic parameters and (pseudo-)pointers, a series of subroutines in Molsim handles the statistical treatment of the left hand side object types. Within the scope of this branch the corresponding parameters and functionalities shall be established.

Model description

The aim of the implemented network model is to simulate the properties of non-periodic polymer networks of spherical shape (i.e. microgels). The model network comprises cross-linking particles (so called nodes) and chain particles. The nodes are set on the carbon positions of the diamond lattice (space group Fd3m, no. 227). The cubic lattice is cropped to obtain a spherical shape by defining a cutoff radius - all nodes within the cutoff radius will be kept; all nodes outside of the cutoff radius will be discarded. The remaining nodes are then connected via polymer chains, which consist of a number of chain particles per chain. Within the model the so called dangling chains are included. They are connected to only one cross-link and thus they form the outer periphery of the modelled gel. Via the input parameter `nnwt` it is possible to define the number of network types present in the system. Different network types may for example differ in their number (`nnwnwt(inwt)`) and size (`rnwt(inwt)`). Further, different types of particles may be used as building blocks and hence, polymer networks with different physical properties may be obtained.

Within this scheme it is important to note, that always one particle type forms the nodes of one certain network type. Analogues, one chain type forms the chains of a certain network type. This is controlled via the parameters `iptclnwt(inwt)` (i.e. the particle type forming the cross-links of network type `inwt`) and `ncctnwt(ict,inwt)` (number of chains of a chain type `ict` belonging to one network of network type `inwt`). The size of the networks cannot be chosen in a continous manner, as the nodes are set on discrete diamond lattice positions, hence the generated networks are available with discrete numbers of nodes.

Details of implementation

Network Configuration

General

The possibility to set finite networks has already been possible prior to the here undertaken implementations by means of the subroutine `SetNetwork`. Within the branch Networks the already existing routine `SetNetwork` has been adapted to the new nomenclature regarding network related parameters (see below). Both, in the subroutine `SetNetwork` and in the general output about the system parameters, the output about networks has been refined.

Shift of cropping sphere

The parameter `shiftnwt(1:3,1:nnwt)` was introduced. It enables one to obtain topology-modified networks, which mainly differ in how the structure (*i.e.* dangling chains and interconnectivity of the outer layer) of their periphery looks like. In the process of generating the network structure a cubic diamond lattice will be span and cropped by a spherical cut-off. The default network will be set having the (0,0,0) position of the diamond lattice unit cell in the center of the cropping sphere. This corresponds to `shiftnwt(1↩ :3,inwt) = 0.0` of a certain network type `inwt`. By setting 'shiftnwt(1:3,inwt)' the center of the cropping sphere may be shifted to any point of the unit cell. Note, that `shiftnwt` accepts its values in the range of 0.0 to 1.0, respectively. Higher values will work aswell, but due to the underlying periodic boundary conditions which the diamond lattice is subject to, `shiftnwt(1:3,inwt) = 0.0` is equivalent to `shiftnwt(1:3,inwt) = 1.0` and so on.

Network Properties

Similar to the possibility to evaluate characteristic properties of chain objects in Molsim, a new subroutine has been implemented, namely `CalcNetworkProperty`. Within the subroutine `CalcNetworkProperty` the following properties are evaluated:

- `ro(3)`: center of mass
- `rg2`: radius of gyration squared
- `rg2x`: radius of gyration squared projection on x-axis
- `rg2y`: radius of gyration squared projection on y-axis
- `rg2z`: radius of gyration squared projection on z-axis
- `rg2s`: square extention along principal axes (smallest)
- `rg2m`: square extention along principal axes (middle)

- `rg2l`: square extention along principal axes (largest)
- `asph`: asphericity (JCP 100, 636 (1994))
- `theta(3)`: angle between x-, y- and z-axis, respectively, and the axis of the network's largest extension
- `alpha`: degree of ionization (for titrating systems)
- `eivr(3,3)`: normalized eigenvectors of the principal frame

Averages of Network Properties

Within the subroutine `NetworkAver` the averages and precisions with regards to the properties available by `CalcNetworkProperties` are formed (except for `ro(3)` and `eivr(3,3)`). The obtained network quantities will be written to the Output-File when networks are present in the system. Example output:

```
*************************************************************************************
*                                   network quantities                              *
*************************************************************************************
quantity                        average  precision  fluctuation  precision  stat eff
--------                        -------  ---------  -----------  ---------  --------
<r(g)**2>**0.5               =  75.32408   0.00915      1.77217    0.06023        2.
<r(g)**2_x>**0.5             =  43.51678   0.00452      1.36440    0.09226        0.
<r(g)**2_y>**0.5             =  43.49056   0.00458      1.37475    0.14198        0.
<r(g)**2_z>**0.5             =  43.45776   0.01588      1.66017    0.01477        3.
smallest rms mom. p.a.       =  43.39747   0.01547      1.77924    0.10399        2.
intermediate rms mom. p.a.   =  43.47212   0.01324      1.65132    0.11078        2.
largest rms mom. p.a.        =  43.59531   0.01354      1.74615    0.07283        1.
<asphericity>                =   0.00001   0.00000      0.00000    0.00000        1.
<xtheta>                     =  72.47060   8.72678     44.88033    3.26752        0.
<ytheta>                     = 101.41857   3.97221     38.12409    4.66430        0.
<ztheta>                     =  78.86201   2.54458     19.66774    2.88301        0.

asphericity (<2:nd moments>) =   0.00001
```

Here, the asphericity is being calculated according to `Zifferer, G. and Olaj, O. F., Journal of Chemical Physics 1994, 100, 636`. The second displayed asphericity (`asphericity (<2:nd moments>)`) is being directly calculated from the respective average of the principal moments of a macrostep or the whole simulation.

Network Distribution Functions

The functionality of finite network-related simulations has been extended by allowing for statistical analysis of network properties by means of distribution functions.

Network Property Distribution Functions

The following network property distribution functions are available by means of the setting in the `nml NetworkDF`.

type	label	description
1	`rg`	network radius of gyration distribution
2	`asph`	network asphericity distribution
3	`alpha`	network degree of ionization distribution
4	`rgchain`	chain radius of gyration distribution of network chains
5	`reechain`	chain end-to-end distance distribution of network chains

The described type `i` refers to the array index to be used in the respective `vtype(i)` construct.

Radial Network Distribution Functions

The following radial network distribution functions are available by means of the setting in the `nml`↩ `NetworkRadialDF`.

type	label	description
1	`rpart`	radial particle number distribution
2	`rdens`	radial particle density distribution
3	`rgchain`	radial chain radius of gyration
4	`q`	radial sum of all charges
5	`qcum`	radial cumulated sum of all charges
6	`alpha`	radial degree of ionization
7	`rchain`	radial chain number distribution

The described type `i` refers to the array index to be used in the respective `vtype(i)` construct.

Network Generation Groups

A new way of forming groups within networks has been established. Within this grouping scheme the chains are numbered starting from the dangling chains at the periphery of the network and taking the shortest way to the core chains of the network structure. A schematic representation of the numbering is presented below.

```
          | 1
          o
          | 2
          o
1   2   3 | 3   2   1
- o - o - o - o - o -
          | 3
          o
          | 2
          o
          | 1
```

Note, that this scheme works only for structures set with ‘txsetconf = 'network’. The cross-links of the network will be assigned to have no group. In order to obtain this grouping set eitherreforfieldin thenmlGrouptonetworkgenerations`.

Testin

fnw.nwt2.mc.in

The testin-file was added to the in-directory. The corresponding out-file was stored in the Save-directory. The documents "todiff.txt" and "goall.sh" were expanded by the new input-file.

fnw.nwgen.df.mc.in

The testin-file was added to the in-directory. The corresponding out-file was stored in the Save-directory. The documents "todiff.txt" and "goall.sh" were expanded by the new input-file.

Network variables

Internal Variables

Besides of the external network related parameters (i.e. input variables) a number of internal variables has been introduced in order to set up the programming infrastucture of non-periodic networks. All variables are declared in `mol.F90`.

Keyword	Type	Description
`nnw`	`integer`	Total number of networks
`nctnwt(1:nnwt)`	`integer`	Number of different chain types in network type `inwt`
`ncnwt(1:nnwt)`	`integer`	Number of chains per network of network type `inwt`
`npnwt(1:nnwt)`	`integer`	Number of particles per network of network type `inwt`
`nclnwt(1:nnwt)`	`integer`	Number of cross-links per network of network type `inwt`
`nnwtnwt`	`integer`	Number of different network type pairs
`txnwtnwt(1:nnwtnwt)`	`character(21)`	Name of network type pair `inwtnwt`
`massnwt(1:nnwt)`	`real`	Mass per network of network type `inwt`
`massinwt(1:nnwt)`	`real`	Inverse mass per network of network type `inwt`
`lnetwork`	`logical`	`.true.`if networks are used
`lptnwt(1:mnpt,1:nnwt)`	`logical`	`.true.` if particle type `ipt` is part of networks of network type `inwt`
`lpnnwn(1:np,1:nnw)`	`logical`	`.true.` if particle `ip` is part of network `inw`
`npweakchargenwt(1:nnwt)`	`integer`	Number of titratable charges per network of network type `inwt`

Network pointer

Keyword	Type	Description
`inwtnwn(1:nnw)`	`integer`	In: network number `inw`, Out: its network type `inwt`
`inwtct(1:nct)`	`integer`	In: chain type `ict`, Out: its network type `inwt`
`inwtcn(1:nc)`	`integer`	In: chain number `ic`, Out: its network type `inwt`

Keyword	Type	Description
`inwncn(1:nc)`	`integer`	In: chain number `ic`, Out: its network number`inw`
`inwnnwt(1:nnwt)`	`integer`	In: network type `inwt`, Out: its first network `inw`
`inwtnwt(1:nnwt,1:nnwt)`	`integer`	In: two network types `inwt/jnwt`, Out: its network type pair number `inwtnwt`
`icnclocnwn(1:ncnwt,1:nnw)`	`integer`	In: local chain number `icloc` and newtwork number `inw`, Out: its chain number `ic`
`ipncllocnwn(1:nclnwt,1:nnw)`	`integer`	In: local cross-link number `iclloc` and network number `inw`, Out: its particle number `ip`
`ipnplocnwn(1:npnwt,1:nnw)`	`integer`	In: local particle number `iploc` and network number `inw`, Out: its particle number `ip`

9.2 Complexation-Analysis

Complexation Analysis

Interparticle complexation can be analyzed using Molsim. To use it set

```
lstaticuser = .true.
```

in `nmlStatic`. Additionallly the first 4 characters of `txuser` in `nmlSystem` have to be `comp`. The control of the complex-statistics is given by the namelist `nmlComplexation`:

```
&nmlcomplexation
 rcut_complexation = ...,
 lcomplexfraction = ....,
 lclusterdf = ...,
/
```

control over the ComplexDF routine is given by the namelist `nmlComplexDF`:

```
&nmlcomplexdf
  vtype = ...
/
```

where `rcut_complexation` defines the maximum distance for two particles to form a complex; `l↩ ComplexFraction` (`logical`) switches on the calculation of the fraction of complexed particles; `l↩ ClusterDF` (`logical`) switches on the calculation of the size distribution of complexed formed by the complexation.

- Testfile: `complexation.mc.in`

Fraction of Complexation

This routine return the fraction of complexed particles of each particle-particle commbination. It return the fraction of `X` beads which are closer than `rcut_complexation` to a `Y` bead as `w(cmplx): X - Y`. E.g.:

```
quantity           average  precision  fluctuation  precision  stat eff
--------           -------  ---------  -----------  ---------   -------
w(cmplx): a - a    0.30000    0.00000      0.00000    0.00000        0.
w(cmplx): a - b    0.20000    0.00000      0.00000    0.00000        0.
w(cmplx): b - a    0.40000    0.00000      0.00000    0.00000        0.
w(cmplx): b - b    0.50000    0.00000      0.00000    0.00000        0.
```

means that 30% of the A beads are complexed to other A beads; 20% of the A beads are complexed to a B bead; 40% of the B beads form a complex with an A bead and 50% of the B beads form a Complex with another B bead.

Complexation Distribution

This routine return the distribution of the fraction of complexed particles of each particle-particle commbination. It return the fraction of X beads which are closer than `rcut_complexation` to a Y bead. To control of the distribution function the `nmlComplexDist` is used. Additionally also the distribution of the complexation rate of the individual chains can be calculated.

Segment Complexation

This routine return the fraction of complexation of each segment along the chains of the system (averaged over all chains of one type).

Size distribution of the binary clusters

Here the size of clusters formed by neighbouring complexed particles of two different types are measured. A cluster is formed only by connections between particles of different particle type. E.g.

```
A   A
    |
    B
    |
    A
    |
A - B
```

forms one cluster of size 5 and one of size 1 (single uncomplexed A particle). In contrast

```
A - B
    |
    A

    A
    |
A - B
```

forms two clusters of size 3.

As an output the size distribution (average number of cluster of a specific size) are given for all combinations of two different particle types.

9.3 Advanced Configurations

The support for branched, random and repeating structures is improved:

Branched Structures

- When setting a hierarchical structure using the subroutine `SetHierarchical` the inability to set one particle will not lead to an abortion of the whole program. Instead another it is attempted `ntrydef` (default 100) times to set the whole structure. If one particle cannot be set the whole structure is reset.
- Testfile: `hierarchical.mc.in`

9.4 Copolymer Sequence

Overview

In general chains consisting of more than one particle type (i.e. copolymers) may be simulated using Molsim. Different ways to generate such chains are triggered by the parameter `txcopolymer`. Existing modes comprise alternating, and block copolymers. Within the scope of this branch the functionality to simulate copolymers with random, repeating or irregular or highly specific monomer distribution has been added (sequence).

Usage

Sequence

The input of the copolymer sequence has been realized by introducing the parameter `iptsegct(iseg,ict)`, which specifies the particle type `ipt` of chain segments `iseg` of chains of type `ict`. First of all the parameter `txcopolymer(ict)` has to be set to `sequence` for the chain type in question in the namelist `nmlParticle`:

```
&nmlParticle
 ...
 txcopolymer(ict) = 'sequence' ,
 ...
/
```

where `ict` should be replaced by the corresponding chain type number. The parameter `iptseg(iseg,ict)` is then being set in the namelist `nmlCopolymerSequence`. For example

```
&nmlCopolymerSequence
 iptsegct(1,ict) = ipt , iptsegct(2,ict) = ipt , ... , iptsegct(npct,ict) = ipt ,
 ...
/
```

where again `ict` should be substituted by the corresponding chain type number, `ipt` should be replaced by the intended particle type number and `npct` corresponds to the number of particles per chain of chain type `ict`.

Repeating Copolymers

Copolymers with a defined repeating block structure can be generated. It consists of repeating units, each consisting of blocks of one particle type.

- in `nmlParticle` set the parameters of the chain type `ict` which is to be repeating

```
txcopolymer(ict) = "repeating",
```

and set

```
nblockict(ict)
```

to the number of blocks within a repeating unit to the intended number.

- The detailed repeating structure is given by `nmlRepeating`:

```
&nmlrepeating
rep_iblock_ict( iblock , ict)%pt = ... ,
rep_iblock_ict(iblock , ict)%np = ... ,
/
```

where the particle type `pt` and number of particles `np` of each `iblock` for each`ict` is defined.

If not enough particles of a certain type are given for a chain to fulfill the regular structure, the repeating structure is continued, but only filling the blocks with the remaining particles.

Example: One chain with the following input (excerpt)

```
&nmlparticle
 nct   = 1,
 ncct  = 1,
 npptct(1:3,1) = 4,5,10,
 nblockict(1) = 3,
 txcopolymer='repeating',
/
&nmlrepeating
rep_iblock_ict(1:3,1)%pt = 2, 3, 1
rep_iblock_ict(1:3,1)%np = 2, 3, 1
/
```

will generate such a repeating structure:

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Particle Number	5	6	10	11	12	1	7	8	13	14	15	2	9	16	17	18	3	19	4
Particle Type	2	2	3	3	3	1	2	2	3	3	3	1	2	3	3	3	1	3	1
Repeating Block	1		2			3	1		2			3	1	2			3	2	3
Repeating Unit	1						2						3					4	

Random Copolymers

Random copolymers can be generated. The random sequence is generated by first creating a list where the particle type of each segment is given as for a block-like copolymer. This list is then shuffled using a `Knuth Shuffle`. Now the particles are sequentially assigned to segments of its type. This way the particle types are randomly distributed, but the particles of one type are of increasing index along the chain. Note that if you want the exact sequence printed in your output file you need to set `itestpart = 10` in the `nmlParticle`.

- in `nmlParticle` set the parameters of the chain type `ict` which is to be random:

```
txcopolymer(ict) = "random",
```

Testin

The Testin-file covering these functionalities are `chain.sequence.fnw.mc.in` and `copoly-seq.mc.↩ in`.

9.5 Celllist

Overview

The cell list is implemented in molsim acts a a neighbor list, so that fewer particle pairs with a distance larger than r_{cut} are considered. This allows for much quicker simulations, especially for large systems with short ranged interactions. The simulation duration scales with $\mathcal{O}(n)$. The advantages are compared to the other neighbor lists are the following:

- The cell list is updated after every accepted step, so that is is always up to date. This allows for non-local moves to be carried out without needing to artificially enlarge the cell size. This is not the case for `vlist` and `llist` which are currently implemented in molsim.
- The cell structure is generated only once at the beginning of the simulation. The time for the generation of the cell list scales with the number of cells, but the execution speed should be independent of the number of cells. In `llist` the number of cells were capped to a small number (1000), so that large systems had to use large cells. Increasing the number of possible cells did not help either, as the time to find the neighbors in each step increased with the number of cells, making the `llist` very slow when having many cells ($\approx 10^6$).

Details of Implementation

Working Procedure

The usage of a neighbor list within Molsim is controlled by the namelist `nmlIntList`. Usage if the cell list is achieved by setting `txintlist = 'cellList'`. The cell size is the chosen to be as small as possible to fill the whole space evenly with cells, while still remaining larger than `rcut`.

One can create cells of different size, by changing the value of `drnlist`, which will be added to `rcut` for setting up the cell list. Most notably one can set `drnlist` to a negative value to even further reduce the cell size. This will lead to less particles with a distance larger than `rcut`, but more cells will be considered. Whether or not this approach increases the speed of the simulation may depend on the system.

All parameters except `txintlist` and `drnlist` of `nmlIntList` are not used for the cell list. A typical input file could be the following:

```
&nmlintlist txintlist = 'cellList', drnlist = 0.0, /
```

Testin

An example for the usage of the cell list can be found the `Testin` directory.

- `lj.mc.clist.in`

Further Reading

- cell List https://en.wikipedia.org/wiki/Cell_lists

9.6 SSO

Overview

SSO

In this section the sso-functionality is added to Molsim. For a detailed description of the algorith see the master-thesis of pascal or the corresponding publication. The SSO algorithm allows for on the fly optimization of the displacement parameter of the single particle move during the equilibration run.

Sperarate Local from Non-Local

In addition the `lmcsep` parameter was introduced. This separates local (Single-Particle and SSO Move) from non-local moves (all the others) during the simulation. During each MC-Pass it is evaluated, wheter non-local moves are to be carried out (accoring to their probabiliy) and only if they will be carried out the neihbour list ist updated with a large drnlist ($4*$ contou length of the longest chain). This allows for using a tigh neighbour-list during the simulation with local-moves.

Testin

The following test-projects are added to the Testin directory:

- `sso.sso.in`: Test of the sso-move
- `sso.lmcsep.in`: Test of the lmcsep-setting

9.7 Image generation with VMD

Overview

In order to generate images for publications, presentations and for visual insight in simulated systems, Molsim offers the possibility to generate image files in the `VTF format ("VTF Trajectory Format")`. The VTF format has specifically been designed for people working with their own simulation code with the aim of visualization via the image software `VMD ("Visual Molecular Dynamics")`. VMD offers a broad spectrum of functionality with regards to the visualization of molecular systems with the possibility of rendering high quality images.

Details of Implementation

File Types

The visualization of simulation snapshots requires two different file types.

- `vtf`: The vtf file contains atom, bond and box length definitions, followed by an arbitrary number of coordinate blocks.
- `tcl`: The tcl file forms the interface between Molsim and VMD. The whole settings of VMD may be controlled via tcl scripts. Here, color specifications, additional geometric objects (*e. g.* simulation cell) and the loading of the vtf file is stored.

Working Procedure

The generation of images with Molsim is controlled by the namelists `nmlSystem`, `nmlImage` and namelist `nmlVTF/nmlVRML`. Note, that this manual covers only the image generation using the `vtf` format for the visualization with VMD. For the generation of images in the `vrml` format, please confer the MOLSIM manual. In the following the work flow for the image generation is described.

- Request the generation of images by setting `limage = .true.` in the namelist `nmlSystem`

```
&nmlsystem limage = .true.,/
```

- Request the generation of images using the `vtf` format by setting `lvtf = .true.` in the namelist `nmlImage`

```
&nmlimage lvtf = .true.,/
```

- Further specifications may be given in the namelist `nmlVTF`. With just the default settings, images of the initial and of the final configuration will be stored, respectively.
- Run the simulation
- Open the `vtf` image file by executing the accompanying `tcl` script in VMD

```
vmd -e {name-of-project}.tcl
```

- Apply individual modifications to representation

Render a snapshot

Here are some tipps on how to generate nice images using `VMD`.

- use the `AOShiny` material
- turn on the lights, often the scene generated looks less bright when rendered as presented one the screen in `VMD`

```
light 2 on
light 3 on
```

- turn on the shadows with `display shadows on`
- turn on the ambientocclusion `display ambientocclusion on`
- use the GLSL rendermode `display rendermode GLSL`

Then render the scene (with the path to you Tachyon executable adapted

```
render Tachyon scene.dat
exec "<path to you Tachyon executable>" -aasamples 48 scene.dat -res 2096 1048 -format BMP -o scene.b
```

You might want to adapt the name of the image (`scene.*`), the resolution and the the aasamples.

Testin

An example for the usage of the image generation in the `vtf` format may be found in the `Testin` directory.

- vtfimage.fnw.mc.in

Further Reading

- VTF https://www.github.com/olenz/vtfplugin/wiki/VTF-format
- TCL https://www.tcl.tk
- VMD http://www.ks.uiuc.edu/Research/vmd

10 Developing

When developing Molsim, please adhere to the contributing guide, which is shipped in the MOLSIM repository.

10.1 Testin

In the Molsim directory you will find a directory called `Testin`. The Testin offers an efficient way to check whether changes of the Molsim source code lead to any malfunction of the program.

General idea

The general idea of the Testin-directory is to account for the sustainment of the functionality of Molsim. Whenever the source code of Molsim is changed, one has to ensure that none of the functionalities of Molsim have been broken. Despite of a few aspects (e.g. format changes) the program execution should yield consistent results. By comparing the results of test simulations of the current with the previous version it is easy to detect whatever kind of malfunctions.

The Testin-directory provides a method to compare the output of the current version to the last stable version.

Usage

- **First you need to generate the output from the stable version.** The stable version should be the latest `master` branch from which your branch diverged. If you are unsure, check when the file `Testin/stable.md5sum` was last changed. The repository at this stage is the latest stable version. When you have checked out the stable version you want to compare with, run `make stable` in the `Testin` directory. This will compile the current source code with the `mode = test` flag and create the output in the `Testin/out_stable` directory.
- at any time you can run `make out` to run the same input files and store the output in the `Testin/out`.
- to compare the output of the `out` and `out_stable` directories, run `make diff`. Note that if the out files are not present, `make diff` will create them accordingly. The collected differences are written into the file `diff.out`

The `diff` should, in general, yield no differences. If differences were found, one needs to carefully inspect each of these instances for whether one deals with expected changes or unexpected changes.

- *Unexpected changes: Investigate where the difference is coming from and fix the bug!*
- Expected changes: When you have only expected changes, add comments to the `diff.out` file, which describe the changes. This commented version of the `diff.out` files should be attached to the pull request when you want to merge into the stable version. Note that the `diff.out` file will be overwritten when running `make` so you might want to add you comments to a copy of `diff.out`

- After your merge request has been positively reviewed, run `make declarestable` to declare your current version of the code as the stable one. The files included in the check for the stable version are all `.F90` files, the `makefile` and the `molsim_ser` file in the `Src` directory. Additionally also the `configure.sh` file in the molsim root is checked.
- to delete all output of the `out` runs (and the `diff.out` file!) run `make clean`, to delete the `out_↩ stable` dir run `make cleanstable`. To delete both run `make cleanall`.

Further notes:

- You can speed up the process, by running make in parallel (e.g. use `make -j 4`, to run on 4 cores).
- When running `make` all needed files are generated automatically, so you do not need to run `make out` before running `make diff`. A single call of `make` is enough.
- The `out_stable` files can not be created automatically, when you are in the directory of a modified version of molsim. Checkout the stable version, run `make stable`, and go back to your branch. The `out_stable` dir should stay, and can be used to compare with.

List of elements contained in the Testin-directory

element	description
`in`	input files covering all functionalities of Molsim
`todiff.txt`	all output files to diff
`Makefile`	Makefile to run all tests
`scripts/diff1.sh`	script to compare the output files
`scripts/molsim_stable.sh`	script to generate outputs from the stable version
`stable.md5sum`	file defining the current stable version

After executing `make` you will additionally find:

element	description
`out`	output files resulting from the Molsim version under examination
`out_stable`	output files corresponding to the input file in `in`, created with the latest stable version.
`diff.out`	Output of the diff between the current and the stable versions

10.2 Random Numbers

The random number generator used in Molsim is `rand` function as described by Press et al. in Numerical Recipes in Fortran 77, *The Art of Scientific Computing*, Second Edition, 1997. It has a a period of $\approx 3.1 \times 10^{18}$ and returns a uniform random number in the range (0.0, 1.0).

The routine uses a pair of of integers (`ixseed` and `iyseed`) to generate random numbers. The `iseed` you give in the `nmlSystem` is used to generate the first integer number pair. When `iseed` is negative the current time is used to set `iseed`. In most cases it is sufficient to set only `iseed`. The case where you also want to set `ixseed` and `iyseed` are when you want to use a specific random number sequence from a previous simulation.

- You want to set `ixseed` and `iyseed` to some specific values:
  - In `nmlSystem` set `iseed`, `iyseed` and `ixseed` to the desired values and set `luseXYseed` to `.t.`.
  - Note that if you set `iseed` to a negative value, then the seed will not be set by the cpu time. A negative seed will reinitialize the random number generator and overwrite `ixseed` and `iyseed`.
- Given an output file where you want to recreate the random numbers you have three possibilities:
  - If you want to generate a new simulation with the same initial seed: set `iseed` to the value of the seed given at the beginning of the input file
  - If you want to generate a new simulation with the same initial seed of the previous simulation which was started by setting `iyseed` and `ixseed` explicitly: set `iseed`, `iyseed` and `ixseed` to the ones given in the beginning of the output file set `luseXYseed` in `nmlSystem` to `.t.`.
  - If you want to run a simulation, continuing with the same chain of random numbers where a the previous simulation left of: set `iseed`, `iyseed` and `ixseed` to the ones given in the end of the output file set `luseXYseed` in `nmlSystem` to `.t.`.

10.3 Compilation Modes

To make the development easier, several compilation modes are provided. They are used as following:

- when making Molsim a second argument can be massed to make: `mode=<mode>`. Example:

```
make ser mode=test
```

to compile Molsim in the test mode. The following modes are supported:

- `normal`: compile with normal options to achieve high computational efficiency.
- `test`: compile in such a way, that the results are reproducible. Use this mode when performing tests in the Testin directory.
- `debug`: During run-time the compiler checks for out-of-bounds arrays and uninitialized variables. Only use for debug purposes.
- `quick`: Fast compilation without optimization. Use only for testing purposes.
- `warn`: The compiler gives all warnings during compile time.
- `gprof`: For profiling the execution with gprof (only for the `gfortran` compiler)

Gprof

To profile you program with `gprof`, do the following (only for the `gfortran` compiler):

- Compile with `mode=gprof`:

```
make ser mode=gprof
```

- run Molsim on you project in the directory of you input file. A file `gmon.out` should be generated

```
molsim_ser <Projectname>
```

- run `gprof`

```
gprof <path to the molsim exe> gmon.out
```

11 Appendix

11.1 Namelist

– A namelist consists of (i) one start statement, (ii), one end statement and (iii) a list (possibly empty) between the start and end statements.

– The start statement consists of an ampersand (&) followed by the name of the namelist.

– The end statement consists of a slash (/).

– The list is composed of variables and their values separated with either by commas.

– The order of the variables and their assignments is normally irrelevant.

– However, if a variable is assigned values more than once, the last one takes place.

– Arrays are allocated either element by element, consecutively by listing values separated by commas, or by a combination of these to ways.

– A value, say 5, appearing r times may be expressed as `r*5`.

– Note the order Fortran stores arrays elements: the left most index is running fastest and the right most slowest.

– Comments are allowed and should proceed by an exclamation sign (!).

– Generally, if a program reads several namelists, they have to occur in the inputfile in the same order as they are read from the program. Here, the order of the namelist is arbitrary.

– A namelist with no list (empty namelist) has to be specified.

– Namelists that are not read by the program do no harm.

Example: Consider the following declarations

```
character(4) :: title
integer(4) :: m, n
real(8) :: arr1(1:3), arr2(1:2,1:2), arr3(1:2,1:2)
```

The namelist `nmlSystem` below illustrates an element-by-element array assignment

```
&nmlSystem
title = 'head',
n = 10, m = 10,
n = 20,
arr1(1) = 1.5, arr1(2) = 2.5, arr1(3) = 2.5,
arr2(1,1) = 1.0, arr2(2,1) = 2.0, arr2(1,2) = 3.0, arr2(2,2) = 4.0,
arr3(1,1) = 1.0, arr3(2,1) = 2.0, arr3(1,2) = 3.0, arr3(2,2) = 4.0,
/
```

which could, for example, be shortened to

```
&nmlSystem
title = 'head',
n = 20,
m = 10,
arr1 = 1.5, 2*2.5,
arr2 = 1.0, 2.0, 3.0, 4.0,
arr3(1:2,1) = 1.0, 2.0, arr2(1:2,2) = 3.0, 4.0,
/
```

11.2 Examples of namelists describing different objects

Examples of namelist `&nmlParticle` describing different types of objects are given in this appendix.

Hard-sphere particles

```
&nmlParticle
            beginning of the namelist
npt = 1,
            number of particle types
txpt = 'hs',
            particle's label
nppt = 100,
            number of particles of this type
natpt = 1,
            number of atom types which this particle is composed of
txat = 'hs',
            atom's label
massat= 1.0,
            molar mass of the atom (in grams per mole)
radat = 1.0,
            hard-sphere radius of the atom
naatpt(1,1) = 1,
            number of 'hs' atoms in the 'hs' particle
txaat(1,1) = 'hs',
            label of 'hs' atom in the 'hs' particle
/
            end of the namelist
```

Lennard-Jones particles

```
&nmlParticle
            beginning of the namelist
npt = 1,
            number of particle types
```

`txpt = 'lj',`
  particle's label

`nppt = 500,`
  number of particles of this type

`natpt = 1,`
  number of atom types which this particle is composed of

`txat = 'center',`
  atom's label

`massat= 40.0,`
  molar mass of the atom

`radat = 1.0,`
  hard-sphere radius of the atom

`sigat = 3.405,`
  Lennard-Jones parameter 'sigma'

`epsat = 0.99606555,`
  Lennard-Jones parameter 'epsilon'

`naatpt(1,1) = 1,`
  number of 'center' atoms in the 'lj' particle

`txaat(1,1) = 'lj',`
  label of 'center' atom in the 'lj' particle

`/`
  end of the namelist

Lennard-Jones particles with an embedded dipole

`&nmlParticle`
  beginning of the namelist

`txelec= 'dip',`
  enable atoms to possess a static dipole

`npt = 1,`
  number of particle types

`txpt = 'LJ',`
  particle's label

`nppt = 100,`
  number of particles of this type

`natpt = 1,`
  number of atom types which this particle is composed of

`txat = 'LJ',`
  atom's label

`massat= 18.0,`
  molar mass of the atom

`radat = 0.5,`
  hard-sphere radius of the atom

`zat = 0.0,`
  atom's valency

`sigat = 2.88630,`
  Lennard-Jones parameter 'sigma'

`epsat = 1.97023,`
  Lennard-Jones parameter 'epsilon'

`naatpt(1,1) = 1,`

number of 'LJ' atoms in the 'LJ' particle
```
txaat(1,1) = 'LJ',
```
label of 'LJ' atom in the 'LJ' particle
```
dipain(1,1,1) = 0.0000,
```
x, y, and z components of the static dipole of the first atom type of the first particle type
```
0.0000, 0.10584,
```
end of the namelist
```
/
```

Note: Here, `dipain(1,1,1)` is a simplified version of `dipain(1:3,1,1)`.

Nemo water (3-site water model with dipoles and polarizabilities)

```
&nmlParticle
```
beginning of the namelist
```
txelec= 'pol',
```
atoms possess charges, static and induced dipoles
```
npt = 1,
```
number of particle types
```
txpt = 'water',
```
particle's label
```
nppt = 2,
```
number of particles of this type
```
natpt = 2,
```
number of atom types which 'water' particles are composed of
```
txat = 'o ','h ',
```
atom's labels; 1st atom type is oxygen
```
massat= 16.0, 1.0,
```
molar mass of the atom
```
radat = 0.0, 0.0,
```
hard-sphere radius of both atom types
```
zat = -0.80100, 0.400500,
```
valency of oxygen and hydrogen atom
```
naatpt(1,1) = 1, 2,
```
number of oxygen and hydrogen atom in water
```
txaat(1,1) = 'o ','h ','h ',
```
molecule atom's label
```
rain(1,1,1) = 0.0, 0.0, -0.0656,
```
x, y, and z coordinates of oxygen
```
rain(1,2,1) = 0.7572, 0.0, 0.5205,
```
x, y, and z coordinates of 1st hydrogen
```
rain(1,3,1) = -0.7572, 0.0, 0.5205,
```
x, y, and z coordinates of 2nd hydrogen
```
dipain(1,1,1) = 0.0000, 0.0000, -0.1299,
```
x, y, and z components of the static dipole of the oxygen atom
```
dipain(1,2,1) = 0.0784, 0.0000, 0.0422,
```
x, y, and z components of the static dipole of the 1st hydrogen atom
```
dipain(1,3,1) = -0.0784, 0.0000, 0.0422,
```
x, y, and z components of the static dipole of the 2nd hydrogen atom
```
polain(1,1,1) = 0.6715, 0.6133, 0.7002, 0.0000, 0.0000, 0.0000,
```
xx, yy, zz, xy, xz, and zz components of the symmetric polarizability of the oxygen atom

```
polain(1,2,1) = 0.2199, 0.0756, 0.1441, 0.0000, 0.1005, 0.0000,
```
xx, yy, zz, xy, xz, and zz components of the symmetric polarizability of the 1st hydrogen atom
```
polain(1,3,1) = 0.2199, 0.0756, 0.1441, 0.0000, -0.1005, 0.0000,
```
xx, yy, zz, xy, xz, and zz components of the symmetric polarizability of the 2nd hydrogen atom
```
/
```
end of the namelist

Simple 1:1 elecrolyte

```
&nmlParticle
```
beginning of the namelist
```
npt = 2,
```
number of particle types
```
txpt = 'ion1', 'ion2',
```
labels of both particle types
```
nppt = 108, 108,
```
number of both particle types
```
natpt = 1, 1,
```
number of atom types which particles are composed of
```
txat = 'site1','site2',
```
labels of atom types
```
massat= 23.0, 35.4,
```
molar mass of the atom (in grams per mole)
```
radat = 2.0, 2.0,
```
hard-sphere radius of both atom types
```
zat = 1.0, -1.0,
```
valency of both atom types
```
naatpt(1,1) = 1,
```
number of 'site1' atoms in 'ion1' particle
```
txaat(1,1) = 'site1',
```
label of 'site1' atom in 'ion1' particle
```
naatpt(1,2) = 1,
```
number of 'site2' atoms in 'ion2' particle
```
txaat(1,2) = 'site2',
```
label of 'site2' atom in 'ion2' particle
```
/
```
end of the namelist

Macroion + counterions

```
&nmlParticle
```
beginning of the namelist
```
npt = 2,
```
number of particle types
```
txpt = 'macroion','ion',
```
labels of both particle types
```
nppt = 1, 60,
```
number of particles of each particle types

`natpt = 1, 1,`
: *number of atom types which particles are composed of*

`txat = 'mic', 'ion',`
: *labels of atom types*

`massat= 460.0, 23.0,`
: *molar mass of the atom (in grams per mole)*

`zat = -60, 1,`
: *valency of both atom types*

`radat = 20.0, 2.0,`
: *hard-sphere radius of both atom types*

`naatpt(1,1) = 1,`
: *number of 'mic' atoms in 'macroion' particle*

`txaat(1,1) = 'macroion',`
: *label of 'macroion' atom in 'macroion' particle*

`naatpt(1,2) = 1,`
: *number of 'ion' atoms in 'ion' particle*

`txaat(1,2) = 'ion',`
: *label of 'ion' atom in 'ion' particle*

`/`
: *end of the namelist*

Polyions + counterions

`&nmlParticle`
: *beginning of the namelist*

`nct = 1,`
: *number of chain types*

`txct ='100-mer',`
: *label of the chain type*

`ncct = 10,`
: *number of chains in the system*

`npptct(1,1) = 100, 0`
: *number of particles in the '100-mer' is equal to 100 of type 'pe' and zero of type ion'*

`npt = 2,`
: *number of particle types*

`txpt = 'pe', 'ion',`
: *labels of both particle types*

`nppt = 1000,1000,`
: *number of particles of each particle types*

`natpt = 1, 1`
: *number of atom types which particles are composed of*

`txat = 'bead', 'ion',`
: *labels of atom types*

`massat= 10.0,10,`
: *molar mass of the atom (in grams per mole)*

`radat = 2.0, 2.0,`
: *hard-sphere radius of both atom types*

`zat = 1.0, -1.0,`
: *valency of both atom types*

`naatpt(1,1) = 1`

number of 'bead' atoms in 'pe' particle
`txaat(1,1) = 'bead',`
label of 'bead' atom in 'pe' particle
`naatpt(1,2) = 1,`
number of 'ion' atoms in 'ion' particle
`txaat(1,2) = 'ion',`
label of 'ion' atom in 'ion' particle
`/`
end of the namelist

Note: Here `npptct(1,1)` is a simplified version of `npptct(1:2,1)`.

Diamond-like polyelectrolyte gel + counterions

`&nmlParticle`
beginning of the namelist
`lclink=.true.,`
flag to enable crosslinking
`maxnbondcl= 4, 2, 0,`
maximum number of crosslinks for types of particles
`nct = 1,`
number of chain types
`txct ='strand'`
label of chain type
`ncct = 16,`
number of chains of type 'strand'
`npptct(2,1) = 10,`
number of particles of type 'strand' in the chain type 'strand'
`npt = 3,`
number of particle types
`txpt='node','strand','countion',`
labels of all particle types
`nppt = 8, 160, 168,`
number of particles of each particle types
`natpt = 1, 1, 1`
number of atom types which particles are composed of
`txat='bead1','bead2','countion'`
labels of atom types
`radat = 2.0, 2.0, 2.0,`
hard-sphere radius of both atom types
`zat = 1.0, 1.0, -1.0,`
valency of all three atom types
`naatpt(1,1) = 1,`
number of 'bead' atoms in 'node' particle
`txaat(1,1) = 'bead',`
label of 'bead' atom in 'node' particle
`naatpt(1,2) = 1,`
number of 'bead2' atoms in 'strand' particle
`txaat(1,2) = 'bead2',`
label of 'bead2' atom in 'strand' particle

```
naatpt(1,3) = 1,
        number of 'countion' atoms in 'countion' particle
txaat(1,3) = 'countion',
        label of 'countion' atom in 'countion' particle
/
        end of the namelist
```

Note: Here `npptct(1,1)` is a simplified version of `npptct(1:2,1)`.

Bottle-brushes polymers with uneven side-chain distribution

```
&nmlParticle
        beginning of the namelist
lclink =.true.,
        flag to enable crosslinking
maxnbondcl = 1, 1,
        maximum number of crosslinks for types of particles
ngen = 1,
        number of generations of hierarchichal structure
ictgen(0) = 1,
        chain type of 0th generation (main chain)
ictgen(1) = 2,
        chain type of 1st generation (side chain)
nbranch = 5,
        number of branches on the main chain
ibranchpbeg = 1,
        main chain particle number of the first branch point
ibranchpinc = 1,
        segment increment for branch points on the main chain
nct = 2,
        number of chain types
txct ='10-mer1','5-mer2',
        label of each chain type
ncct = 2, 10,
        number of chains of each type
npptct(1,1) = 10, 0,
        number of particles of each type in the 1st chain
npptct(1,2) = 0, 5,
        number of particles of each type in the 2nd chain
npt = 2,
        number of particle types
txpt = 'bead1', 'bead2',
        labels of both particle types
nppt = 20, 50,
        number of particles of each particle types
natpt = 1, 1,
        number of atom types which particles are composed of
txat = 'site1', 'site2',
        labels of atom types
massat= 10.0, 10.0,
```

molar mass of the atom (in grams per mole)

`radat = 2.0, 2.0,`

hard-sphere radius of both atom types

`naatpt(1,1) = 1,`

number of 'site1' atoms in 'bead1' particle

`txaat(1,1) = 'site1',`

label of 'site1' atoms in 'bead1' particle

`naatpt(1,2) = 1,`

number of 'site2' atoms in 'bead2' particle

`txaat(1,2) = 'site2',`

label of 'site2' atoms in 'bead2' particle

`/`

end of the namelist

Note: Here `npptct(1,1)` is a simplified version of `npptct(1:2,1)` etc.

```
particle id
  1       5       10  11      15        20
  x x x x x x x x x x    x x x x x x x x x x
 21  0  0  26  0  41     46  0  0  51  0  66
   0 0      0            0 0    0
   0 0 ...  0            0 0 .. 0
   0 0      0            0 0    0
   0 0      0            0 0    0
```

Third generation dendrimers

`&nmlParticle`
beginning of the namelist

`lclink =.true.,`
flag to enable crosslinking

`maxnbondcl = 2, 2, 2, 1,`
maximum number of crosslinks for types of particles

`ngen = 3`
number of generations of hierarchichal structure

`ictgen(0) = 1,`
chain type of 0th generation

`ictgen(1) = 2, 3, 4,`
chain type of 1st, 2nd, and 3rd generation

`nbranch = 2, 2, 2,`
number of branches of 1st, 2nd and 3rd generation

`ibranchpbeg = 1, 4, 4,`
segment number of the first branch point of 1st, 2nd, and 3rd generation

`ibranchpinc = 0, 0, 0,`
segment increment for branch points of 1st, 2nd, and 3rd generation

`nct = 4,`
number of chain types

`txct ='1-mer1', '4-mer2', '4-mer3', '4-mer4'`
label of each chain type

`ncct = 1, 2, 4, 8,`
number of chains of each type

`npptct(1,1) = 1, 0, 0, 0,`
number of particles of type 'bead1' in the chain type '1-mer1'. Chain '1-mer1' is composed only of 'bead1' particles.

`npptct(1,2) = 0, 4, 0, 0,`
number of particles of type 'bead2' in the chain type '4-mer2'. Chain '4-mer2' is composed only of 'bead2' particles.

`npptct(1,3) = 0, 0, 4, 0,`
number of particles of type 'bead3' in the chain type '4-mer3'. Chain '4-mer3' is composed only of 'bead3' particles.

`npptct(1,4) = 0, 0, 0, 4,`
number of particles of type 'bead4' in the chain type '4-mer4'. Chain '4-mer4' is composed only of 'bead4' particles.

`npt = 4,`
number of particle types

`txpt = 'bead1', 'bead2', 'bead3', 'bead4',`
    labels of all particle types

`nppt = 1, 8, 16, 32,`
    number of particles of each particle types

`natpt = 1, 1, 1,`
    number of atom types which particles are composed of

`txat = 'site1', 'site2', 'site3', 'site4'`
    labels of atom types

`massat= 10., 10., 10., 10.,`
    molar mass of the atom (in grams per mole)

`radat = 2.0, 2.0, 2.0, 2.0,`
    hard-sphere radius of both atom types

`naatpt(1,1) = 1,`
    number of 'site1' atoms in 'bead1' particle

`txaat(1,1) = 'site1',`
    label of 'site1' atoms in 'site1' particle

`naatpt(1,2) = 1,`
    number of 'site2' atoms in 'bead2' particle

`txaat(1,2) = 'site2',`
    label of 'site2' atoms in 'site2' particle

`naatpt(1,3) = 1,`
    number of 'site3' atoms in 'bead3' particle

`txaat(1,3) = 'site3',`
    label of 'site3' atoms in 'site3' particle

`naatpt(1,4) = 1,`
    number of 'site4' atoms in 'bead4' particle

`txaat(1,4) = 'site4',`
    label of 'site4' atoms in 'site4' particle

`itestpart = 10,`
    test output of chain pointers

`/`
    end of the namelist

Note: Here `npptct(1,1)` is a simplified version of `npptct(1:2,1)` etc.

```
particle id
     30    33   49  46
   13+ # # # #   # # # # +21
  26# +             + #42
   #   +          +  #
 29# #  10+5  2 1 6  9+18  # #45
     0 0 0 0 x 0 0 0 0
 37# #  14+        +22 #  #53
    #  +         + #
   34# +          + #50
    17+ ####     #### +25
     38  41     57  54
```

11.3 Suggestion or Problems?

If you have suggestions or problems with Molsim, open an issue `online`. Please add the following information to help us help you:

- MOLSIM version number (check by running `molsim --version`)
- Host computer (name and configuration)
- Suggestion or problem in as much detail as possible
- Any Input/Cnf files needed to reproduce the error