AnalysisTools user manual

RNDr. Karel Šindelka, Ph.D.
k.sindelka@gmail.com

Version 4.0 (Released ~~August 18, 2020,~~ at some point, it will be...)

Contents

1. Introduction

AnalysisTools is a set of utilities to (mainly) analyse trajectories of particle-based simulations. The utilities (including one from the DLMESO simulation package) can be roughly divided into four types:

- utilities to calculate system-wide properties; e.g., pair correlation functions or particle densities along a simulation box axis
- utilities to calculate per-molecule or per-aggregate properties (where aggregate stands for any supramolecular structure); e.g., shape descriptors for individual molecules or whole micelles
- utilities to manipulate a configuration; e.g., create initial configuration for a simulation from scratch or by adding molecules to an existing one
- helper utilities to analyse text files; e.g., calculate averages and standard deviations of a data series

The `Examples` directory contains examples showcasing capabilities of individual utilities as well as possible workflows chaining several utilities.

1.1 Installation

AnalysisTools requires `C` and `Fortran` compilers and `cmake`. The utilities should be compiled in a separate directory, typically `<root>/build` (where `<root>` is the AnalysisTools root directory). The following generates a `Makefile` within `<root>/build`:

```
mkdir <root>/build; cd <root>/build; cmake ../
```

To then compile the utilities, simply run `make` (to compile all utilities) or `make <utility name>` (to compile a single utility) in `<root>/build`. The binaries are located in the `<root>/build/bin` directory.

2. Format of input/output files

This section describes several file types used by many of the AnalysisTools utilities. Output files for the utilities themselves are described in their respective sections in chapter 3.

Input files are divided into two categories: structure and coordinate files. The structure files contain information about the system composition, that is, number of beads and molecules, bead properties (name, mass, charge, etc.), molecule properties (name, number and types of beads, connectivity, etc.). The coordinate files contain individual timesteps with beads' coordinates (and, possibly, velocities, forces, etc.). While most files can be used as both a structure and a coordinate file, using a separate structure file may be essential. For example, an `xyz` file (see below) may be used as both, but because it only provides bead names, using a separate structure file would provide additional information (bead and/or molecule properties).

For working with aggregates, AnalysisTools' `agg` file format (see below) describes their compositions in terms of molecule numbers taken from the structure file used to generate it. Any of the `Aggregates` (section 3.2) utilities can create the `agg` file.

2.1 Structure files

As a structure file, any of the following formats can be used; browse the `Examples` directory for example files. Most utilities can load a specific structure file using the `-i <file>` option; if the option is not used, it is usually assumed the structure file is the same as the coordinate file.

Importantly, many of the AnalysisTools utilities work (either fundamentally or optionally) with types of beads and/or molecules. In these cases, names are used specify the bead/molecule types. While many input files support naming beads and molecules, not all do (in some formats, these are optional); in such cases, AnalysisTools names the bead types as `b` (and `b1`, `b2`, etc., if more than one bead type is present) and molecule types as `m` (and `m1`, `m2`, etc., in case of multiple molecule types). If unsure, using the Info utility (section 3.8), one can see the names of beads and molecules for the specific structure file.

`Info` utility can also convert these formats between each other via the `-o <file>` option.

2.1.1 VTF format

See here for the complete description of `vtf/vsf` files (a `vtf` file contains a structure part followed by a coordinate part, while a `vsf` file contains only the structure part). In this format, every bead is defined on its own line (except for the `atom default` bead type) and all bonds all listed.

Not all the keywords listed on the web page for the 'Atom lines' are recognized; keyword `n[ame]` is mandatory and optional keywords are `m[ass]`, `charge|q`, `r[adius]`, `resid`, and `res[name]`. For the `<aid-specifier>`, only a single bead index number or the `default` keyword can be used. Other keywords are ignored.

AnalysisTools groups beads (and molecules) into bead (and molecule) types. Generally, only name defines types of beads, i.e., should two beads share the name, they will be of the same type; their mass, charge, and radius will each equal to that for the bead type's topmost atom line containing the corresponding keyword. If the keyword is missing from all atom lines, that characteristic is undefined. In the case of the `Info` utility (see section 3.8 for details), the `--detailed` command line option triggers the detailed recognition, where beads that share the name but differ in mass, charge, and/or radius will be of different type (with a number appended to the original name).

Different molecule types, on the other hand, are always distinguished based on all their characteristics, i.e., name, connectivity, and order of bead types (order of its `vsf` indices). Molecule types may also be affected by the `--detailed` option because of the bead type definition.

For bond lines, only one bond per line is permitted (i.e., `bond <index1>:<index2>` format).

Periodic boundary condition can be included as `pbc <x> <y> <z>` for a cuboid box or `pbc <a> <b> <c> <alpha> <beta> <gamma>` for a general rhomboid box.

2.1.2 FIELD format

A format used by the DL_MESO_DPD simulation package concisely defines species of beads and molecules, providing number of beads/molecules of given species as well as their properties.

The `species` section is the same as in the `DL_MESO` `FIELD` file, but the `molecules` section differs slightly. Besides the `beads` part, every molecule can have `bonds`, `angles`, and `dihedrals`, but even though up to three parameters for potential parameters are read, AnalysisTools does not recognize the potential keywords (`harm`, `cos`, etc.). Moreover, these potential parameters are only used for generating new `LAMMPS` or `DL_MESO` data files via, e.g., the `Info` utility (section 3.8). The `molecules` section can also contain one extra part: `impropers` for improper angles which has the same format as the `dihedrals` section (this was added because the LAMMPS molecular dynamics simulator and other software distinguish between the two types).

Note that the `FIELD` file does not have to contain the `Interactions` section; this part is ignored.

However, using this file in conjunction with coordinate files is not recommended. Because `FIELD` contains only numbers of beads of given types, it is not possible to ensure that bead indices in the coordinate file will line up with the correct bead types in the `FIELD` file. AnalysisTools assigns the bead indices on the 'first read, first labelled' basis, that is, the unbonded beads are first (in the order of the lines in the `species` section) followed by beads from the molecules. Should one want to check the bead index assignments, the `Info` utility can be used to generate, e.g., a `vsf` file from the `FIELD` file.

Nevertheless, the file can be used to supply extra information for bead and molecule types for the `Info` utility's `-i` option as these do not depend on individual beads but rather on bead and molecule type names (see section 3.8 for details). Moreover, `FIELD` is used as an input file to add extra species into an existing system or to create a new system from scratch via, e.g., AddToSystem (section 3.1) or ref GenSystem, GenLayers, or some such.

To be recognized by AnalysisTools utilities, the file must either be called `FIELD` or have the `.FIELD` extension.

2.1.3 XYZ format

This is the simplest and probably best known format, see, e.g., here for the description. An `xyz` file is essentially a file with timesteps (bead coordinates), and the only structure information are bead names. Note that the second line of each timestep (a comment line) is ignored.

When used as a structure file, the first timestep is used to define the system composition. Therefore while the timesteps can each have a different number of beads, their amount cannot exceed the number in the first step.

2.1.4 LAMMPS data format

This rather complicated format comes from the LAMMPS molecular dynamics simulator and lists the numbers of beads, bonds, angles, etc., as well as individual beads, bonds, angles, etc., and information about potentials for bonds, angles, etc. See here for description of the format and the `Examples` directory for example files.

AnalysisTools reads the header information as well as `Masses`, `Atoms`, `Velocities`, `Bonds`, `Angles`, `Dihedrals`, `Impropers`. It also reads `Bond Coeffs`, `Angle Coeffs`, `Dihedral Coeffs`, and `Improper Coeffs` sections.

The `Atoms` section may have `atom_style full`, `bond`, `angle`, `atomic`, `molecular`, or `charge` format (specified as a trailing comment on the `Atoms` line as required by the `LAMMPS` software).

For the `Bond Coeffs` and `Angle Coeffs` sections, harmonic potential is assumed, and the first value of each bond/angle type is multiplied by 2 because LAMMPS uses harmonic spring strength of $k/2$ (as opposed to, e.g., DL_MESO which uses k). For the `Dihedral Coeffs` and `Improper Coeffs` sections, AnalysisTools reads at most three numbers without assuming any specific potential. The `Coeffs` sections are used only for generating new structure files via, e.g., the Info (section 3.8) utility; note that to run `LAMMPS` simulations, these section may have to be manually corrected to conform to the specific potential type.

To be recognized by AnalysisTools utilities, the file must have an extension `.data`.

2.1.5 LAMMPS lammpstrj format

This file format comes from the `LAMMPS` dump style custom command.

Of the possible `LAMMPS` attributes, AnalysisTools recognizes only `id` (bead index – mandatory), `element` (bead name), `x y z` (bead coordinates), `vx vy vz` (bead

velocities), and `fx fy fz` (bead forces). Other attributes are ignored. Note that if `element` is missing, all beads are considered of the same type called `b`.

When used as a structure file, the first timestep is used to define the system composition. Therefore while the timesteps can each have a different number of beads, their amount cannot exceed the number in the first step.

To be recognized by AnalysisTools utilities, the file must have an extension `.lammpstrj`.

2.2 Coordinate files

As a coordinate file, any of the following formats (described above) can be used:

- `vtf/vcf` format (`vtf` file contains both structure and coordinate sections while `vcf` file contains only the coordinate section); both indexed and ordered timesteps can be used, but the AnalysisTools utilities always output files with ordered timesteps
- `xyz` format where timesteps can have a different number of beads
- LAMMPS `data` format which by definition can only contain one timestep
- LAMMPS `lammpstrj` format where timesteps can have a different number of beads

2.3 Aggregate file

An `agg` is generated using any of the `Aggregates` utilities. The file contains information about the number of aggregates in each timestep and which molecules belong to which aggregate. It serves as an additional input file for utilities that calculate aggregate properties; `agg` file is, therefore, linked to the structure and coordinate file(s) used to generate it.

The `agg` file is a simple text file. The first two lines are just comments (the second one should contain the command used to generate the file as parts of the command may be used by subsequent aggregate analysis). Starting at the third line, the data for individual timesteps are shown. It follows these rules:

- each timestep starts with `Step: <int>`
- the second line is the number of aggregates in the given timestep
- for each aggregate, there is a single line in the format `<size> : <id1> <id2> ...`; the `<id#>` are molecular indices from the input structure file
- no blank or comment lines are allowed
- not all molecules present in the coordinate/structure file(s) used to generate this file must be present in every timestep
- the keyword `Last` instead of `Step:` signalizes the end of data

Note that the term aggregate also refers to free molecules (i.e., unimers or fully dissolved chains).

If `vtf` format is used for the coordinates, the indices from `agg` file can be used in vmd to visualize, e.g., only a specific aggregate by using `resid <id1> ... <id#>` in the `Selected Atoms` box inside the vmd. A rough script to visualize aggregates in different colours is in the `Examples/scripts/Visualize` directory (`VisAgg*` files),

accompanied by the resultant picture (the bash script uses the provided `vtf` and `agg` files to generate a `tcl` script and run it via `vmd`).

An example of an `agg` file can be found in the `Examples/DistrAgg` directory.

3. Utilities

All utilities have command line options affecting their behaviour. The following options are common for many utilities, so they are described here rather than at the individual utilities where only list of the possible options is provided.

General options	
`--verbose`	print information about all bead and molecule types to the screen
`--silent`	run silently, i.e., without any output at all (overrides `--verbose` option)
`--help`	print short description of the utility and usage
`--version`	print version of the utilities and exit
Options for structure and coordinate input files	
`-i <name>`	use specified structure file, see section 2.1 for possible formats
`-st <n>`	start calculations at `n`-th timestep
`-e <n>`	end calculations at `n`-th timestep
`-sk <n>`	skip every `n` timesteps (i.e., use every `(n+1)`-th timestep)

The `-st`, `-e`, and `-sk` options can be combined, so that, for example, specifying `-st 2 -e 10 -sk 2` would use timesteps 2, 5, and 8.

Note that should two (or more) identical options be specified on the command line, only the first one is used.

3.1 AddToSystem

This utility takes an existing system (or an empty system, i.e., creates new system from scratch) and adds new beads (read from a `FIELD` file) into it, placing them either completely randomly or according to supplied constraints.

There are two types of constraints which can be combined: place new beads (i) specified distance from other beads or (ii) in a specified interval in x-, y-, and/or z-axis directions. In (i), options `-ld` and/or `-hd` specify the distance; if present, these must be accompanied by `-bt` or `--bonded` option. The new beads are then placed at least `-ld <float>` and at most `-hd <float>` distance from beads specified by the `-bt` option or from any bonded bead (`--bonded` option).

In (ii), options `-cx`, `-cy`, and `-cz` basically change the box size for the added beads. By default, this constraint is specified in units relative to the box dimensions (i.e., allowed values go from 0 to 1); use the `--real` option for absolute units. For example, `-cx 0.5 1` would generate x coordinates between 50% and 100% of the box's x sidelength.

All these options can be combined, but note that `AddToSystem` does not perform any sanity checks; that is, if any combination of the provided options is impossible to achieve, the utility may run forever. See figure 3.1 for schematic examples.

For added molecules, either the molecule's geometric centre (default behaviour) or its first bead (`--head` option) obey these constraints. The coordinates of the remaining beads in the molecule are governed by the coordinates in the input `FIELD` file. Therefore, not all the molecular beads necessarily obey the constraining rules.

By default, molecules are added with a random orientation; when `--no-rotate` switch is used, the molecules are not rotated, but when `-a` option is used, all the molecules are rotated by the three angles specified in degrees.

The α, β, and γ angles in `-a <alpha> <beta> <gamma>` represent yaw, pitch, and roll, respectively. To put it in other words, it is an intrinsic rotation with Tait-Bryan angles α, β, and γ about axes x, y, and z, respectively. The rotation matrix,

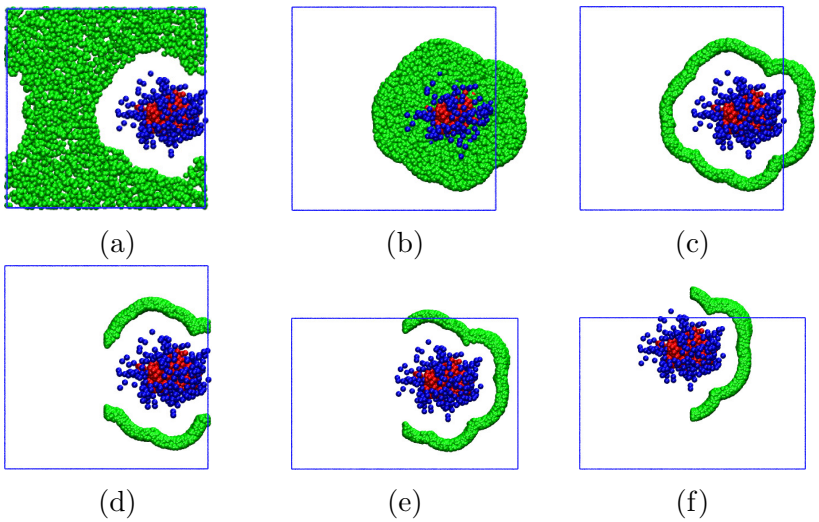


Figure 3.1: Examples of distance checks (cross-section snapshots): (a) `-ld 3 -bt A B` specifies that any added bead (green) is at least at a distance of three from any `A` or `B` bead (red and blue, respectively); (b) `-hd 4 -bt A B` specifies added beads are at most four units away from any `A` or `B` bead; (c) combines (a) and (b) into `-ld 3 -hd 4 -bt A B`, that is, new beads are added at a distance between three and four from any `A` or `B` bead; (d) combines (c) with axis constraint into `-ld 3 -hd 4 -cx 0.5 1 -bt A B` which constrains the x-coordinate of new beads between 50% and 100% of the box's x sidelength; (e) adds `-b 30 20 25` to the options in (d), changing the box size; finally, (f) also moves the original system, adding `-off -0.2 0.2 0`. See `Examples/AddToSystem/Manual` folder for source of these examples.

R, is, therefore, defined as

$$\begin{aligned} R &= R_z(\alpha)R_y(\beta)R_x(\gamma) \\ &= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\ &= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \end{aligned} \tag{3.1}$$

By default, the new beads are exchanged for beads in the original system; what bead type to exchange is either the most numerous one (default behaviour) or provided via the `-xb` option. Note that the utility doesn't check whether exchanged beads are in a molecule, so a bonded bead may be exchanged, leaving only part of the molecule in the new system. If beads are to be added into the system rather than switched, use the `--add` option.

The size of the new simulation box can be changed using the `-b` option (the new size must be in absolute units, i.e., it is unaffected by the `--real` option). Any constraints for placing beads are applied to this box rather than the one in the original system. By default, the new system is placed so the centres of the original box and the new one coincide. Note that periodic boundary conditions are not applied to the new system, so the beads can be outside the simulation box. If `-off` option is used, the beads from the input system are instead moved by the offset specified by the fraction of the output simulation box (or in real units, if `--real` is used) before the new beads are added.

To create a new system instead of adding to an existing one, use `-` instead of an `<input>` file. An extra file can be generated via the `-o` option; this can be useful to, e.g., generate both `vtf` file for visualization and `data` file for use by LAMMPS.

Examples of using the utility (such as how to generate a bilayer or wire-like aggregate) are provided in the `Examples/AddToSystem` folder (including those in figure 3.1).

Usage: `AddToSystem <input.vcf> <out.vsf> <out.vcf> [options]`

Mandatory arguments	
`<input>/-`	input coordinate file (or create a system from scratch)
`<in.field>`	input `FIELD` structure file for the new system
`<output>`	output coordinate file for the new system

options	
`-o <file>`	extra structure file
`-ld <float>`	lowest distance from beads specified by `-bt` option
`-hd <float>`	highest distance from beads specified by `-bt` option
`-bt <name(s)>`	bead types to use in conjunction with `-ld` and/or `-hd` options

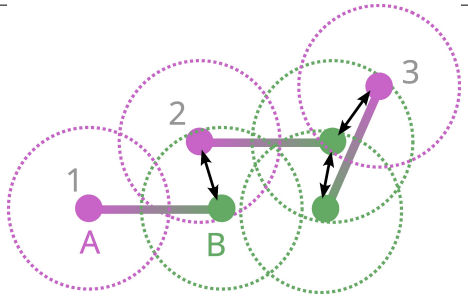
`--bonded`	use bonded beads for the distance condition (overwrites `-bt` option)
`-xb <name>`	specify bead type to exchange (default: most numerous bead type)
`--add`	replace original beads instead of increasing the total number of beads
`--real`	use real coordinates for -cx/-cy/-cz/-off options instead of fractions of input box size
`--no-rotate`	do not randomly rotate added molecules
`-a 3×<angle>`	rotate all added by molecules along x-, y-, and z-axis
`--head`	use molecule's first bead for distance check (default: molecule's geometric centre)
`-cx 2×<float>`	constrain x coordinate to interval $\langle num, num2)$ (units relative to box size)
`-cy 2×<float>`	constrain y coordinate to interval $\langle num, num2)$ (units relative to box size)
`-cz 2×<float>`	constrain z coordinate to interval $\langle num, num2)$ (units relative to box size)
`--real`	use 'real' units instead of relative ones for `-cx`/`-cy`/`-cz` options
`-b <x> <y> <z>`	side lengths of the new simulation box
`-s <int>`	seed for random number generator
`-i`, `-st`, `--verbose`, `--silent`, `--help`, `--version`	

3.2 Aggregates and Aggregates-NotSameBeads

These utilities determine which molecules belong to which aggregates (note that 'aggregate' is used for any supramolecular structure) according to a simple criterion: two molecules belong to the same aggregate if they share at least a specified number of contact pairs. A contact pair is a pair of two beads belonging to different molecules which are closer than a specified distance. The information is written into an `.agg` text file described in Section 2.3.

Periodic boundary conditions can be removed from whole aggregates via the `-j` option; the aggregate's centre of mass is always inside the simulation box. This is useful for visualization (an aggregate will no longer be split by the walls of the simulation box, but it may stretch far outside the boundaries of the box) as well as for further analysis (the other utilities do not have to join the aggregates, possibly greatly speeding up subsequent analyses).

While the `Aggregates` utility uses all possible pairs of given bead types, `Aggregates-NotSameBeads` does not use same-type pairs. For example, if bead types `A` and `B` are given, `Aggregates` will use all three possible bead type pairs (i.e., `A-A`, `A-B`, and `B-B`), but `Aggregates-NotSameBeads` will use only `A-B` bead type pairs. Fig 3.2 illustrates the behaviour: using `Aggregates ... A B` would find one contact pair between molecules 1 and 2 and two contacts between molecules 2 and 3, whereas



`Aggregates ... A B`
⇒ 3 contact pairs
`Aggregates-NotSameBeads ... A B`
⇒ 2 contact pairs

Figure 3.2: Simple example of distance checks—three two-bead molecules, where dotted lines represent maximum pair distance (`-d` option) and black arrows show contact pairs. Note the same bead can be a part of multiple pairs.

`Aggregates-NotSameBeads ... A B` would ignore A-A and B-B pairs, finding no contact pair between molecules 1 and 2 and only one between molecules 2 and 3.

If `-w` option is used, the utilities also take the found aggregates, testing which are touching the specified wall(s) and saving those touching the wall(s) and those in bulk into two extra `.agg` files. The wall(s) are specified via the `<a>` argument, which defines the axis perpendicular to the wall (the argument must be `x`, `y`, or `z`), and numeric argument(s), which define the coordinate(s) of the wall(s) along the axis. For example, assuming a simulation box with size 10^3 and a slit configuration with two walls in the xy-plane 1 distance unit from the box edges, then `-w z 1 9` would be used. The distance for checking whether a bead touches the wall is the same as the one for determining bead contact pairs. Aggregate data are saved into two extra `.agg` files with names based on the main output `<out.agg>` – `-w` and `-no_w` is placed before the `.agg` extension.

Note that when `-st`, `-e`, or `-sk` options are used, the resulting `agg` file is no longer coupled to the original coordinate file (i.e., they cannot be used together for further analyses), but it is coupled to the optional output file with joined coordinates (`-j` option).

Usage: `Aggregates <input> <out.agg> <bead type(s)> [options]`

Mandatory arguments	
`<input>`	input coordinate file
`<out.agg>`	output `agg` file
Options	
`-d <num>`	maximum distance for contact (default: 1)
`-c <num>`	minimum number of contacts (default: 1)
`-j <file>`	save coordinates of joined aggregate to a coordinate file
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `-i`, `--verbose`, `--silent`, `--help`, `--version`	

3.3 Angle Molecules

This utility calculates angles (distribution and overall averages) between beads in each molecule of specified molecule type(s); the input structure file must contain angles (such as LAMMPS `data` file). It can also calculate angles between any three beads in the molecule type(s) via the `-n` option. See `Examples/AngleMolecules` for an example.

By default, the angles are calculated for every molecule type, but the `-m` option can be used to specify only some of them.

The utility groups angles according to bead types; e.g., in a linear 6-bead molecule (such as in the `Example/AngleMolecules` directory) with bead order A-A-A-B-A-A with all possible angles defined (i.e., 1-2-3, 2-3-4, 3-4-5, and 4-5-6 where the numbers specify bead indices in the molecule), there are three angle types: A-A-A, A-A-B, and A-B-A. However, using the `--all` option, all molecular angles are averaged separately as well; i.e., beside for the three angles based on bead types, the distributions and averages are calculated for all five angles in the molecule too.

Using `-n` option, extra angles are specified by three bead indices taken from the bead order in the molecule type(s). These indices go from 1 to N, where N is the number of beads in the molecule type (`Info` utility can be used to check the indices). Multiple angles may be specified, and the results are written to the specified file. Angles containing an index number that is higher than the number of beads in a molecule are ignored only for that molecule.

Usage: `AngleMolecules <input> <width> <output> [options]`

Mandatory arguments	
`<input>`	input coordinate file
`<width>`	width of each distribution bin
`<output>`	output file for distribution
Options	
`-m <name(s)>`	specify molecule type(s) to use
`--joined`	specify that `<input>` contains joined coordinates
`--all`	write all angles for the molecule types
`-n <file> <ints>`	multiple of three indices for extra angle
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `-i`, `--verbose`, `--silent`, `--help`, `--version`	

Format of output files:

1) `<output>` – distribution of angles (grouped by bead types and possibly for all molecular angles as well if `--all` option is used)
   - first line: AnalysisTools version
   - second line: command used to generate the file
   - following lines (number of specified molecules plus 1): column headers
     - first column is the centre of each bin in angles (governed by `<width>`); i.e., if `<width>` is 5°, then the centre of bin 0 to 5° is 2.5, centre of bin 5 to 10° is

7.5 and so on
  - the rest are for the calculated data: for each specified molecule there are angles grouped by bead types followed by all angles in that molecule (if `--all` option is used)
- next lines: the calculated data
- last several lines (number of specified molecules plus 2): minimum, maximum, and arithmetic mean for each calculated angle (last line) and headers (preceding lines)

2) `-n <file>` – angles specified by supplied molecular indices
- the file structure is the same as for the default output file

3.4 Average

This utility calculates simple average and standard error from specified column(s) of data in the input text file (all `#`-starting lines and blank lines are ignored), printing it to standard output. It can also calculate one of three types of averages based on a used option – an overall average with statistical error and an autocorrelation time estimate (`-tau` option), block averages (`-b` option), or moving averages (`-m` option). While the `-tau` option appends a single line to the output file, either of the `-b` or `-m` options creates a new output file with somewhat smoother data.

The first and last lines used for the average calculation can be controlled using the standard `-st` and `-e` options.

3.4.1 Estimate autocorrelation time via -tau option

The average value of an observable $\mathcal{O}$ is a simple arithmetic mean,

$$\langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}_i, \tag{3.2}$$

where N is the number of measurements and the subscript i denotes individual measurements. If the measurements are independent (i.e., uncorrelated), the statistical error, ϵ, is given by:

$$\epsilon^2 = \frac{\sigma^2_{\mathcal{O}_i}}{N}, \tag{3.3}$$

where $\sigma^2_{\mathcal{O}_i}$ is the variance of the individual measurements,

$$\sigma^2_{\mathcal{O}_i} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathcal{O}_i - \langle \mathcal{O} \rangle)^2. \tag{3.4}$$

For correlated data, the autocorrelation time, τ, representing the number of steps between two uncorrelated measurements must be determined. Every τ-th measurement is uncorrelated, so the equation (3.3) can then be used to estimate the error.

A commonly used method to estimate τ is the binning (or block) method. In this method, the correlated data are divided into N_B non-overlapping blocks of size k ($N = kN_\mathrm{B}$) with per-block averages, $\mathcal{O}_{\mathrm{B},n}$, defined as:

$$\mathcal{O}_{\mathrm{B},n} = \frac{1}{k} \sum_{\substack{i=1+\\(n-1)k}}^{kn} \mathcal{O}_i. \tag{3.5}$$

If $k \gg \tau$, the blocks are assumed to be uncorrelated and equation (3.3) can be used:

$$\epsilon^2 = \frac{\sigma^2_\mathrm{B}}{N_\mathrm{B}} = \frac{1}{N_\mathrm{B}(N_\mathrm{B}-1)} \sum_{n=1}^{N_\mathrm{B}} (\mathcal{O}_{\mathrm{B},n} - \overline{\mathcal{O}})^2. \tag{3.6}$$

An estimate of the autocorrelation time can be obtained using the following formula:

$$\tau_\mathcal{O} = \frac{k\sigma^2_\mathrm{B}}{2\sigma^2_{\mathcal{O}_i}}. \tag{3.7}$$

The number of blocks, N_B, is supplied as an argument of the `-tau` option and Average then appends a single line to the `<output>` file; the line starts with N_B and continues with three values ($\langle\mathcal{O}\rangle$, ϵ, and $\tau_\mathcal{O}$) per every data column specified in the Average command.

A way to quickly get a τ estimate is to use a wide range of N_B values and plot $\tau_\mathcal{O}$ from equation (3.7) as a function of N_B. Because the number of data points in one block should be significantly larger than the autocorrelation time (e.g., ten times larger), plotting $f(x) = N/(10x)$ will produce a monotonously decreasing curve that intersects the $\tau_\mathcal{O}$ vs. N_B curve. A value of $\tau_\mathcal{O}$ near the intersection (but to the left, where the decreasing curve is above $\tau_\mathcal{O}$ vs. N_B curve) can be considered a safe estimate of τ.

3.4.2 Block averages via -b option

Besides estimating the autocorrelation time, the per-block averages can be themselves plotted to get (probably) smoother dataset. Using the `-b` option, the number of data points per block to average, k, is supplied, and the utility prints the per-block averages from equation (3.5) to the output file.

This way of averaging could be useful for example with density data produced by DensityBox or related utilities; if the bin width supplied to the DensityBox was too small, it is (possibly much) faster to block-average the densities rather than rerun DensityBox. For this case, specify the first column (distance) along with any density columns from the density file.

3.4.3 Moving averages via -m option

More common way to smoothing noisy data is to use the moving (or rolling or running) average (or moving mean or rolling mean); this common method does have many names.

Similarly to the block-average, the first element of the moving average is a simple mean of k values. Unlike with block-average, however, the k values for the next element are obtained by ignoring only one value and taking the next k values; i.e., $k-1$ values from the previous element of the moving average are always reused as opposed to the block-average where the blocks of data are not overlapping.

The moving average elements, $\mathcal{O}_{\mathrm{M},n}$, are defined as

$$\mathcal{O}_{\mathrm{M},n} = \frac{1}{k} \sum_{n}^{n+k-1} \mathcal{O}_i. \tag{3.8}$$

Usage: `Average <input> <output> <column(s)> [options]`

Mandatory arguments	
`<input>`	input text file
`<output>`	output text file
`<column(s)>`	at least one column number from `<input>`

Options	
`-tau <int>`	τ estimation where `<int>` represents the number of blocks N_{B}
`-b <int>`	block-average printing mode where `<int>` represents the number of data points per one block, k (equation (3.5))
`-m <int>`	moving average mode where `<int>` represents the number of data points per one moving block, k (equation (3.8))

Other options (see the beginning of Chapter 3)
`-st`, `-e`, `--verbose`, `--silent`, `--help`, `--version`

3.5 BondLength

Completely rewrite...

This utility calculates a distribution of bond lengths in specified molecule type(s). It can also calculate a distribution of distances between any two beads in the molecules.

For each of the specified molecule type(s), `BondLength` calculates bond lengths between all different types of connected bead pairs. For example, assume two linear molecule types `Mol_1` and `Mol_2`, both composed of bead types `A` and `B`. `Mol_1` is connected like this: `A-A-B`; `Mol_2` like this: `A-B-B`. If both molecule types are used, `BondLength` calculates for each molecule type distribution of lengths for bonds `A-A`, `A-B`, and `B-B` (separate for each molecule even though the molecules share the same bead types).

To calculate the distribution of distances between specific (possibly unconnected) beads in a molecule, use `-d` option which takes as arguments pairs of bead indices (according to the order of beads in the molecule in `vsf` – similarly to the `-n` option in `AngleMolecules`, i.e., Section 3.3). More than one pair can be specified. These indices are the same for all `<mol name(s)>`. If an index higher than the number of

beads in the molecule is provided, the utility takes the last bead of the molecule (i.e., the highest index). For example, using `-d file.txt 1 2 1 9999` would write two distributions for each `<mol name(s)>` into `<file.txt>`: distribution of distances between the first and the second bead in each `<mol name(s)>` and between the first bead and the last one (or the 9999th bead).

In both cases, `BondLength` appends at the end of the file minimum and maximum bond lengths/distances.

Usage:

```
BondLength <input> <width> <output> <mol name(s)> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	width of each bin of the distribution
`<output>`	output file with distribution of bond lengths
`<mol name(s)>`	molecule name(s) to calculcate bond lengths for

Non-standard options	
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-d <out> <ints>`	write distribution of distances between specified beads in each specified molecule to `<out>`
`-w <double>`	warn if a bond length exceeds `<double>` (default: half a box length)

Format of output files:

1) `<output>` – distribution of bond length between all bead pairs
   - first line: command used to generate the file
   - second line: column headers
     - first is the centre of each bin (governed by `<width>`); i.e., if `<width>` is 0.1, then the centre of bin 0 to 0.1 is 0.05, centre of bin 0.1 to 0.2 is 0.15, etc.
     - the rest are for the calculated data: for each molecule type, there is a list of column numbers corresponding to all possible bead type pairs in the molecule; if no beads of the given types are connected, the data column contains `nan`
   - next lines: the calculated data
   - second to last line: column headers for minimal and maximal bond lengths
     - for each molecule, all the possible beads pairs are again listed
     - for each pair, the column number corresponds to the minimum, while the next column is always the maximum
   - last line: commented out (i.e., the line starts with `#`) minimal and maximal bond lengths
2) `-d <output> <ints>` – distribution of distances between specified beads
   - first line: command used to generate the file
   - second line: order of beads (given only by bead names) in each molecule
   - third line: column headers
     - first is again the centre of every bin

  - the rest are for the calculated data: for each molecule type, there is a list of column numbers corresponding to the given pairs of bead indices in the molecule (and to the `-d` option's arguments); the numbers also correspond to the order of beads in the previous line
- next lines: the calculated data
- second to last line: column headers for minimal and maximal bond lengths
  - for each molecule, all the possible beads pairs are again listed
  - for each pair, the column number corresponds to the minimum, while the next column is always the maximum
- last line: commented out (i.e., the line starts with `#`) minimal and maximal bond lengths

3.6 DensityBox

This utility calculates number density for all bead types along all three axis directions of the simulation box, generating one file per axis. The density is calculated from 0 to box length in the given direction, that is, the box is 'sliced' into blocks with width `<width>`, and numbers of different bead types are counted in each 'slice'. Or to put it in other words, a density profile for a bead type i along an axis α, $\rho(\alpha)$, is calculated as

$$\rho_i(\alpha) = \frac{\sum_j \delta(\alpha - \alpha_j^i)}{\Delta\alpha L_\beta L_\gamma}, \tag{3.9}$$

where $\delta(\alpha - \alpha_i^j)$ gives the number of i beads inside a slice of the simulation box of the thickness $\Delta\alpha$ (i.e., `<width>`) along the α-axis; L_β and L_γ represent box side lengths along the two remaining axes.

The utility does not distinguish between beads with the same name in different molecules, so if one bead type is in more than one molecule type, its density will be averaged over all molecule types it appears in. If one requires densities specific to certain molecules containing the same bead types, the `-x` option can be used to first run the utility without one molecule type and then rerun it, excluding the other molecule type. Thus, two output files (per axis) are generated, each missing densities from the specified molecule types.

Note that this utility assumes orthogonal box with constant side lengths; in case of triclinic box and/or varying box size, undefined behaviour may occur, i.e., the utility may crash or freeze, and any results will not be reliable.

Usage: `DensityBox <input> <width> <output> [options]`

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	width of each bin of the distribution
`<output>`	three output files with automatic `-x.rho`, `-y.rho`, and `-z.rho` endings
Options	

`-x <mol name(s)>`	exclude specified molecule type(s) (i.e., do not calculate density for beads in molecules `<mol name(s)>`)
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `-i`, `--help`, `--verbose`, `--silent`, `--version`	

Format of output files:
1) `<output>` – bead densities; one file per x-, y-, and z-axis
- first line: AnalysisTools version
- second line: command used to generate the file
- third line: column headers
  - first is the centre of each bin (governed by `<width>`); i.e., if `<width>` is 0.1, then the centre of bin 0 to 0.1 is 0.05, centre of bin 0.1 to 0.2 is 0.15, etc.
  - the rest are for the calculated data: each column corresponds to the number density of the specified bead type
- the rest of the file are data lines

3.7 DistrAgg

This utility calculates average aggregate mass and aggregation number for each timestep (i.e., time evolution) and the averages over all timesteps from a supplied `agg` file (see Section 2.3 for its format). It calculates number, weight, and z averages. It also calculates distribution functions of aggregation sizes.

Generally, for a quantity $\mathcal{O}$, the number, weight, and z averages, $\langle\mathcal{O}\rangle_\mathrm{n}$, $\langle\mathcal{O}\rangle_\mathrm{w}$, and $\langle\mathcal{O}\rangle_\mathrm{z}$, respectively, are defined as

$$\langle\mathcal{O}\rangle_\mathrm{n} = \frac{\sum_i N_i\mathcal{O}_i}{N}, \quad \langle\mathcal{O}\rangle_\mathrm{w} = \frac{\sum_i N_i m_i\mathcal{O}_i}{\sum_i N_i m_i}, \text{ and } \quad \langle\mathcal{O}\rangle_\mathrm{z} = \frac{\sum_i N_i m_i^2\mathcal{O}_i}{\sum_i N_i m_i^2}, \tag{3.10}$$

where N is the total number of measurements, i.e., the total number of aggregates for per-aggregate averages (or molecules for per-molecule averages); N_i is the number of measurements with the value $\mathcal{O}_i$, and m_i is mass of an aggregate i (or a molecule i).

Number, weight, and z distribution functions of aggregate sizes, $F_\mathrm{n}(A_\mathrm{S})$, $F_\mathrm{w}(A_\mathrm{S})$, and $F_\mathrm{z}(A_\mathrm{S})$, respectively, are defined as

$$\begin{aligned} F_\mathrm{n}(A_\mathrm{S}) &= \frac{N_{A_\mathrm{S}}}{\sum_{A_\mathrm{S}} N_i} = \frac{N_{A_\mathrm{S}}}{N}, \\ F_\mathrm{w}(A_\mathrm{S}) &= \frac{N_{A_\mathrm{S}} m_{A_\mathrm{S}}}{\sum_{A_\mathrm{S}} N_i m_i} = \frac{N_{A_\mathrm{S}} m_{A_\mathrm{S}}}{\sum_{i=1}^{N} m_i} = \frac{N_{A_\mathrm{S}} m_{A_\mathrm{S}}}{M}, \text{ and} \\ F_\mathrm{z}(A_\mathrm{S}) &= \frac{N_{A_\mathrm{S}} m_{A_\mathrm{S}}^2}{\sum_{A_\mathrm{S}} N_i m_i^2} = \frac{N_{A_\mathrm{S}} m_{A_\mathrm{S}}^2}{\sum_{i=1}^{N} m_i^2}, \end{aligned} \tag{3.11}$$

where N_{A_S} and m_{A_S} stand for the number and mass, respectively, of aggregates with aggregate size A_S; M is the total mass of all aggregates. The equations are normalized so that $\sum F_x(A_\mathrm{S}) = 1$.

Per-timestep averages are written to the `<output avg>` and distributions into the `<output distr>` file. Overall averages are appended as comments (with commented legend) to both `<output avg>` and `<output distr>` files.

Lastly, `DistrAgg` can calculate distribution of composition for aggregates with specified size(s) (`-c` option). Two versions of a 'composition distribution' are generated. The first is the distribution of numbers of each molecule type in the aggregates of that size. The second is a distribution of ratios of all possible molecular pairs in those aggregates The distribution of numbers of each molecule type is written into `<file>-<size>.txt` file, and the distribution of all ratios of all possible bead pairs is written into `<file>-ratio_<size>.txt` file; that is, two files are created for each aggregate size. In both cases, the number distribution for aggregates with aggregation number A_S, $F_{A_S}(i)$, is defined as

$$F_{A_S}(i) = \frac{N_{A_S,i}}{N_{A_S}}, \tag{3.12}$$

where N_{A_S} is the total number of aggregates with aggregation number A_S. The $N_{A_S,i}$ is the number of aggregates with size A_S that either contain i molecules of given type (the first distribution type) or has the ratio of molecules `mol1` and `mol2`; i.e., i = `mol1/mol2` (the second distribution type).

The `<avg file>` contains averages for all timesteps regardless of `-st`, `-e`, and `-sk` options. The starting and ending timesteps as well as the number of skipped timesteps are taken into account for all the distributions and overall averages.

The definition of aggregate size is flexible. If none of `-m`, `-x`, or `-only` options is used, aggregate size is the 'true' aggregation number, i.e., the number of all molecules in the aggregate; if `-m` is used, aggregate size is the sum of only specified molecule type(s); if `-x` is used, aggregates containing only specified molecule type(s) are disregarded; if `-only` is used, only aggregates composed of the specified molecule type(s) are taken into account. These options can be mixed. For example, consider a system containing three aggregates composed of various numbers of three different molecule types:

Molecule types	Aggregate composition
`Mol_A`	`Agg_1`: 1 `Mol_A` +2 `Mol_B` +3 `Mol_C` = 6 molecules
`Mol_B`	`Agg_2`: 1 `Mol_A` +2 `Mol_B` = 3 molecules
`Mol_C`	`Agg_3`: 1 `Mol_A` = 1 molecule

Here is a list of some of the possibilities depending on the option(s) used:

1) if none of `-m`, `-x`, `-only` is used, all three aggregates are counted and their sizes are their 'true' aggregation numbers, i.e., A_S = 6, 3, and 1
2) if `-m Mol_A Mol_B` is used, all three aggregates are counted, but their size is the sum of only `Mol_A` and `Mol_B` molecules: `Agg_1` – 3; `Agg_2` – 3; `Agg_3` – 1
3) if `-m Mol_B Mol_C` is used, `Agg_3` is not counted, because its size would be zero; `DistrAgg` would detect only two aggregates with sizes: `Agg_1` – 5; `Agg_2` – 2
4) if `-x Mol_A Mol_B` is used, `Agg_2` and `Agg_3` are not counted, because neither contains anything else than `Mol_A` and/or `Mol_B`; `DistrAgg` would detect only one aggregate with size: `Agg_1` – 6

5) if `-x Mol_A Mol_B` is combined with `-m Mol_A Mol_B`, `DistrAgg` would again detect only `Agg_1`, but its size would be 3
6) if `-only Mol_A Mol_B` is used, `Agg_1` is not counted, because it contains a molecule not specified by `-only`; `DistrAgg` would detect two aggregates with sizes: `Agg_2` – 3; `Agg_3` – 1
7) if `-only Mol_A Mol_B` is combined with `-m Mol_A`, the two detected aggregates have sizes: `Agg_2` – 1; `Agg_3` – 1
8) if `-only Mol_A Mol_B` is combined with `-x Mol_A`, only `Agg_2` is detected as it is the only one composed of only `Mol_A` and `Mol_B` molecules and its size would be 3
9) if `-only Mol_A Mol_B` and `-x Mol_A` are combined with `-m Mol_A`, the size of the one detected aggregate would be 1

Note that aggregate mass is always taken as the total mass, e.g., in the above points 8) and 9), the mass of the one detected aggregate would be the sum of masses of all the molecules in the aggregate even though the size is defined differently.

Should the `-c` option be used (without any of the `-x`, `-m`, or `-only` options), the output `<file>-<size>.txt` file would contain three data columns, one for each molecule type; the output `<file>-ratio_<size>.txt` file would contain three columns for the three ratios, that is `Mol_A/Mol_B`, `Mol_A/Mol_C`, and `Mol_B/Mol_C`.

Moreover, only a specified range of aggregate sizes can be taken into account (`-n <int> <int>` option). These sizes are defined by the `-m`, `-x`, and `-only` options as well.

Usage:

```
DistrAgg <input.agg> <distr file> <avg file> <options>
```

Mandatory arguments	
`<input>`	input structure file
`<input.agg>`	input `agg` file
`<distr file>`	output file with distribution of aggregate sizes
`<avg file>`	output file with per-timestep averages
Non-standard options	
`-n <int> <int>`	use aggregate sizes in a given range
`-m <mol name(s)>`	use number of specified molecule(s) as aggregate size
`-x <mol name(s)>`	exclude aggregates containing only specified molecule(s)
`-only <mol name(s)>`	use only aggregates composed of specified molecule(s)
`-c <file> <int(s)>`	save composition distribution for specified aggregate size(s) to `<output>` file
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `--help`, `--verbose`, `--silent`, `--version`	

Format of output files:
1) `<output_distr>` – distributions of aggregate sizes

- first line: AnalysisTools version
- second line: command used to generate the file
- third line: column headers
  - first is the aggregate size, `As` – either true aggregation number or the size specified by options
  - `F_n(As)`, `F_w(As)`, and `F_z(As)` are number, weight, and z distribution of aggregate sizes (Equation (3.11))
  - next is the total number of aggregates with specified size (sum from all timesteps)
  - the remaining columns (`<mol name>_n`) show average numbers of every molecule type in an aggregate with the specified size
- data lines follow
- second to last line: column headers for overall averages
  - `<As>_n`, `<As>_w`, and `<As>_z` are number, weight, and z averages, respectively, of aggregate numbers (see Equation (3.10) for definitions)
  - `<M>_n`, `<M>_w`, and `<M>_z` are number, weight, and z averages, respectively, of aggregate masses (see Equation (3.10) for definitions)
  - next are average numbers of every molecule type in an aggregate with the specified size (`<mol name>_n`)
  - average number of aggregates per timestep, `<n_agg>`
- last line: the overall averages

2) `<output avg>` – per-timestep averages
- first line: AnalysisTools version
- second line: command used to generate the file
- third line: column headers
  - first is simulation timestep
  - the calculated data follow: number, weight, and z average aggregate size (`<As>_n`, `<As>_w`, and `<As>_z`, respectively) and mass (`<M>_n`, `<M>_w`, and `<M>_z`, respectively)
  - the last column is the number of aggregates in the given step
- data lines follow
- the last two lines are the same as in `<output distr>`

3) `<file>-<size>.txt` from `-c` option – composition distribution of numbers of molecules
- first line: AnalysisTools version
- second line: command used to generate the file
- third line: number of aggregates of given size (sum from all timesteps)
- fourth line: column headers
  - first is the number of molecules in the aggregate
  - the rest are the number distributions of for each molecule type in the given aggregate size
- data lines follow

4) `<file>-ratio_<size>.txt` from `-c` option – composition distribution of ratios of molecular pairs
- first line: AnalysisTools version
- second line: command used to generate the file

- third line: number of aggregates of given size (sum from all timesteps)
- fourth line: column headers
  - first is the ratio of the two molecules (going from 0 to aggregate size with the interval of 0.1)
  - the rest are the ratios of all molecular pairs
- data lines follow

3.8 Info

This utility prints system information read from an input structure file, possibly augmenting it with information from another structure file (`-i <file>` option) and/or pruning the system to contain only beads from a provided coordinate file (`-c` option).

Note that using LAMMPS `data` , `lammpstrj` or `xyz` input structure file automatically reads coordinates in those files, but using `vtf` does not. This is because a `vtf` timestep does not necessarily contain all beads defined in the `vtf` structure section. If coordinates should be read from the input `vtf` structure file (and the system potentially pruned), just use the `-c` option with that filename.

For `vtf` structure files, the bead (and, consequently, molecule) type identification can be enhanced via the `--detailed` option. By default, bead types are defined solely based on their name; the other properties (mass, charge, and radius) are taken from the first `a[tom]` line with the bead of that name that contains the appropriate keyword. For example, lines

```
atom default name A m 1
atom 1 name B
atom 2 name B m 2
atom 3 name A m 2
atom 4 name A
```

would define two bead type called `A` and `B` with masses equal to 1 and 2, respectively. Using the `--detailed` option would split the `A` beads into three types: two with masses equal to 1 and 2, respectively, and a third one with undefined mass (it remains undefined because there is an ambiguity—should the last bead have mass 1 or 2?). On the other hand, the `B` beads would be assigned the same bead type with mass equal to 2 (there is no ambiguity because only one mass is specified for the `B` beads).

Using the `-i` option with an extra structure file assigns bead mass, charge, and/or radius from the extra file if the bead types in the original file have unspecified values; this is done only for bead types that share names. Also, if the two structure files share molecules that have the same bead type order (which can be initially checked via individual `Info` runs on the two files), bond types, angles, and angle types can be added to the molecule types in the original file. Note that to add this extra information, the bead types in the bonds/angles/dihedrals/impropers must be the same, i.e., all their parameters (name, mass, charge, radius) must be the same.

The `--mol` option transforms unbonded beads into one-bead molecules with the properties of the bead type (name, charge, mass). This can be useful, for example, to calculate particle clusters via the `Aggregates` utility because it can only determine

structures composed of molecules, not unbonded beads.

`Info` can also write this information into a new file (`-o` option) of any format. Note that only one timestep is read and, therefore, saved. Also, if no coordinates are read (i.e., neither `-c` option nor a structure file with coordinates is used), all coordinates in an output file would be 0.

There are a few options for the output structure file. For LAMMPS `data` file, `--mass` can be used to specify LAMMPS atom types by their mass, but print different charges in the `Atoms` section (i.e., `Info`-recognized bead types which differ only in charge are aggregated into the same LAMMPS atom types in the `Mass` section). Furthermore, extra atom types (with mass 1) can be added via the `-ebt` option; these atoms do not appear in the `Atoms` section of the `data` file and can be used as, e.g., extra atom type when srp potential (i.e., non-crossing bonds created by adding ghost particles between bonded beads) is used. For `vtf/vsf` file, `atom default` bead type can be specified via the `-def` option; by default, the bead type with the most unbonded beads is used as the `atom default` type.

For examples of `Info` usage, see the `Examples/Info` folder.

Usage: `Info <input> [options]`

Mandatory argument	
`<input>`	input structure file
Options for input files	
`-i <file>`	extra input structure file (default: None)
`-c <file>`	input coordinate file (default: None)
`--detailed`	only `vtf` structure file: detailed recognition for bead (and, consequently, molecule) types based on the bead names as well as charges, masses, and radii
Options for output file	
`-o <file>`	output coordinate file (default: None)
`-def <bead>`	only `vtf` structure file: use bead type `<bead>` for the 'atom default' line (default:bead type with the most unbonded beads)
`--mass`	only `data` structure file: define LAMMPS atom types by mass but print per-atom charges in the `Atoms` section (default: atom types are the same as `Info`-recognized bead types)
`--mol`	change unbonded beads into 1-bead molecules
`-ebt <int>`	only `data` structure file: number of extra atom types (for, e.g., srp)
Other options (see the beginning of Chapter 3)	
`-st`, `--verbose`, `--silent`, `--help`, `--version`	

3.9 JoinSystems

This utility takes two existing systems and joins them into a single system that contains all beads from the two coordinate files.

The second system can be offset against the first one (`-off` option). In each direction, all beads in the second system can be moved by specified distance (only positive numbers are allowed) or moved so that the centre of that box side is in the centre of the first box's side (use `c` instead of a number as an argument). The final box size is defined as the smallest box that encompasses the two original boxes after the second one is moved. However, the final box size can be redefined via the `-b` option; the centre of the new box is aligned with the centre of the original 'smallest encompassing box'. Note that no check is done whether the bead coordinates are within the new box.

Figure 3.3 illustrates the above-described behaviour (the red and blue lines and balls represent the two original systems while the green lines and transparent balls

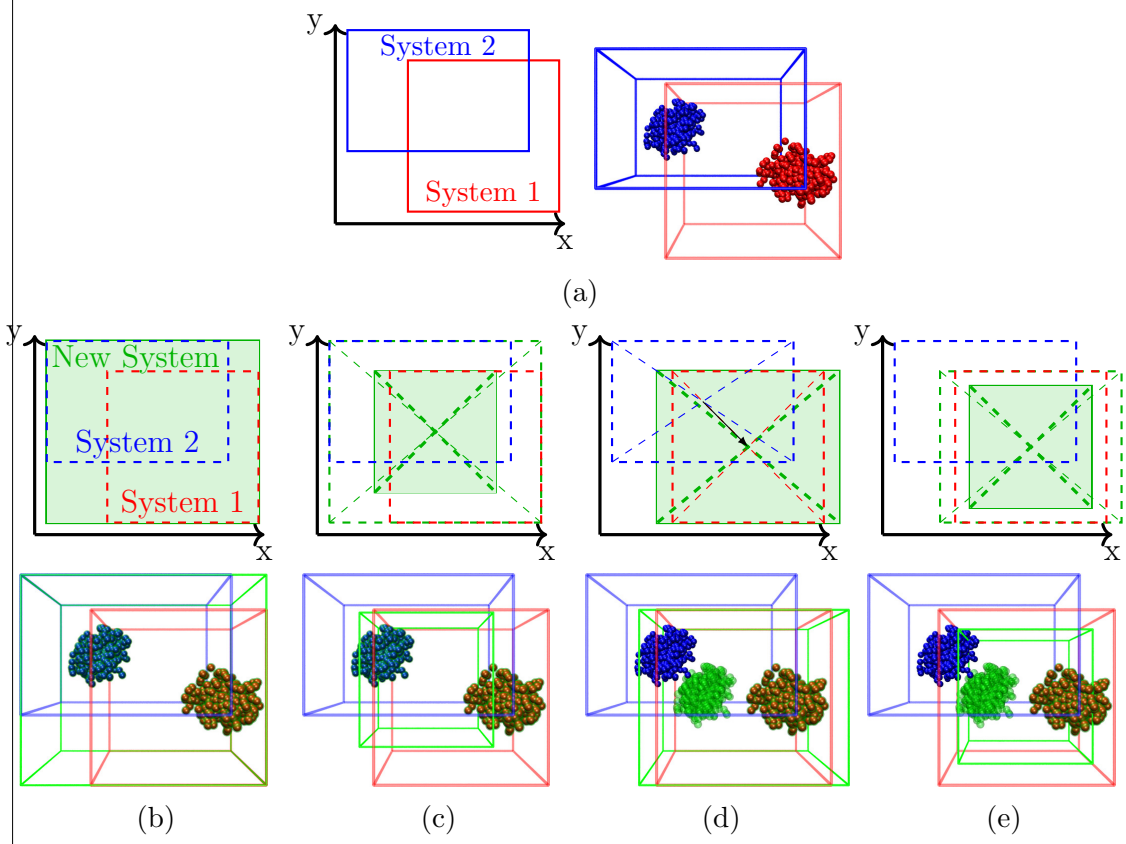


Figure 3.3: Schematic representation and corresponding snapshots illustrating `JoinSystems` behaviour. (a) The original systems: `System1.lammpstrj` (red) and `System2.lammpstrj` (blue) in the `Examples/JoinSystems` folder. The rest show examples of possible options (green rectangles and transparent balls represent the new systems): (b) no options; (c) `-b 20 20 20` option; (d) `-off c c c`; (e) `-b 20 20 20 -off c c c`, i.e., combining (c) and (d). These new systems are in the `Examples/JoinSystems` folder as `NewSystem_#.lammpstrj`, where `#` is `b`, `c`, `d`, or `e`.

represent the new system). Figure 3.3b shows the simplest case when two systems are put into one larger system. In figure 3.3c, the final box size is also redefined; the positions of the beads is unchanged, only the box size changes. Note that the absolute bead coordinates remain unchanged only when the output coordinate file supports defining the bounds of the simulation box; e.g., in `lammpstrj` file, the lower and upper bounds of the box are defined while for `vtf` file, only the sidelengths of the box are specified. Should `vtf` be used as the output file, the saved coordinates would, therefore, move so the box's origin is in the coordinate beginning.

Using the `-off c c c` option in figure 3.3d moves the second system (the blue one), so that the centre of its box (in all three dimensions) lies on top of the centre of the first system's box (see the arrow in the schematic in figure 3.3d); coordinates of only the second system are adjusted, changing the relative coordinates of beads from the two original systems. If the `-off` and `-b` options are combined (as in figure 3.3e), the system created via the `-off` option is then assigned a new box size.

These examples are also provided in the `Examples/JoinSystems` directory.

The output file can be any file that supports coordinates, and an extra structure file can be created via the `-o` option. Note that if an output `vcf` coordinate file is used, a `vsf` structure file (with the same name except for the extension) is also created.

Usage: `JoinSystems <input1> <input2> <output> [options]`

Mandatory arguments	
`<input1>`	first input coordinate file
`<input2>`	second input coordinate file
`<output>`	output file with the new system

Options	
`-off <x>|c <y>|c <z>|c`	offset of the second system against the first (`c` to place it in the middle of the first system)
`-b <x> <y> <z>`	side lengths of the new simulation box
`-o <file>`	optional output structure file

Other options (see the beginning of chapter 3) As two input coordinate/structure files are needed, some options have 1 or 2 appended to specify which file they correspond to
`-i1`/`-i2`, `-st1`/`-st2`, `--verbose`, `--silent`, `--help`, `--version`

3.10 Selected

This utility creates a new coordinate file. By default, the utility saves all bead and molecules types (basically transforming one coordinate file format into another), but using `-bt` and/or `-mt` options specifies which bead and/or molecule types to exclude from the output file. If `--keep` option is used, the specified bead and/or molecule types are instead the only ones that are written into the output file.

Besides the standard `-st`, `-e`, and `-sk` options, which timesteps to save can be explicitly specified via the `-n` option that can take a maximum of 100 arguments (the `-st`, `-e`, and `-sk` are then ignored). Also, using `--last` option saves only the last valid step from the input coordinate file (all the previous options are ignored if `--last` is used). If LAMMPS `data` file is used as a coordinate file, all the above options are ignored as the `data` file contains by definition only a single timestep.

There is also an option to remove periodic boundary conditions for molecules (i.e., to 'join' them) via the `--join` switch. Conversely, the simulation box can be wrapped (i.e., the periodic boundary conditions applied, putting all beads inside the box) via the `--wrap` switch. If both `--wrap` and `--join` options are used, the simulation box is first wrapped and then the molecules are joined.

Furthermore, some basic manipulations of the coordinates are possible:

1) using `-cx`, `-cy`, and/or `-cz` options, only beads whose coordinates fit the constraints are considered for saving (see below for what 'fit' means)
2) using `-sc <float>` option scales coordinates by dividing them by `<float>`
3) using `-m 3×<float>` option moves all coordinates by the given vector

Both `-cx/y/z` and `-m` options accept fractions of box size (i.e., numbers between 0 and 1) by default. To specify 'real' coordinates instead, use the `--real` option. The output simulation box remains unchanged, and the resulting coordinates may

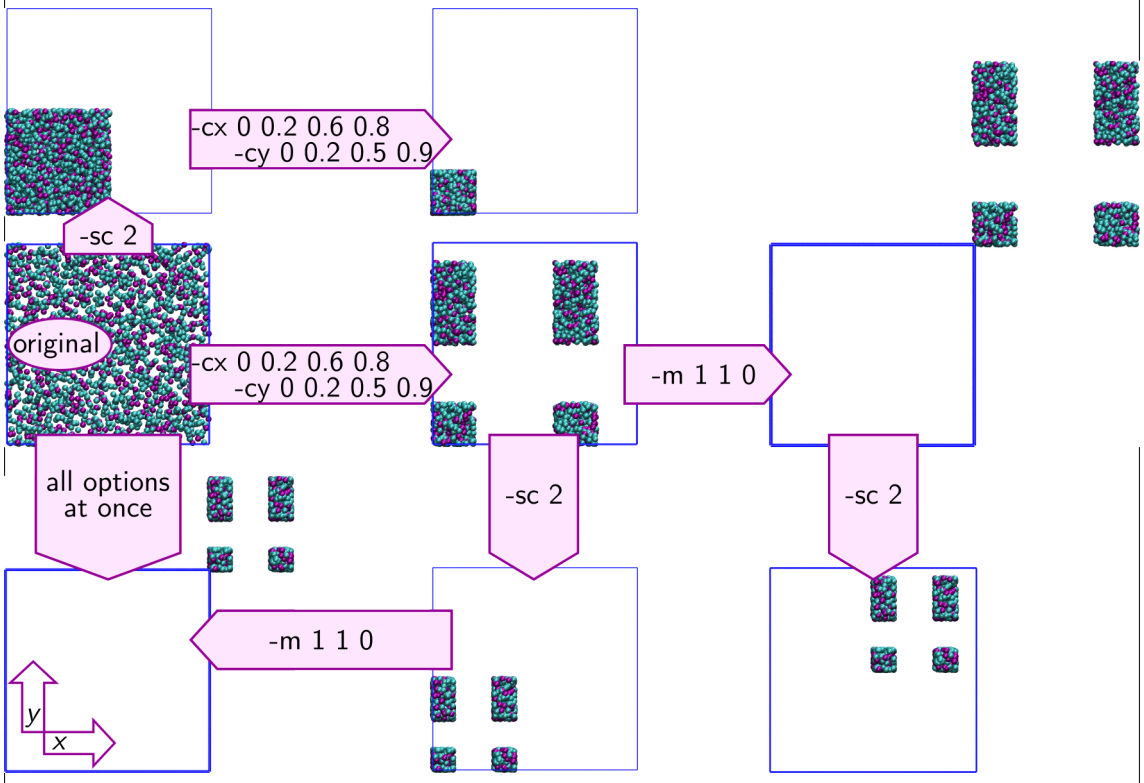


Figure 3.4: Some of the possible results when combining sequential runs of the `Selected` utility, namely (1) constraining x and y coordinates via `-cx 0 0.2 0.6 0.8 -cy 0 0.2 0.5 0.9`, (2) scaling (dividing) coordinates via `-sc 2`, and (3) moving coordinates via `-m 1 1 0`.

be outside the box. The `--wrap` and `--join` options are applied after these manipulations.

The constraining options accept multiples of number pairs for each axis; a bead coordinate must fall within at least one of the ranges, e.g., using `-cx 0 0.1 0.9 1` would save beads whose x coordinates are within 0 to 10% or 90% to 100% of the box size in the x-direction. Adding, e.g., `-cy 0.4 0.6` would save only the beads that fulfil the `-cx` option as well as this one, i.e., that y coordinate is between 40% and 60% of the simulation box.

Note that when a combination of these options is provided, the manipulations happen in the numbered order, e.g., using all three types of options, the coordinates are first constrained via `-cx/y/z` option(s), then all coordinates are scaled via `-sc`, and finally, the coordinates are moved via `-m`. This is represented in figure 3.4 by going from the 'original' system straight down. The sequence 'original'-right-down-left shows it is the same as running three `Selected` command in sequence. Figure 3.4 also shows examples of what can happen when `Selected` commands are run in sequence with different options. The `Selected` commands to generate these examples are in the `Examples/Selected` directory.

Usage: `Selected <input> <output> <bead type(s)> [options]`

Mandatory arguments	
`<input>`	input coordinate file
`<output>`	output coordinate file
Options	
`-bt <bead type>`	bead types to exclude
`-mt <mol type>`	molecule types to exclude
`--keep`	save only specified types instead of excluding them
`--join`	join molecules by removing periodic boundary conditions
`--wrap`	wrap simulation box (i.e., apply periodic boundary conditions)
`-n <int(s)>`	save only specified timesteps
`--last`	save only the last step
`-sc <float>`	divide all coordinates by given value
`-m 3×<float>`	add specified vector to all coordinates (`-sc` is applied first)
`-cx n×2×<float>`	constrain x-coordinates to specified dimension; multiple pairs possible
`-cy n×2×<float>`	same as `-cx` but with y-coordinates
`-cz n×2×<float>`	same as `-cx` but with z-coordinates
`--real`	use real coordinates instead of fractions of box size (for `-cx/y/z` and `-m` options)
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `-i`, `--verbose`, `--silent`, `--help`, `--version`	

3.11 Surface

Surface utility determines the average surface of structures such as membranes or polymer brushes and calculates the surface areas.

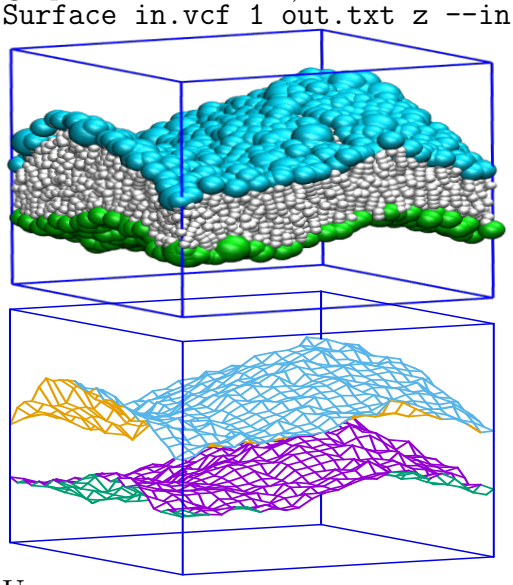
The utility cuts the plane perpendicular to the chosen axis into squares of size `<width>*<width>` and then finds the beads with the highest and lowest coordinates along the chosen axis whose other two coordinates fall into the square for each square on the plane. By default, it searches the chosen axis for the beads with the highest coordinate in the interval $\langle 0, (boxlength)/2\rangle$ and with the lowest coordinate in the interval $\langle (box\ length)/2; (box\ length)\rangle$, i.e., it assumes something such as a polymer brush on each wall. If `--in` option is used, it searches for a layer structure inside the box, i.e., it searches for the bead with the lowest and highest coordinates in the whole box, $\langle 0; (box\ length)\rangle$ (this mode finds surfaces for something like a bilayer).

Note that the utility does not determine any structures for now, therefore any molecules outside the brush/membrane can be recognized as a part of the surface (`-bt` and `-m` can be sometimes used to eliminate this problem).

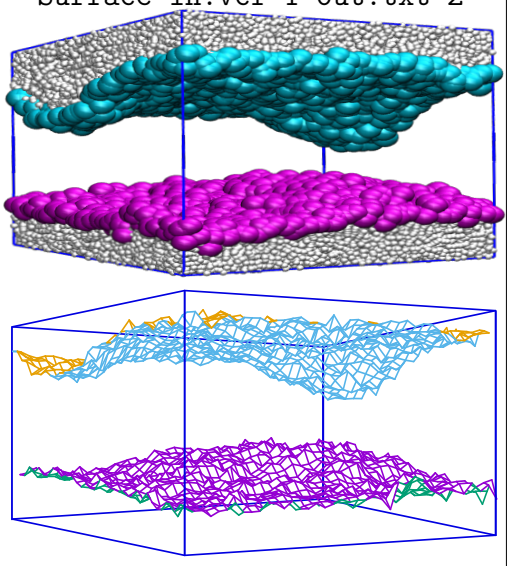
By default, all bead types and all molecule types are used, but using `-bt` and `-m` options, only specified bead and/or molecule types can be used. This is particularly useful when, e.g., other molecules solubilize inside the brush/membrane, leaving some molecules in the solution.

Following are two examples of usage with and without `--in` option (on top are snapshots with coloured balls representing the detected surface and on the bottom are graphs of the surface):

`Surface in.vcf 1 out.txt z --in`



`Surface in.vcf 1 out.txt z`



Usage:

Usage: `Surface <input> <width> <surf.txt> <area.txt> <axis> [options]`

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	side length of each square
`<surf.txt>`	output file with average surface

`<area.txt>`	output file with per-timestep areas
`<axis>`	direction in which to determine the surface: `x`, `y`, or `z`
Options	
`--in`	start from the box centre instead of from its edges
`--bonded`	use beads in molecules for surface recognition
`-bt <name(s)>`	use specified bead types for surface recognition
Other options (see the beginning of Chapter 3)	
`-st`, `-e`, `-sk`, `-i`, `--verbose`, `--silent`, `--help`, `--version`	

Format of output files:

1) `<surf.txt>` – 3D coordinates
   - first line: AnalysisTools version
   - second line: command used to generate the file
   - third line: column headers
     - first two numbers represent the centre of each square (being the coordinates on the sliced up plane); i.e., if `<width>` is 1, then the centre of the first square is 0.5 0.5, the centre of the second one is 0.5 1.5, etc.
     - the second two numbers are coordinates of the surface in the third dimension, i.e., along the chosen axis
   - data lines follow

2) `<area.txt>` – per-timestep areas (calculated as a sum of triangles)
   - first line: AnalysisTools version
   - second line: command used to generate the file
   - third line: column headers
     - first is simulation timestep
     - the rest are areas of the top, bottom, and central surfaces
   - data lines follow
   - second to last line: column headers for overall averages
   - last line: the overall averages