AnalysisTools user manual

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

Version 3.3 (Released August 18, 2020)

Contents

1. Introdution

This set of utilities analyses trajectories of particle-based simulations. The emphasis is on assembly of molecules such as self-assembly of polymers and the properties of the resultant aggregates. Note that throughout the utilities and this manual, the term aggregates refers to any supramolecular structure (e.g., a bilayer, a micelle, or a vesicle).

The package also contains some utilities to generate initial configurations (or adapt existing) as well as to create input files for the LAMMPS molecular dynamic simulator and the DL_MESO simulation package.

It mainly works with vtf trajectory files, but it can also read `xyz` or LAMMPS custom format coordinate files as well as configuration files for LAMMPS and DL_MESO simulation package.

2. Installation

All programs can be compiled on linux using `cmake` and subsequently running `make` on the generated `Makefile`. It requires `C` and `FORTRAN` compilers.

The compilation should be done in a separate directory, such as `$ROOT_DIR/build` (`$ROOT_DIR` is the `AnalysisTools` root directory). To create the `Makefile`, simply run `cmake $ROOT_DIR` in the `build` directory. All utilities are then compiled by running `make` without any arguments. To compile individual utilities, run `make <utility name>`. The binaries will be in `build/bin` directory.

Following is a one line command to create the `build` directory and compile all utilities (assuming the working directory is AnalysisTools root directory):

```
mkdir build; cd build; cmake ../; make
```

3. 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 4.

3.1 System information

Most of the utilities read system information from vtf files, sometimes complemented by a `FIELD` file (input file for DL_MESO simulation package) or a LAMMPS `data` file.

The system information can either be split into a `vsf` structure file (Section 3.1.1) and a `vcf` coordinate file (Section 3.1.2), or presented in a single `vtf` file containing both the structure and coordinate sections; note that when individual `vsf/vcf` files are mentioned in the manual, they can usually be viewed as `vsf/vcf` sections of the combined `vtf` file. The advantage of using separate structure and coordinate files is that the same coordinate file can be used with different structure files to get different results.

In general, the utilities consider only bead types that are present in a `vcf` file (i.e., bead types present in a `vsf` file but missing from a `vcf` file are ignored).

All utilities assume cuboid simulation box with dimensions from 0 to N, where N is the side length of the box (which can be different in all three dimensions).

Chapter 5 briefly describes how the data are read.

3.1.1 `vsf` structure file

The structure file contains all information about all beads and bonds except for their Cartesian coordinates in two types of entries. The first type of entry, an atom line, contains a bead definition. Each atom line is the description of a single bead following these rules:

- the line starts with `a[tom]` (i.e., `atom` or just `a`)
- the second string is a either a bead index (starting at 0) or the `default` keyword; every bead that is not explicitly written in the bead definition lines is of the `default` type
- the line contains bead name as `n[ame] <char(16)>`

- if the bead is in a molecule, the line contains both molecule name (`res[name] <char(8)>`) and molecule id (`resid <int>`) starting at 1
- `m[ass]`, `charge|q` (i.e., `charge` or `q`), and `r[adius]` keywords are read if present
- other keywords are ignored

The types of beads are generally identified solely by their name, i.e., should two beads share a name but differ in other characteristics (i.e., mass, charge, or radius), they will be considered as belonging to the same type. The mass, charge, and radius of each bead type is equal to that for the topmost bead with that name in the `vsf` file that contains the corresponding keyword (if the keyword is not present in any line of the given bead type, that characteristic is undefined). However in some cases, beads that share a name but differ in their characteristics, are treated as different bead types (with `_<int>` attached to its name); notably, `TransformVsf` does this to produce a new `vsf` file with these names. For example, the lines

```
atom default name A mass 1 q 0
a 0 name B mass 2 r 1
a 2 name B m 1 charge -1
atom 4 name B
```

would ordinarilly lead to two `A` type beads (both with mass 1, charge 0, and undefined radius) and three `B` type beads (all with mass 2, charge -1 and radius 1). However, `TransformVsf` would split the `B` type beads into three distinct types one `B_1` bead (with mass 2, undefined charge, and radius 1), one `B_2` bead (with mass 1, charge -1, and undefined radius), and one `B_3` bead (with all three characteristics undefined). Note that if the `a 0...` line weren't there, the `B` beads would be the same even for `TransformVsf`; i.e., if there's only one value for any characteristic, all beads sharing the same name will have that value even if it is not specified on every atom line.

The second type of entry is a bond line defining connectivity between beads. Each bond line starts with `b[ond]` followed by two bead indexes (corresponding to an atom line) separated by a colon. Note that the colon must be right after the first index, i.e., no white space between the number and the colon is allowed. The double colon or comma separated bond entries are not allowed in `AnalysisTools`.

In general, `AnalysistTools` recognizes types of molecules only by their names (taking the connectivity and bead order of the first molecule of every type); if two molecules share the same `res[name]`, but differ in connectivity or number of beads, the utilities will exit with an error. Again, the `TransformVsf` utility goes to more detail, distinguishing molecule types by all its characteristics. For example, the lines

```
a 0 A m 1 res Mol resid 1
a 1 B m 1 res Mol resid 1
```

```
a 2 B res Mol resid 1
a 3 A res Mol resid 2
a 4 B res Mol resid 2
a 5 B m 2 res Mol resid 2
b 0:1
b 1:2
b 3:4
b 4:5
b 3:5
```

would ordinarilly lead to two linear `Mol` type molecules, each containing one `A` and two `B` beads and connected `A-B-B`. However, `TransformVsf` would split the molecules to one linear `Mol_1` type molecule (with one `A` and two `B_1` beads, connected `A-B_-1-B_1`) and one ring `Mol_2` type molecule (with one `A`, one `B_1`, and one `B_2` beads, connected in a triangle); `B_1` bead has mass 1 and `B_2` bead mass 2.

The `vsf` file can contain any number of blank lines and comments (lines beginning with `#`).

See complete examples of the `vsf` file in the `Examples` directory.

3.1.2 `vcf` coordinate file

The coordinate file contains Cartesian coordinates of the beads and the size of the cuboid simulation box. Coordinates are read from a `vcf` file containing either ordered timesteps or indexed timesteps; all timesteps must be of the same type and contain the same beads.

An ordered `vcf` file must contain all beads defined in the `vsf` file, while an indexed `vcf` file can contain only a subset of defined beads. Both indexed and ordered `vcf` files contain a line before every timestep specifying the file type, `timestep o[rdered]/i[ndexed]`; the `timestep` keyword can be omitted. In both ordered and indexed `vcf` files, the size of the simulation box is given by a line appearing before the first timestep `pbc <float> <float> <float>`.

The `vcf` file may contain comment lines (beginning with `#`) and blank lines between timesteps, but the coordinate block must be continuous.

The coordinate blocks in an ordered `vcf` file contain only Cartesian coordinates – every line has the `<x> <y> <z>` format The beads are written in an ascending order of their indices as defined in the `vsf` file. All beads defined in the `vsf` file must be present in the ordered `vcf` file.

The coordinate blocks in an indexed `vcf` file contain not only Cartesian coordinates but also bead indices – every line has the `<index> <x> <y> <z>` format. An indexed timestep does not have to contain all beads defined in the `vsf` structure file; however, `AnalysisTools` utilities work with whole sets of beads, that is, when one

bead of a specific type is missing, all beads of that type (or with the same name) must be omitted as well. The beads do not have to be ordered in any specific way.

3.1.3 DL_MESO FIELD file

3.1.4 LAMMPS data file

3.2 Aggregate file (agg)

A file.agg is generated using Aggregates (or Aggregates-NotSameBeads) utility. The file contains information about the number of aggregates in each timestep and which molecules and monomeric (i.e., unbonded) beads belong to which aggregate. It serves as an additional input file for utilities that calculate aggregate properties; agg file is, therefore, linked to the vcf file that was used to generate it.

The agg file is a simple text file. The first line contains the command used to generate it – parts of this command may be necessary for subsequent analysis of aggregates. The second line is blank, and from the third line, the data for individual timesteps are shown. It follows these rules:

- each timestep starts with Step: <int> (only Step keyword is read by the utilities)
- the second line is the number of aggregates in the given timestep and is followed by a blank line
- there are two lines for each aggregate:
  (1) number of molecules in the aggregate followed by their indices taken from the vsf file (resid indices)
  (2) number of monomeric beads in the aggregate followed by their indices taken from the vsf file (atom indices)
- no blank or comment lines are allowed inside the aggregate block
- not all molecules present in the vcf file used to generate this file must be present in every timestep

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

When the keyword Last is present instead of Step, it signalises the end of the agg file; no utility will read anything beyond this keyword.

Besides using this file for further analysis by other utilities, the indices can be used in vmd to visualize, e.g., only a specific aggregate by using resid 1 2 3 in the Selected Atoms box in vmd.

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

4. Utilities

All utilities have command line help with short description when `-h` argument is used. Besides `-h`, most of the utilities have several standard command line options that are the same. The standard options can be used with any utility unless stated otherwise.

Standard options	
`-i <name>`	use custom structure file instead of `traject.vsf` (`vsf` or `vtf` format)
`-v`	verbose output that provides information about all bead and molecule types
`-s`	run silently, i.e., without any output at all (overrides `-v` option)
`--script`	no progress output (useful if output is routed to a file)
`-h`	print this help and exit
`--version`	print version of the utilities and exit

4.1 AddToSystem

This utility takes an existing system specified by `vtf` file(s) and adds new beads into it, placing them either randomly or using provided coordinates. The utility requires an input coordinate file containing all beads (i.e., all beads defined in the structure file must be present in the coordinate). If an incomplete coordinate file is provided, `AddToSystem` exhibits undefined behaviour, i.e., it can run without problems, exit with an error, or freeze.

The utility has two modes: by default, it reads what is to be added from a `FIELD`-like file, placing the new beads and molecules randomly (Subsection 4.1.1); if, however, `-vtf` option is used, the utility reads the provided `vsf` and `vcf` files, using coordinates from those (Subsection 4.1.2).

4.1.1 Random placement

The random coordinates of added beads can be constrained via several options; for added molecules, either the molecule's first bead (default behaviour) or its geometric centre (`-gc` option) obey the constraints. The coordinates of the remaining beads

in the molecule are governed by the coordinates in the FIELD-like file. Therefore, not all the molecular beads necessarily obey the constraining rules. Molecules are added with a random orientation.

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 directions. In (i), options `-ld` and/or `-hd` specify the distance; if present, these must be accompanied by `-bt` option. Then, the new beads are placed at least `-ld <float>` and at most `-hd <float>` distance from beads specified by the `-bt` option. In (ii), options `-cx`, `-cy`, and `-cz` basically change the box size for the added beads; for example, `-cx 5 9` would generate x coordinates only in the interval $\langle 5, 9)$. 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 will run forever.

As stated above, the structure and number of added molecules and monomeric beads are read from a FIELD-like file. This file must contain a `species` section followed by a `molecule` section as described in the DL_MESO simulation package. Following is a short description of the two sections.

The `species` section contains the number of bead types and their properties:

```
species <int>
<name>  <mass> <charge> <number of unbonded beads>
```

The first line must start with `species` keyword followed by the number of bead types. For each bead type, a single line must contain the name of the bead, its mass and charge, and a number of these beads that are not in a molecule (i.e., monomeric or unbonded beads). No blank lines are allowed in the section.

The `molecule` section, which comes after the `species` section, contains information about structure and numbers of molecules to be added:

`molecule <int>`	Number of types of molecules
`<name>`	Name of the first molecule type
`nummols <int>`	Number of `<name>` molecules
`beads <int>`	Number of beads in each `<name>`
`<bead name> <x> <y> <z>`	One line for each of the `<int>` beads,
`...`	specifying bead name and its
`<bead name> <x> <y> <z>`	Cartesian coordinates
`bonds <int>`	Number of bonds in `<name>`
`harm <indices> <length> <strength>`	One line for each of the `<int>` bonds
`...`	containing two bead indices and
`harm <indices> <length> <strength>`	bond length and strength
`angles <int>`	Number of angles in `<name>` (optional)
`harm <indices> <angle> <strength>`	One line for each of the `<int>` angles

`...`	containing three bead indices,
`harm <indices> <angle> <strength>`	angle and angle strength
`...`	Anything beyond here is ignored
`finish`	End of `<name>`'s description

The `molecule` keyword specifies the number of molecule types, that is the number of `finish` keywords that must be present. The `<bead name>` must be present in the `species` section. The `harm` string in the `bond` and `angle` lines is ignored as `AddToSystem` assumes harmonic potential for both. The indices in `bond` and `angle` lines run from 1 to the number of beads in the molecules and are ordered according to the `beads` part of the section. Anything else (such as dihedral angles) is ignored. While a molecule does not have to have angles, at least one bond per molecule is required. If no molecules are to be added, the line `molecule 0` is required. Examples of `FIELD`-like files are in the `Examples/AddToSystem` directory.

4.1.2 Provided coordinates

If `-vtf` option is used, provided `vsf` and `vcf` files are read instead of a `FIELD`-like file. `AddToSystem` incorporates the beads and molecules from these files, using the provided coordinates instead of generating new ones.

The `-vtf` option has priority over default behaviour, that is, if it is present, the utility ignores a `FIELD`-like file as well as the constraining options.

The side lengths of the new simulation box are always taken as the higher ones from the original system and the system to be added.

In both modes, the new unbonded beads appear in the resulting files after the unbonded beads from the original system (but before any bonded beads). If molecules are added, `AddToSystem` places them at the end.

By default, the new beads are added to the system, increasing the total number of beads. However, if `--switch` is used, the new beads replace beads in the original system. The replaced beads are either neutral beads with the lowest indices (as per ordering in the provided structure file) or beads of the type specified using `-xb` option. In both cases, the beads to be replaced must be unbonded.

The size of the new simulation box can be changed using the `-b` option. There are no constraints on the size of the cuboid box; if the new box is small, not all beads from the original system are necesserily inside the box (although the added once always are). The origin of the new box coincides with the origin of the original box unless `--centre` option is used; in that case, the centres of the two boxes coincide.

`AddToSystem` creates `vcf` and `vsf` files containing the new system; the coordinates can also be written into an `xyz` file.

Several examples of using the utility are provided in the `Examples/AddToSystem` directory.

Usage:

```
AddToSystem <input.vcf> <out.vsf> <out.vcf> <options>
```

Mandatory arguments	
`<input.vcf>`	input coordinate file (either `vcf` or `vtf` format)
`<out.vsf>`	output `vsf` structure file for the new system
`<out.vcf>`	output `vcf` coordinate file for the new system
Non-standard options	
`-st <int>`	`<input.vcf>` timestep to add new beads to (default: 1)
`-xyz <name>`	also save coordinates to `xyz` file
`--switch`	replace original beads instead of increasing the total number of beads
`-b <x> <y> <z>`	side lengths of the new simulation box
`--centre`	place the original simulation box in the middle of the new one
Random placement	
`-f <name>`	`FIELD`-like file specifying additions to the system (default: `FIELD`)
`-ld <float>`	lowest distance from beads specified by `-bt` option
`-hd <float>`	highest distance from beads specified by `-bt` option
`-bt <bead names>`	bead types to use in conjunction with `-ld` and/or `-hd` options
`-cx <num> <num2>`	constrain x coordinate to interval $\langle num, num2)$
`-cy <num> <num2>`	constrain y coordinate to interval $\langle num, num2)$
`-cz <num> <num2>`	constrain z coordinate to interval $\langle num, num2)$
`-gc`	use molecule's geometric centre for distance check
Provided coordinates	
`-vtf <vsf> <vcf>`	add ready-made system from `vtf` files instead of randomly placing beads read from the `FIELD`-like file

4.2 Aggregates and Aggregates-NotSameBeads

These utilities determine which molecules belong to which aggregates (or a different compact 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 in `.agg` text file described in Section **??**.

Note that throughout `AnalysisTools`, an aggregate refers to any structure formed from molecules.

The number of contact pairs, the distance, and bead type(s) to use for aggregate determination are all arguments of the utilities. Any molecule type(s) can be excluded from aggregate determination (`-x <mol name(s)>` option); they are also excluded from the output `agg` file). Moreover, any molecules close to specified molecule(s) can be excluded (`-xm <mol name(s)>` option); here, 'close' means any of the bead types used in aggregate determination is closer than `<distance>` to any bead of the specified molecule.

Also, periodic boundary conditions can be removed from whole aggregates and the new coordinates saved to an indexed `vcf` file (`-j` option). Therefore aggregates will not be split by simulation box boundaries when, for example, visualizing the molecules with vmd.

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.

Usage:

`Aggregates` (or `Aggregates-NotSameBeads`) `<input> <distance> <contacts>`

`<output.agg> <bead type name(s)> <options>`

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<distance>`	minimum distance for two beads to be in contact (thus constituting a contact pair)
`<contacts>`	minimum number of contact pairs between two molecules to be in one aggregate
`<output.agg>`	output `agg` file (must end with `.agg`) with aggregate information
`<bead type(s)>`	bead type name(s) to use for determining contact pairs (at least two for `Aggregates-NotSameBeads`)
`<options>`	

Non-standard options	
`-x <mol name(s)>`	exclude specified molecule type(s) from aggregate determination (and from the output `agg` file)

`-xm <mol name(s)>`	exclude molecules that are close to specified molecule type(s)
`-j <output.vcf>`	output `vcf` file with coordinates of joined aggregates (i.e., without periodic boundary conditions)

4.3 AngleMolecules

This utility calculates angles between beads in each molecule of specified molecule type(s). The beads do not have to be connected, so the angle does not have to be between two bonds.

Using `-n` option, the angle is specified by three bead indices taken from the bead order in the `vsf` file. These indices go from 1 to N, where N is the number of beads in the molecule type. Generally, the numbering of beads inside a molecule is made according to the first molecule of the given type in `vsf` file. For example, assume that beads of the first molecule called `mol` in the `vsf` file are ordered `A` (`vsf` index 123), `B` (`vsf` index 124), `C` (`vsf` index 200). Then, bead `A` is 1, bead `B` is 2, and `C` is 3.

More than one angle can be specified (i.e., a multiple of three numbers have to be supplied to the `-n` option.). For example, assuming indices `1 2 3 1 3 2` are specified, two angles will be calculeted. The first angle will be between lines defined by beads with indices `1 2` and `2 3`; the second one will be between lines defined by beads with indices `1 3` and `3 2`. An angle is calculated in degrees and is between 0 and 180°.

The utility calculates distribution of angles for each specified trio of bead indices for each molecule type and prints overall averages at the end of an `<output>` file. If `-a` option is used, it can also write all the angles for all individual molecules in each timestep (i.e., time evolution of the angle for each individual molecule).

Usage:

```
AngleMolecules <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
`<mol name(s)>`	molecule name(s) to calculcate angles for
`<output>`	output file for distribution

Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for molecules do not have to be removed)

`-a <name>`	write all angles for all molecules in all timesteps to `<name>`
`-n <ints>`	multiple of three indices for angle calculation (default: 1 2 3)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

Format of output files:
(1) `<output>` – distribution of angles
- first line: command used to generate the file
- second line: angle-specifying bead indices (the dash-separated numbers correspond to indices inside every molecule and are the same as the arguments to the `-n` option) with the numbers in brackets corresponding to nth column of data for each molecule type
- third line: numbering of columns (i.e., column headers)
  - first 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: the range for each molecule type specifies which column numbers correspond to the calculated angles for that particular molecule type and the order of angles is given by the second line
- last two lines: arithmetic means for each calculated angle (last line) and headers (second to last line) that again give range of columns in the last line for each molecule type

(2) `-a <name>` – all angles for all molecules in all timesteps
- first and second lines are the same as for `<output>`
- third line: column headers
  - first is simulation timestep
  - the rest are the calculated data: the range for each molecule type corresponds to the number of molecules of the given type times the number of calculated angles; for each molecule the angles are ordered according to the second line

4.4 Average

This utility uses the binning method to analyse data in a text file. It does not use any of the standard options and prints the result only to standard output (e.i., screen).

`Average` calculates average value, statistical error, and estimate of autocorrelation time τ. Empty lines and comments (lines beginning with `#`) are skipped. `Average` prints four numbers to screen: `<n_blocks> <average> <std error> <tau estimate>`:

`<n_blocks>`	number of blocks used for the binning analysis
`<average>`	simple arithmetic average
`<std error>`	one-σ statistical error
`<tau estimate>`	estimate of autocorrelation time τ

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

$$\langle \mathcal{O} \rangle = \frac{1}{N} \sum_{i=1}^{N} \mathcal{O}_i, \tag{4.1}$$

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:

$$\epsilon^2 = \frac{\sigma^2_{\mathcal{O}_i}}{N}, \tag{4.2}$$

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

$$\sigma^2_{\mathcal{O}_i} = \frac{1}{N-1} \sum_{i=1}^{N} (\mathcal{O}_i - \langle \mathcal{O} \rangle)^2. \tag{4.3}$$

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 (4.2) 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:

$$\mathcal{O}_{\mathrm{B},n} = \frac{1}{k} \sum_{\substack{i=1+ \\ (n-1)k}}^{kn} \mathcal{O}_i. \tag{4.4}$$

If $k \gg \tau$, the blocks are assumed to be uncorrelated and equation (4.2) can be used:

$$\epsilon^2 = \frac{\sigma^2_\mathrm{B}}{N_\mathrm{B}} = \frac{1}{N_\mathrm{B}(N_\mathrm{B}-1)} \sum_{n=1}^{N_\mathrm{B}} (\mathcal{O}_{\mathrm{B},n} - \overline{\mathcal{O}})^2. \tag{4.5}$$

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

$$\tau_\mathcal{O} = \frac{k\sigma^2_\mathrm{B}}{2\sigma^2_{\mathcal{O}_i}}. \tag{4.6}$$

A way to quickly get τ estimate is to use a wide range of `<n_blocks>` values and plot the `<tau estimate>` values as a function of `<n_blocks>`. Because the number of data points in one block should be significantly larger than τ (e.g., ten times larger), plotting $f(x)$ =(number of data lines in the file)/$(10x)$ will produce an exponential function that intersects the `<tau estimate>` line. A value of `<tau estimate>` near the intersection (but to the left, where the exponential is above `<tau average>`) can be considered a safe estimate of τ.

Usage:

```
Average <input> <column> <discard> <n_blocks>
```

Mandatory arguments	
`<input>`	input text file
`<column>`	column number in `<input>` for data analysis
`<discard>`	number of data lines to discard from the beginning of `<input>`
`<n_blocks>`	number of blocks for binning analysis

4.5 BondLength

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 4.3). 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

4.6 Config

This utility creates DL_MESO `CONFIG` file. It requires input coordinate file with all beads; otherwise the utility will still run without any error, but it will produce an incomplete `CONFIG`. Config always prints first unbonded beads and then beads in molecules (required by DL_MESO).

Usage:

```
Config <input.vcf> <options>
```

Mandatory argument	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
Non-standard option	
`-st <int>`	timestep for creating `CONFIG` file from (default: last step)

There is also utility `Config_from_xyz` which takes a `xyz` coordinate file and creates the DL_MESO `CONFIG` file. Because `xyz` file does not contain information about the simulation box, the resulting `CONFIG` file must be modified manually – `Config_from_xyz` prints `x`, `y`, and `z` into the output file where box dimensions should be.

Usage:

```
Config_from_xyz <input.xyz> <options>
```

Mandatory argument	
`<input.xyz>`	input `xyz` coordinate file
Non-standard option	
`-st <int>`	timestep for creating `CONFIG` file from (default: last step)

4.7 DensityAggregates

This utility calculates radial density profiles (RDPs, or radial number densities) for bead types in an aggregate with specified size (the number of molecules or aggregation number, A_S) from its centre of mass.

$\mathrm{RDP}_i(r)$ of bead type i, where r is distance from an aggregate's centre of mass, is the number of these beads in a spherical shell between the distances r and $r + \mathrm{d}r$ (in `DensityAggregates`, dr is the `<width>` argument) divided by the volume of this shell. The utility also prints radial number profile ($\mathrm{RNP}_i(r)$), or the number of beads in each shell without dividing it by its volume. In addition, it prints one-σ error for both RDP and RNP.

Composition of an aggregate with given size (i.e., average numbers of different types of molecules in that aggregate) is appended to the output file.

Instead of 'true' aggregate size, a number of molecules of specific type(s) can be used (`-m` option). For example, an aggregate containg 1 `Mol_A` molecule and 2 `Mol_B` molecules (i.e., three molecules in all) can be specified in several ways:

(1) with `<agg size(s)>` of 3;
(2) with `<agg size(s)>` of 3 and `-m Mol_A Mol_B`;
(3) with `<agg size(s)>` of 1 and `-m Mol_A`; or
(4) with `<agg size(s)>` of 2 and `-m Mol_B`.

Care must be taken when different molecule types share the same bead type. If one bead type is in more molecule types, the resulting density for that bead type will be the sum of its densities from all molecule types it appears in. The `-x` option can overcome this – specific molecule types can be excluded from density calculations, i.e., density of beads in the excluded molecule types will not be calculated. For example, assume two molecule types – `Mol_1` and `Mol_2`. `Mol_1` contains bead types `A` and `B`; `Mol_2` contains bead types `A` and `C`. Depending on whether and how the `-x` option is used, the utility will calculate:

(1) densities of `A`, `B`, and `C` beads (density of `A` beads is a sum from both molecules), if no `-x` is used;
(2) densities of only `A` and `B` beads (with `A` beads only from `Mol_1`), if `-x Mol_2` is used;
(3) densities of only `A` and `C` beads (with `A` beads only from `Mol_2`), if `-x Mol_1` is used; or
(4) no densities at all if `-x Mol_1 Mol_2` is used.

Therefore, to be able to plot density of `A` beads from `Mol_1` and `Mol_2` separately, (2) and (3) should be used (i.e., `DensityAggregates` should be run twice).

Usage:

```
DensityAggregates <input> <input.agg> <width> <output> <agg
size(s)> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<input.agg>`	input `.agg` file
`<width>`	width of a single distribution bin
`<output>`	output file(s) (one per aggregate size) with automatic `#.rho` ending (`#` is aggregate size)
`<agg size(s)>`	aggregate size(s) (the number of molecules in an aggregate or the aggregation number, A_S) to calculcate density for
Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for aggregates do not have to be removed)
`-n <int>`	number of bins to average to get smoother density (default: 1)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-m <mol name(s)>`	instead of 'true' aggregate size, use the number of specified molecule type(s) in an aggregate
`-x <mol name(s)>`	exclude specified molecule type(s) (i.e., do not calculate density for beads in molecules `<mol name(s)>`)

Format of output files:
(1) `<output>#.rho` – bead densities for one aggregate size
- first line: command used to generate the file
- second line: the order of data columns for each bead type – `rdp` is radial density profile, `rnp` radial number profile and `stderr` are one-σ errors for `rdp` and `rnp`
- 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 number specifies the first column with data for the given bead type (i.e., `rdp` column)
  - last line contains the total number of aggregates the density was calculated for
- second to last line: column headers for average numbers of different molecules in the given aggregate
- last line: average numbers of the molecules

4.8 DensityBox

This utility calculates number density for all bead types in the specified direction of the simulation box. 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 of the slices.

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, it's density will be averaged over all molecule types it appears in. In this case, however, the `-x` option can be used to first run the utility and exclude one molecule type and then rerun it, excluding the other molecule type. Thus, two output files are generated, each missing densities from the specified molecule types.

Usage:

```
DensityBox <input> <width> <output> <axis> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	width of each bin of the distribution
`<output>`	output file with automatic `<axis>.rho` ending
`<axis>`	direction in which to calculate density: `x`, `y`, or `z`
Non-standard options	
`-n <int>`	number of bins to average to get smoother density (default: 1)
`-st <int>`	starting timestep for calculation (default: 1)
`-x <mol name(s)>`	exclude specified molecule type(s) (i.e., do not calculate density for beads in molecules `<mol name(s)>`)

Format of output files:
(1) `<output>` – bead densities

- 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: each number corresponds to the density of the specified bead type

4.9 DensityMolecules

This utility works similarly to `DensityAggregates`, only instead for whole aggregates, RDPs are calculated for individual molecules. Similarly to `DensityAggregates`, the output file(s) also contain statistical errors and radial number profiles.

By default, the utility calculates RDPs from the molecule's centre of mass, but any bead in the molecule (with an index according to `vsf` – similarly to `-n` option in `AngleMolecules`, Section 4.3) can be used instead (`-c` option).

Usage:

```
DensityMolecules <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 automatic `<mol_name>.rho` ending (one file per molecule type is generated)
`<mol name(s)>`	molecule name(s) to calculcate density for
Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for molecules do not have to be removed)
`-n <int>`	number of bins to average for smoother density (default: 1)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-c <name> <int>`	use specified bead in a molecule `<name>` instead of its centre of mass

Format of output files:

(1) `<output><mol_name>.rho` – bead densities for one molecule

- first line: command used to generate the file
- second line: the order of data columns for each bead type – `rdp` is radial density profile, `rnp` is radial number profile and `stderr` are one-σ errors for `rdp` and `rnp`
- 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 number specifies the first column with data for the given bead type (i.e., `rdp` column)

4.10 DihedralMolecules

This utility calculates angles between specified planes in each molecule of provided molecule type(s). The planes in a molecule are arbitrary, so they can represent true dihedral angles or improper dihedrals.

The angle is specified by six bead indices (according to the order of beads in the molecule in `vsf` – similarly to the `-n` option in `AngleMolecules`, Section 4.3). The first three indices specify one plane and the next three the other. For example, assuming indices `1 2 3 4 5 6`, the first plane is specified by the first three beads in the molecule; second plane by the next three beads (beads `4 5 6`). The default indices (i.e., if `-n` option is not used) are `1 2 3 2 3 4`. More than one angle can be specified (i.e., a multiple of six numbers can be supplied to the `-n` option.).

The utility calculates a distribution for each specified angle for each molecule type and prints overall averages at the end of `<output>` file. If `-a` option is used, it also writes all the angles for all individual molecules in each timestep (i.e., time evolution of the angle for each individual molecule).

Usage:

```
DihedralMolecules <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 (in degrees)
`<output>`	output file for distribution
`<mol name(s)>`	molecule name(s) to calculcate angles for
Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for molecules do not have to be removed)
`-a <file>`	write all angles for all molecules in all timesteps to `<file>`
`-n <ints>`	multiple of six indices for angle calculation (default: 1 2 3 2 3 4)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

Format of output files:
(1) `<output>` – distribution of angles

- first line: command used to generate the file
- second line: calculated angles (the dash-separated numbers correspond to indices inside every molecule and are the same as the arguments to the `-n` option)

with the numbers in brackets corresponding to nth column of data for each molecule type
- third line: numbering of columns (i.e., column headers)
  - first 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: the range for each molecule type specifies which column numbers correspond to the calculated angles for that particular molecule type and the order of angles is given by the second line
- last three lines: arithmetic means for each calculated angle (last line), headers (second to last line) that again give range of columns in the last line for each molecule type, and calculated angles (third to last line which is the same as the second line at the file beginning)

(2) `-a <file>` – all angles for all molecules in all timesteps
- first and second lines are the same as for `<output>`
- third lines: column headers
  - first is simulation timestep
  - the rest are the calculated data (in degreees): the range for each molecule type corresponds to the number of molecules of the given type times the number of calculated angles; for each molecule the angles are ordered according to the second line

4.11 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 **??** for its format). It calculates number, weight, and z averages. It also calculates distribution functions of aggregation sizes and volumes.

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

$$\langle\mathcal{O}\rangle_{\mathrm{n}} = \frac{\sum_i N_i\mathcal{O}_i}{N}, \quad \langle\mathcal{O}\rangle_{\mathrm{w}} = \frac{\sum_i N_i m_i\mathcal{O}_i}{\sum_i N_i m_i}, \text{ and } \quad \langle\mathcal{O}\rangle_{\mathrm{z}} = \frac{\sum_i N_i m_i^2\mathcal{O}_i}{\sum_i N_i m_i^2}, \tag{4.7}$$

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).

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

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

$$\begin{aligned} F_{\mathrm{n}} &= \frac{N_{A_{\mathrm{S}}}}{\sum_{A_{\mathrm{S}}} N_i} = \frac{N_{A_{\mathrm{S}}}}{N}, \\ F_{\mathrm{w}} &= \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{\sum_{A_{\mathrm{S}}} N_i m_i} = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{\sum_{i=1}^{N} m_i} = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{M}, \text{ and} \\ F_{\mathrm{z}} &= \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}^2}{\sum_{A_{\mathrm{S}}} N_i m_i^2} = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}^2}{\sum_{i=1}^{N} m_i^2}, \end{aligned} \tag{4.8}$$

where $N_{A_{\mathrm{S}}}$ and $m_{A_{\mathrm{S}}}$ 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 normalised so that $\sum F_x(A_{\mathrm{S}}) = 1$.

Distribution of volume fractions of aggregates, $\phi(A_{\mathrm{S}})$, is defined (assuming all beads have the same volume) as

$$\phi(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{\sum_{i=1}^{N} n_i} = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{n}, \tag{4.9}$$

where n_i is the number of beads in aggregate i and n is the total number of beads in all aggregates. If all beads have unit mass (as is often the case in dissipative particle dynamics), the volume distribution, $\phi(A_{\mathrm{S}})$, is equal to the number distribution, $F_{\mathrm{n}}(A_{\mathrm{S}})$. All distributions are written into the `<output distr>` file.

Lastly, `DistrAgg` can calculate number distribution of composition for aggregates with specified size(s) (`-c` option); that is, the distribution of ratios of two molecule types in aggregates with specified size, $\xi = N_{\mathrm{Mol\_A}}/N_{\mathrm{Mol\_B}}$, where $N_{\mathrm{Mol\_A}}$ and $N_{\mathrm{Mol\_B}}$ are the numbers of `Mol_A` and `Mol_B` molecules in the aggregate with size $A_{\mathrm{S}} = N_{\mathrm{Mol\_A}} + N_{\mathrm{Mol\_B}}$. The composition distribution is defined as

$$F_{\mathrm{n}}(\xi) = \frac{N_{\xi,A_{\mathrm{S}}}}{N_{A_{\mathrm{S}}}}, \tag{4.10}$$

where $N_{\xi,A_{\mathrm{S}}}$ is the number of aggregates with aggregate size A_{S} and ratio ξ; $N_{A_{\mathrm{S}}}$ is the total number of aggregates with aggregate size A_{S}. If there are more than two molecule types in the simulation, `-nc` option must be used to specify which molecules should be used in the composition calculation.

The `-nc` option has no effect if used without the `-c`.

The `<avg file>` contains averages for all timesteps regardless of `-st` and `-e` options; that is, the starting step (`-st` option) and ending step (`-e` option) are ignored.

The definition of aggregate size is flexible. If none of `-m`, `-x`, or `--only` options is used, aggregate size is the 'true' aggregation number, A_S, 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. 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 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 aggregate would be 1

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.agg>`	input `agg` file
`<distr file>`	output file with distribution of aggregate sizes
`<avg file>`	output file with per-timestep averages
Non-standard options	
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-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 <output> <int(s)>`	save composition distribution for specified aggregate size(s) to `<output>` file
`-nc <name> <name>`	molecule names to use for composition calculation

Format of output files:

(1) `<output distr>` – distributions of aggregate sizes
- first line: command used to generate the file
- second 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 (4.8))
  - `<volume distribution>` is distribution according to Equation (4.9)
  - next is the total number of aggregates with specified size
  - the remaining columns (`<mol name>_n`) show average numbers of every molecule type in an aggregate with the specified size
- second to last line: column headers for overall averages written in the last line
  - `<M>_n`, `<M>_w`, and `<M>_z` are number, weight, and z averages, respectively, of aggregate masses (the averages are defined in Equation (4.7))
  - other column headers are the same as at the file beginning

(2) `<output avg>` – per-timestep averages
- first line: command used to generate the file
- second lines: column headers
  - first is simulation timestep
  - the rest are for the calculated data: number, weight, and z average aggregate

mass (<M>_n, <M>_w, and <M>_z, respectively) and aggregate size (<As>_n, <As>_w, and <As>_z, respectively) and the last column is the number of aggregates in the given step

- the last two lines are the same as in <output distr>

(3) -c <name> – composition distribution

- first line: command used to generate the file
- second lines: column headers
  - first is ratio of the two molecule types (i.e., 0 to 1)
  - the rest are aggregate sizes
  - in the data, only ratios that are non-zero for at least one aggregate size are written; in case of more than one aggregate size specified by -a option, if the ratio does not exist for some aggregate size(s), '?' is displayed instead of zero

4.12 GenLayers

This utility generates two monolayers composed of molecules specified in a FIELD-like file. Note that only one type of molecule is used; if more complex systems are required, several different layers can be generated separately (allowing for better control of the final complex system) and joined using the AddToSystem utility (Section 4.1). The two layers are mirror images of each other, that is, molecules in both layers are grown from box edge to box middle. The layers are placed in z direction (or on xy planes of the simulation box).

The first beads of the molecules are arranged on a square lattice defined either by given spacing in x and y directions (-s option) or by the number of molecules per layer (-nm option); the default is spacing of 1 in both x and y directions. The rest of the beads of each molecule are placed based on the coordinates in the FIELD-like file. By default, GenLayers places the two mirror layers at the edges of a simulation box; using -g option, a gap from the box edge can be introduced. Therefore, this utility can generate, for example, polymer brushes at box edges or a double layer (such as a biological membrane) in the middle of the box.

By default, the total number of beads in the generated system is equal to three times the box volume, that is, the typical number of beads in dissipative particle dynamics simulations. Beads that are not in the molecules, are put at the beginning of the output files (i.e., they have lower indices) with coordinates of the box centre, and name None. The idea is that once the layers are generated, AddToSystem utility (Section 4.1) can be used to exchange these excess beads for different species (using its --switch option). The -n option changes the total number of beads. If the number is lower than the total number of beads needed to construct the two layers of molecules, their amount is adjusted to exactly that number.

The input FIELD-like file must contain species and molecule sections (al-

though, only the first molecule is considered and `nummols` is disregarded), but the `interaction` section is ignored (see `DL_MESO` manual for details on the `FIELD` file or the `Example/GenLayers` directory for a simple example). The first line of the `FIELD`-like file, must start with box dimensions, i.e., with three numbers.

The utility generates `vsf` structure and `vcf` coordinate files. An example of the usage is shown in `Example/GenLayers` directory.

Usage (`GenLayers` does not use standard options):

```
GenLayers <out.vsf> <out.vcf> <options>
```

Mandatory arguments	
`<out.vsf>`	output `vsf` structure file
`<out.vcf>`	output `vcf` coordinate file
Options	
`-s <x> <y>`	spacing of molecules in x and y directions (default: 1 1)
`-n <int>`	total number of beads (default: three times the box volume)
`-nm <int>`	number of molecules in each layer (rewrites `-s` option)
`-g <float>`	gap between box edges and the molecules (default: 0)
`-f <name>`	`FIELD`-like file (default: `FIELD`)
`-v`	verbose output that provides information about all bead and molecule types
`-h`	print this help and exit

4.13 GenSystem

This utility was not extensively tested and is not a good generator of initial configuration.

This simple utility uses modified `FIELD` file to create `vsf` structure file and to generate coordinates that could be used as an initial configuration. The utility assumes linear chains (no matter the connectivity in the provided `FIELD` file; it looks at exactly the first $n-1$ bonds to get bond lengths, where n is the number of beads in a molecule). For each molecule type, it uses equilibrium bond length (or 0.7 if the bond length is 0) to construct a prototype molecule that is fully stretched in z-direction. The utility then creates layers of molecules that are separated by layers of unbonded beads (if there are any). The utility should fill the whole box with given beads.

The input `FIELD` file must contain `species` and `molecule` sections, but the `interaction` section is ignored (see `DL_MESO` manual for details on the `FIELD` file).

The first line of `FIELD` that is ignored by `DL_MESO` must start with box dimensions, i.e., with three numbers.

This utility does not have error checking for the provided `FIELD` file. If the file is not correct, `GenSystem` will exhibit undefined behaviour, that is, it will either freeze, crash, or run without errors, producing bad output files.

Usage (`GenSystem` does not use standard options):

```
GenSystem <out.vsf> <out.vcf> <options>
```

Mandatory arguments	
`<out.vsf>`	output `vsf` structure file
`<out.vcf>`	output `vcf` coordinate file
Options	
`-f <name>`	FIELD-like file (default: FIELD)
`-v`	verbose output that provides information about all bead and molecule types
`-h`	print help and exit

4.14 GyrationAggregates

This utility calculates the gyration tensor and its eigenvalues (or the roots of the tensor's characteristic polynomial) for all aggregates. Using these eigenvalues, the utility determines shape descriptors: radius of gyration, asphericity, acylindricity, and relative shape anisotropy.

The eigenvalues, λ_x^2, λ_y^2, and λ_z^2, (sorted so that $\lambda_x^2 \leq \lambda_y^2 \leq \lambda_z^2$) are also written to output file(s), because their square roots represent half-axes of an equivalent ellipsoid.

The radius of gyration, R_G, is defined as

$$R_\mathrm{G}^2 = \lambda_x^2 + \lambda_y^2 + \lambda_z^2. \tag{4.11}$$

The asphericity, b, and the acylindricity, c, are defined, respectively, as

$$b = \lambda_z^2 - \frac{1}{2}(\lambda_x^2 + \lambda_y^2) = \frac{3}{2}\lambda_z^2 - \frac{R_\mathrm{G}^2}{2} \quad \text{and} \quad c = \lambda_y^2 - \lambda_x^2. \tag{4.12}$$

The relative shape anisotropy, κ, is defined in terms of the other descriptors as

$$\kappa^2 = \frac{b^2 + 0.75c^2}{R_\mathrm{G}^4} \tag{4.13}$$

Number average of all properties and, additionally, weight and z averages for the radius of gyration are calculated (see Equation (4.7) in Section 4.11 for general definitions of averages). Per-timestep averages (i.e., time evolution) are written to the `<output>` file. Per-size averages can be saved using the `-ps` option.

The gyration tensor is by default calculated for whole aggregates, but `-bt` option can be used to specify which bead types to consider.

Similarly to `DistrAgg`, the definition of aggregate size is flexible – see Section 4.11 for explanations of the `-m`, `-x`, and `--only` options.

The starting step (`-st` option) and ending step (`-e` option) are used only for averages (both overall averages and per-size averages with `-ps` option). Per-timestep averages in the `<output>` file disregard `-st` and `-e` options.

Usage:

```
GyrationAggregates <input> <input.agg> <output> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<input.agg>`	input `agg` file
`<output>`	output file with per-timestep averages
Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for aggregates do not have to be removed)
`-bt <bead name(s)>`	bead type(s) used for calculation
`-ps <name>`	output file with per-size averages
`-m <mol name(s)>`	instead of 'true' aggregate size, use the number of specified molecule type(s) in an aggregate
`-x <mol name(s)>`	exclude aggregates containing only specified molecule type(s)
`--only <mol name(s)>`	use only aggregates composed of specified molecule(s)
`-n <int> <int>`	use only aggregate sizes in given range
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

(1) `<output>` – per-timestep averages
- first line: command used to generate the file
- second line: column headers
  - first is timestep
  - rest are for calculated data: number, weight, and z averages (denoted by `_n`, `_w`, and `_z`, respectively) of radius of gyration and square of radius of

gyration (`Rg` and `Rg^2`, respectively – Equation (4.11)); number averages of relative shape anisotropy (`Anis` – Equation (4.13)), acylindricity, and asphericity (`Acyl` and `Aspher`, respectively – Equation (4.12)), and all three eigenvalues (`eigen.x`, `eigen.y`, and `eigen.z` – λ_x^2, λ_y^2, and λ_z^2)

- second to last line: column headers for overall averages written in the last line
  - `<M>_n` and `<M>_w` are number and weight, respectively, averages of aggregate masses (the averages are defined in Equation (4.7)); aggregate mass here is the mass of all beads of the chosen type(s) in the aggregate
  - average numbers of molecules of each type in an aggregate are shown in columns denoted `<mol names>`
  - the remaining column represent overall averages of the per-timestep quantities described above

(2) `-ps <name>` – per-size averages

- first line: command used to generate the file
- second line: column headers
  - first is aggregate size
  - last column is the total number of aggregates of the given size
  - the columns in between are for the calculated data: simple averages of the numbers of molecules of each type in the aggregate, of radius of gyration and its square, of relative shape anisotropy, acylindricity, and asphericity, and of all three eigenvalues (all are denoted by the above-described symbols)

4.15 GyrationMolecules

This utility calculates the gyration tensor and from it the shape descriptors similarly to `GyrationAggregates` (Section 4.14) but for individual molecules instead of for whole aggregates.

Usage:

```
GyrationMolecules <input> <output> <mol name(s)> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<output>`	output file(s) (one per molecule type) with automatic `-<mol_name>.txt` ending
`<mol name(s)>`	molecule name(s) to calculcate shape descriptors for
Non-standard options	

`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for molecules do not have to be removed)
`-bt <bead name(s)>`	bead type(s) to be used for calculation
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

(1) `<output>` – per-timestep averages (one file per molecule type)
- first line: command used to generate the file
- second line: name of molecule type
- third line: column headers
  - first is timestep
  - rest are for calculated data: number, weight, and z averages (denoted by `_n`, `_w`, and `_z` respectively) of radius of gyration (`Rg` – Equation (4.11)); simple averages of relative shape anisotropy (`Anis` – Equation (4.13)), of acylindricity and asphericity (`Acyl` and `Aspher`, respectively – Equation (4.12)), and of the three eigenvalues (`eigen.x`, `eigen.y`, and `eigen.z` or λ_x^2, λ_y^2, and λ_z^2)
- last three lines show simple averages over the whole simulation
  - third to last line: molecule name
  - second to last line: column headers for radius of gyration and shape descriptors
  - last line: the averages

4.16 Info

This simple utility just prints information about the system read from the provided structure file. If `-c` option is used, only beads in the provided coordinate file are printed. The information is the same as when the verbose option (`-v`) is used with any other utility. Here, the `-v` option prints detailed informations about every bead and every molecule – this information also contains indices as implemented inside the `AnalysisTools` utilities (i.e., not necessarily consistent with the structure file).

Usage (`Info` does not use standard options):

```
Info <input> <options>
```

Mandatory argument	
`<input>`	input structure file (either `.vsf` or `.vtf` format)
Options	
`-c`	input coordinate file in either `.vcf` or `.vtf` format (default: None)

`-h`	print this help and exit

4.17 JoinAggregates

This utility is meant for cases when non-standard option `-j` is omitted from `Aggregates` (or `Aggregates-NotSameBeads`) command. `JoinAggregates` uses the provided `vcf` and `agg` files to join aggregates, i.e., to remove their periodic boundary conditions and save the new coordinates into a `vcf` file. The utility reads `Aggregates` command from the `agg` file to determine distance and number of contact pairs for aggregate check (see Section 4.2 for details on `Aggregates` utility).

Be warned that if `-sk`, `-st`, or `-e` options are used, `<output.vcf>` will not be in sync with `<input.agg>` and, therefore, these two files cannot be used in tandem for further analysis via `AnalysisTools`.

Usage:

```
JoinAggregates <input> <input.agg> <output.vcf> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<input.agg>`	input `agg` file
`<output.vcf>`	output `vcf` coordinate file with indexed coordinates
Non-standard options	
`-sk <int>`	number of steps skip per one used (default: 0)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

4.18 JoinRuns

This utility joins two independent simulation runs of the same system. That is, the two systems contain identical beads and molecules, but these beads and molecules are numbered differently in the `vsf` and `vcf` files from different simulations. The two input `vcf` files must contain the same timestep type (i.e., both indexed or both ordered) and the same number of beads (i.e., if one bead type is absent from one `vcf` file, it must be absent from the second one as well).

The output is a `vcf` coordinate file with beads indexed according to the first `vsf` structure file (i.e., `traject.vsf` or provided by `-i` option). Only specified bead types are saved; using `-r` switch, the reverse is true, i.e., the specified bead types are excluded from the output file.

Usage:

```
JoinRuns <1st input> <2nd input> <2nd vsf> <output> <bead type(s)>
<options>
```

Mandatory arguments	
`<1st input>`	input coordinate file from the first simulation (either `vcf` or `vtf` format)
`<2nd input>`	input coordinate file from the second simulation (either `vcf` or `vtf` format)
`<2nd vsf>`	input structure file from the second simulation (structure file from the first simulation is `traject.vsf`; changeable via `-i` option)
`<output>`	output `vcf` coordinate file with indexed coordinates
`<bead type(s)>`	bead type names to save (can be omitted if `-r` is used)
Non-standard options	
`-r`	reverse function, i.e., exclude `<bead type(s)>` instead of including them; if no `<bead type(s)>` are specified, all bead types are used (requires input coordinate files with all types)
`--join`	join molecules by removing periodic boundary conditions
`-st1 <int>`	starting timestep for first run (default: 1)
`-st2 <int>`	starting timestep for second run (default: 1)
`-e1 <int>`	ending timestep for first run (default: none)
`-e2 <int>`	ending timestep for second run (default: none)
`-sk1 <int>`	number of steps to skip per one used for first run (default: 0)
`-sk2 <int>`	number of steps to skip per one used for second run (default: 0)

4.19 lmp_data

This simple utility generates `data` file for the lammps simulation package (see lammps manual page for details on the `data` file format).

This utility reads system composition from the provided `vsf` structure file, structure of molecules from `DL_MESO FIELD` file (numbers of bond and angle types and angles – other informations are read from the `vsf` file), and coordinates from `vcf` coordinate file. The utility ignores the `interactions` part from `FIELD`. For now, the utility also ignores dihedrals.

Option `--srp` can be used to add one more bead type into the output file (i.e., increment number of atom types and add one more line for `Masses` section with a mass of 1.0). This is intended for cases when the segmental repulsive potential (srp) is used in lammps (srp prevents bond crossing when soft interparticle potential is used in a simulation).

Usage:

```
lmp_data <input> <out.data> <options>
```

Mandatory arguments	
`<input>`	input `vcf` coordinate file
`<out.data>`	output `data` file
Options	
`-f <name>`	FIELD file (default: FIELD)
`--srp`	add one bead type for srp
`-st <int>`	coordinate timestep for creating the `data` file

4.20 lmp_vtf

This utility generates `vsf` and `vcf` files from a lammps `data` file (see lammps manual page for details on its format). Note that while lammps uses the word 'atom', in this manual, 'bead' is used instead. `lmp_vtf` reads the `data` file header (it registers only keywords `atoms`, `bonds`, `angles`, `atom types`, `bond types`, `angle types`, `xlo xhi`, `ylo yho`, and `zlo zhi`) and `Masses`, `Atoms`, `Bonds`, and `Angles` section which must be in this order; everything else (e.g., dihedrals) is ignored.

The utility requires lines in the `Atoms` section to have the following format: `<bead index> <molecule index> <bead type> <charge> <x> <y> <z>` (i.e., `atom_style full` in `lammps` terminology).

If any line of the `Masses` section ends with a comment, its first string is taken as the name of the bead type. Otherwise, the bead types are called `beadN`, where `N` is their type number in the `data` file.

Charge of every bead type is taken as the charge of the last bead of this type in the `Atoms` section.

The utility assumes bonds and angles with simple harmonic potential, u^{b} and u^{a}, respectively:

$$u^{\mathrm{b}} = k_{\mathrm{b}} \left(r - r_0\right)^2 \text{ and } u^{\mathrm{b}} = k_{\mathrm{a}} \left(\theta - \theta_0\right)^2, \tag{4.14}$$

where k_{textb} and k_{a} are strengths of the potentials and r_0 and θ_0 represent equilibrium distance and angle, respectively.

Molecule types are determined according to three criteria: (i) the order of bead types in molecules, (ii) bead connectivity (and bond types), and (iii) angles between bonds (and angle types). Bead order in every molecule is considered from the lowest to the highest bead index in the `data` file. If `<molecule index>` in a bead line is 0, this bead is unbonded (i.e., not part of any molecule). If there's only one bead in a molecule, this bead is considered unbonded as well. The types of molecules differing in bead order are called `molI`, where `I` goes from 1 to the total number of molecule types with different bead order. If two molecules have the same bead order but differ in connectivity (or bond types), `-bJ` is appended to the name (`J` stands for the number of molecule types differing from the original, i.e., it goes from 1 to the number of molecule types differing in connectivity). Similarly, if angles are different, `-aK` is appended with `K` from 1 to the number of molecule types differing in the angles.

Note that lines in both `Atoms` and `Bonds` sections do not have to be ordered in any way.

A relatively complex example of a `data` file and the resulting `vsf` and `vcf` files are in the `Examples/lmp_vtf` directory.

Usage (this utility does not use standard options):

```
lmp_vtf <input> <out.vsf> <out.vcf> <options>
```

Mandatory arguments	
`<input>`	input lammps `data` file
`<out.vsf>`	output `vsf` structure file
`<out.vcf>`	output `vcf` coordinate file
Options	
`-v`	verbose output
`-h`	print this help and exit

4.21 Orientation

This utility calculates orientation order parameter for specified molecules. It determines distribution of the values as well as average values.

Orientation order parameter s_{ij} indicates how stretched molecules in, e.g., membranes are. It is defined as

$$s_{ij} = \frac{1}{2}\left(3\cos\theta - 1\right), \tag{4.15}$$

where

$$\cos\theta = \frac{\mathbf{r_{ij}} \cdot \mathbf{n}}{|\mathbf{r_{ij}}||\mathbf{n}|}. \tag{4.16}$$

That is, the angle θ is between a normal vector to a plane, $\mathbf{n}$, and a vector connecting beads i and j, $\mathbf{r_{ij}}$. The value of s is between -0.5 (parallel to that plane) and 1 (perpendicular to that plane).

The default normal is z-axis, but using `-a` option, a different axis can be used. The `-a` option takes as an argument `x`, `y`, or `z`.

The two beads specifying the second vector can be defined via the `-a` option (similarly to the `-n` option in `AngleMolecules`, Section 4.3). The default is the first and second bead of the molecule, i.e., `-a 1 2`. A multiple of two indices can be provided to calculate different s_{ij}. All provided index pairs are used for all molecule types, unless both indices in a pair are higher than the number of beads in the molecule type. If only one index is higher, then this index is reduced to the highest index for the given molecule. For example, consider two molecule types to be used: `mol1` with four beads and `mol2` with two beads. Should `-a 1 2 1 3 3 4` be specified, `Orientation` would calculate $s_{1,2}$, $s_{1,3}$, and $s_{3,4}$ for `mol1` and $s_{1,2}$ and $s_{1,2}$ for `mol2` (it would literally calculate and print the same result twice). The pair `3 4` would be ignored for `mol2`.

The `<output>` file contains distributions of the orientation order parameter with averages of the parameter appended at the end.

Usage:

```
Orientation <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 the orientation order parameter
`<mol name(s)>`	molecule name(s) used for calculation
Non-standard options	
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-a <axis>`	axis to use as the normal vector (default: z)
`-n <ints>`	bead indices in the molecules (multiple of two; default: 1 2)

Format of output files:

(1) `<output>` – distributions of orientation order parameters, s_{ij}

- first line: command used to generate the file
- next lines: column headers (on line per specified molecule type)
  - first column is the centre of each bin (governed by `width`); i.e., if `<width>` is 0.1, then the centre of the first bin is -0.45 (because $s_{ij} \in \langle 0.5, 1 \rangle$), centre of the second bin -0.35, etc.

  - next, each line shows the bead i and j indices in the specified molecule (according to the provided `vsf` file) to be used for calculation of s_{ij}
- next lines: calculated data
- last line: commented out (i.e., the line starts with `#`) overall averages
- commented out lines right above the last line: column headers for the averages (same as at the beginning of the file)

4.22 PairCorrel

This utility calculates pair correlation function (pcf), or the radial distribution function, between specified bead types. All bead type pairs are used – if `A` and `B` are the specified bead types, `A-A`, `A-B`, and `B-B` bead pairs are used.

The utility does not differentiate beads of the same type that are in different molecules, so a pcf will be a sum of the beads from different molecule types (similar the `DensityAggregates` utility – see Section 4.7).

Usage:

```
PairCorrel <input> <width> <output> <bead name(s)> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	width of each bin of the pair correlation functions
`<output>`	output file with pair correlation functions
`<bead name(s)>`	bead type(s) used for calculation
Non-standard options	
`-n <int>`	number of bins to average to get smoother pair correlation function (default: 1)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

Format of output files:
(1) `<output>` – pair correlation functions between all bead types

- 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: each column corresponds to one bead types pair

4.23 PotentialAggregates

This utility should be working, but it needs more testing.

`PotentialAggregates` calculates electrostatic potential for aggregates of specified size(s) as a function of distance from their centre of mass. It places a virtual particle with charge $q = 1$ at several places on the surface of an ever increasing sphere and calculates electrostatic potential acting on that virtual particle.

At long range, the potential is calculated using Coulomb potential,

$$U_{ij}^{\text{long}} = \frac{l_{\text{B}} q_i q_j}{r_{ij}}, \tag{4.17}$$

where l_{B} is the Bjerrum length, q_i and q_j are charges of particles i and j, and r_{ij} is interparticle distance. At short range, the potential is for now calculated using potential between two charges with exponentially decreasing charge density,

$$U_{ij}^{\text{short}} = U_{ij}^{\text{long}} \left[1 - (1 + \beta r_{ij})\, \mathrm{e}^{-2\beta r_{ij}}\right], \tag{4.18}$$

where $\beta = \frac{5 r_{\text{c}}}{8\lambda}$ (r_{c} is cut-off distance and λ is smearing constant). The utility takes into account periodic images of the simulation box.

For now, parameters for the potential are hard coded in the source code: the Bjerrum length is `bjerrum=1.1` (aqueous solution), cut-off is `r_c=3`, charge smearing constant `lambda=0.2`, and number of periodic images of the simulation box is `images=5`. The parameters can be changed in the source code (around line 20), and the utility can then be recompiled.

The aggregate size can be modified using `-m` option similarly to `Density-Aggregates` (Section 4.7).

Be aware that the utility does not use any Ewald sum-based algorithm, but simple 'brute force' approach, so the calculation is extremely slow (depending on the number of charged particles in the system).

Usage:

```
PotentialAggregates <input> <input.agg> <width> <output> <agg size(s)>
<options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<input.agg>`	input `agg` file
`<width>`	width of each bin of the distribution
`<output>`	output file with electrostatic potential
`<agg size(s)>`	aggregate size(s) for calculation of electrostatic potential

Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for aggregates do not have to be removed)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-m <mol name(s)>`	instead of 'true' aggregate size, use the number of specified molecule type(s) in an aggregate

4.24 SelectedVcf

This utility creates a new `vcf` coordinate file containing only beads of specified types. The selected bead types are printed as comments at the beginning of the output `vcf` file. An `xyz` coordinate file can also be created from the selected bead type(s).

Using `-r` switch, the bead types can also be specified in reverse, i.e., bead types to be excluded from the output file(s).

Which timesteps are saved can be controlled using `-st` option for the first timestep to save and `-e` option for the last timestep to save; using `-sk <int>`, `<int>` steps are ignored per one step saved. If the option to save only specified timesteps (`-n`) is used, `-sk` is ignored. A maximum of 100 steps can be explicitly specified using the `-n` option.

There is also an option to remove periodic boundary conditions (i.e., to join molecules). Conversely, the simualation box can be wrapped (i.e., the periodic boundary conditions applied). If both `--join` and `-w` options are used, the simuation box is first wrapped and then the molecules are joined.

Also, specified molecules can be excluded which is useful when the same bead type is shared between more molecule types. However, no utilities can read a `vcf` file that does not contain all beads of a given type, so the `-x` option is useful only for visualization, e.g., using vmd.

Lastly, the `--last` saves to output file(s) only the last step from the input coordinate file; it is useful, when only the last timestep is required, but the total number of timesteps in the input file is unknown.

Usage:

```
SelectedVcf <input> <output> <bead type(s)> <options>
```

Mandatory arguments

`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<output.vcf>`	output `vcf` coordinate file with indexed coordinates
`<bead type(s)>`	bead type names to save (can be omitted if `-r` is used)
Non-standard options	
`-r`	reverse function, i.e., exclude `<bead type(s)>` instead of including them; if no `<bead type(s)>` are specified, all bead types are used (requires `<input>` with all bead types)
`--join`	join molecules by removing periodic boundary conditions
`-w`	wrap simulation box (i.e., apply periodic boundary conditions)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-sk <int>`	number of steps skip per one used (default: 0)
`-n <int(s)>`	save only specified timesteps
`-x <name(s)>`	exclude molecules of specified name(s) – do not use if `<output.vcf>` is further analysed
`-xyz <name>`	save coordinates to `xyz` file – does not take into account `-x` option
`--last`	save only the last step

4.25 Surface

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

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).

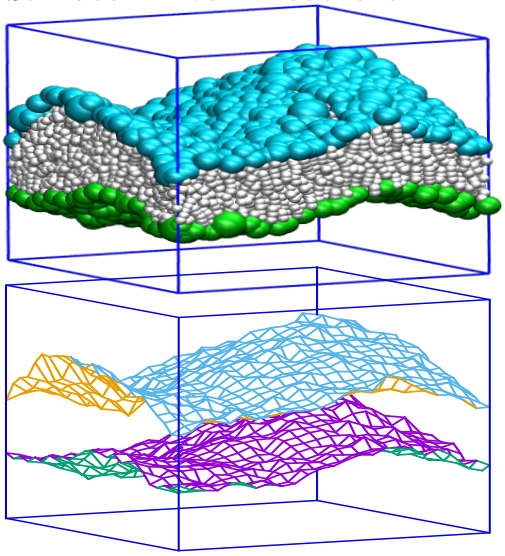
Note that the utility does not determines any structures for now, therefore any molecules outside the brush/membrane can be recognised as a part of the surface (`-bt` and `-m` can be sometimes used to eliminate this problem).

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

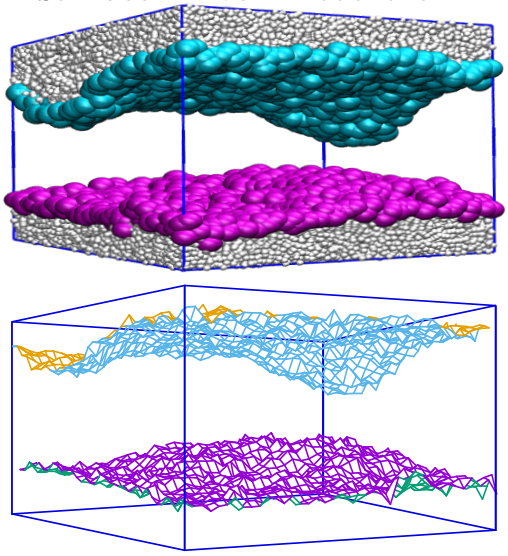
useful when, e.g., other molecules solubilise inside the brush/membrane, leaving some molecules in the solution.

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:

```
Surface <input> <width> <output> <axis> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<width>`	side length of each square
`<output>`	output file
`<axis>`	direction in which to determine the surface: `x`, `y`, or `z`

Non-standard options	
`-in`	start from the box centre instead of from its edges
`-m <name(s)>`	molecule type(s) to use (default: all)
`-bt <name(s)>`	bead type(s) to use (default: all)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)

Format of output files:

(1) `<output>` – 3D coordinates

- first line: command used to generate the file
- second 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

4.26 traject

This utility is from the DL_MESO simulation package and comes in three versions for 2.5, 2.6, and 2.7 versions of DL_MESO. For the latest DL_MESO version, the utility is unmodified and therefore is not included here. The utilities `traject-v2_5` and `traject-v2_6` shipped with DL_MESO 2.5 and 2.6 were modified to generate separate `vsf` and `vcf` file (the `traject.exe` for DL_MESO 2.7 does this natively with `-sc` command line option).

Usage of 2.5 and 2.6 versions (see DL_MESO manual for the description of latest version):

`traject-v2_5 <int>` or `traject-v2_6 <int>`

Mandatory argument	
`<int>`	number of computer cores used for the simulation run (equals the number of `HISTORY` files)

4.27 TransformVsf

This utility reads `FIELD` and `vsf` files to create a new structure file. Generally, this is useful only if `traject-v2_5` or `traject-v2_6` were used to generate the `dl_meso.vsf` file. This file does not contain mass and charge of particles, whereas the one generated with `TransformVsf` (or with `traject.exe` from the dl_meso version 2.7) does and therefore the original `FIELD` file is not needed for further analysis.

Usage (`TransformVsf` does not use standard options):

```
TransformVsf <output.vsf> <options>
```

Mandatory argument	
`<output.vsf>`	output structure file

Options

`-i <name>`	use custom structure file instead of `traject.vsf` (`vsf` or `vtf` format)
`-v`	verbose output
`-h`	print this help and exit

4.28 VisualizeAgg

This utility prints aggregates of given size to output `vcf` files (one file per aggregate size). The output files contain one aggregate per timestep regardless of how many aggregates of the given size are in the initial coordinate file.

The definition of aggregate size is fairly flexible (similarly to `DensityAggregates` utility – Section 4.7).

As the output file(s) do not necessarily contain all beads of a bead type, these `vcf` files cannot be used for further analysis via `AnalysisTools`.

Usage:

```
VisualizeAgg <input> <input.agg> <output> <agg
size(s)> <options>
```

Mandatory arguments	
`<input>`	input coordinate file (either `vcf` or `vtf` format)
`<input.agg>`	input `.agg` file
`<output>`	output file(s) (one per aggregate size) with automatic `#.vcf` ending (`#` is aggregate size)
`<agg size(s)>`	aggregate size(s) to save
Non-standard options	
`--joined`	specify that `<input>` contains joined coordinates (i.e., periodic boundary conditions for aggregates do not have to be removed)
`-st <int>`	starting timestep for calculation (default: 1)
`-e <int>`	ending timestep for calculation (default: none)
`-m <mol name(s)>`	instead of 'true' aggregate size, use the number of specified molecule type(s) in an aggregate
`-x <mol name(s)>`	exclude aggregates containing only specified molecule type(s)

5. Computational details

This chapter will contain some information about how things are coded in the utilities.

5.1 Commonly used structures

Basic system information

`typedef struct Counts {<members>} COUNTS;`

member	explanation
`(int)TypesOfBeads`	number of bead types
`(int)TypesOfMolecules`	number of molecule types
`(int)Beads`*	total number of beads in a coordinate file
`(int)Bonded`*	total number of beads in all molecules
`(int)Unbonded`*	total number of monomeric beads
`(int)BeadsInVsf`*	total number of all beads in the system
`(int)Molecules`*	total number of molecules
`(int)Aggregates`	total number of aggregates
`(int)TypesOfBonds`†	number of bond types
`(int)TypesOfAngles`†	number of bond types

*for `vtf` , this number can correspond to a `vcf` file, i.e., when some beads are missing from the `vcf` file, these beads do not have to be included in this number
†if bond/angle parameters are not known (such as for a `vsf` file), this has a value of `-1`

Information about bead types

`typedef struct BeadType {<members>} BEADTYPE;`

member	explanation
`(char[17])Name`	bead type name (at most 16 characters)
`(int)Number`	number of beads of the given type
`(bool)Use`*	`true`/`false` value whether these beads should be used in a calculation
`(bool)Write`*	`true`/`false` value whether these beads should be written to an output coordinate file
`(double)Charge`	electric charge of the bead type (`1000` if undefined)
`(double)Mass`	mass of the bead type (`0` if undefined)
`(double)Radius`	radius of the spherical bead (`0` if undefined)

* TODO: think about usefulness of multiple `bool`s

- array size: `Counts.TypesOfBeads`

Information about individual beads

`typedef struct Bead {<members>} BEAD;`

member	explanation
`(int)Type`	bead type index corresponding to `struct BeadType` array index
`(int)Molecule`	index of a molecule the bead is in corresponding to `vsf` 's `<resid>-1` value (or to `-1` for monomeric bead)
`(int)nAggregates`*	number of aggregates the bead is in
`(int *)Aggregate`*	1D array with aggregate indices
`(int)Index`	index corresponding to a `vsf` file
`(struct Vector)Position`†	Cartesian coordinates

* TODO: think about usefullness of being in more aggregates & practical considerations
† `struct Vector` contains members `(double)x`, `(double)y`, and `(double)z`

- array size: `Counts.Beads`
- array elements 0 to `Counts.Unbonded` contain monomeric beads
- array elements `Counts.Unbonded+1` to `Counts.Beads` contain bonded beads
- TODO: add `(struct Vector)Velocity` for velocities
- usually accompanied by an `(int *)Index` array (with size of `Counts.BeadsInVsf`) connecting in-code bead indices with `vsf` bead indices, i.e.,
  `Bead[<in-code index>].Index = <vsf index>` and
  `Index[<vsf index>] = <in-code index>`

Information about molecule types

`typedef struct MoleculeType {<members>} MOLECULETYPE;`

member	explanation
`(char[17])Name`*	bead type name (at most 16 characters)
`(int)Number`	number of molecules of the given type
`(int)nBeads`	number of beads in these molecules
`(int *)Bead`	1D array with bead in-code indices (i.e., corresponding to the `struct Bead` array indices)
`(int)nBonds`	number of bonds in these molecules
`(int **)Bond`	2D array with two bead indices of the connected beads and a bond type size of `nBonds`×3: `Bond[i][0]` and 1 hold in-code bead indices for bond i and `Bond[i][2]` holds bond type (or `-1` if bond type undefined)
`(int)nAngles`	number of angles in these molecules
`(int **)Angle`	2D array with three bead indices of the beads in the angle and an angle type size of `nAngles`×4: `Angle[i][0]`, 1, and 2 hold in-code bead indices and `Angle[i][3]` holds angle type (or `-1` if angle type undefined)
`(int)nBTypes`	number of bead types in these molecules

(int *)BType	1D array with bead type indices (i.e., corresponding to a struct BeadType array index)
(bool)InVcf†	true/false value whether these molecules are present in the vcf file
(bool)Use†	true/false value whether these molecules should be used in a calculation
(bool)Write†	true/false value whether these molecules should be written to an output coordinate file
(double)Charge‡	total electric charge of the molecule type
(double)Mass‡	total mass of the molecule type

* TODO: change to char(9) as per vtf specification
† TODO: think about usefulness of multiple bools
‡ undefined if any of the included bead types have undefined charge/mass

- array size: Counts.TypesOfMolecules

Information about individual molecules

typedef struct Molecule {<members>} MOLECULE;

member	explanation
(int)Type	index of molecule type corresponding to a struct MoleculeType array index
(int *)Bead	1D array with in-code bead indices (i.e., corresponding to struct Bead array indices) size: (int)nBeads member of a struct MoleculeType array element
(int)Aggregate	index of an aggregate the molecule is in corresponding to a struct Aggregate array index

- array size: Counts.Molecules
- array indices correspond to <resid>-1 values in a vsf file (because in vsf , residue numbering starts at 1)

Information about individual aggregates

typedef struct Aggregate {<members>} AGGREGATE;

member	explanation
(int)nMolecules	number of molecules in an aggregate
(int *)Molecule	1D array with molecule indices (i.e., corresponding to struct Molecule array indices and to <resid>-1 values in a vsf file)
(int)nBeads	number of bonded beads in an aggregate
(int *)Bead	1D array with in-code bead indices (i.e., corresponding to struct Bead array indices) of bonded beads in an aggregate
(int)nMonomers	number of monomeric beads in an aggregate
(int *)Monomer	1D array with in-code bead indices (i.e., corresponding to struct Bead array indices) of monomeric beads in an aggregate
(double)Mass	total mass of an aggregate (undefined if any of the molecules have undefined mass)
(bool)Use*	true/false value whether this aggregate should be used in a calculation

- array size: `Counts.Aggregates`

5.2 Read system data

In general, information about a system can be read from a `vtf` file(s), `DL_MESO FIELD` file, `LAMMPS data` file, or from a combination of those (such as most of the data from a `vsf` file supplemented by angles and angle parameters and/or bond parameters from a `FIELD` and/or `data` file(s)). Cartesian coordinates can then be read from a `vcf` , `xyz`, or `DL_MESO CONFIG` file.

In principle, not all beads present in the structure file have to be present in the coordinate file.

Subsection 5.2.1 describes reading system data from `vtf` file(s).

5.2.1 Structure from a `vsf` file

First, box size is read from a `vcf` file (if a `vcf` is included in the calculation), then structure of the complete system is read from a `vsf` file, and lastly, the system can be reduced only to beads present in an associated `vcf` file.

Getting simulation box size

```
VECTOR GetPBC(char *vcf_file)
```

variable	input/output?	explanation
`vcf_file`	in	name of the `vcf` file
`GetPBC` returns a `struct Vector` containing x, y, and z side length of the cuboid simulation box		

This function simply reads the `vcf` file until it encounter the `pbc <double> <double> <double>` line.

Getting complete system information

```
void ReadVtfStructure(char *vsf_file, bool detailed, COUNTS *Counts,
BEADTYPE **BeadType, BEAD **Bead, int **Index,
MOLECULETYPE **MoleculeType, MOLECULE **Molecule)
```

variable	input/output?	explanation
`vsf_file`	in	name of the `vsf` file
`detailed`	in	mode for differentiating bead and molecule types
`Counts`	out	broad system information
`BeadType`	out	information about bead types
`Bead`	out	information about individual beads
`Index`	out	connection between in-code bead indices and `vsf` indices
`MoleculeType`	out	information about molecule types
`Molecule`	out	information about individual molecules

The procedure to get all information is as follows:

(1) Go through the `vsf` to get:

- number of a[tom] lines ((int)count_atom_lines)
- number of b[ond] lines ((int)count_bond_lines)
- if present, the line number of the first a[tom] default line ((int)default_atom_line)
- bead names into (char **)atom_name 2D array (size atom_names×17 to save at most 16 characters of each unique name)
- molecule names into (char **)res_name 2D array (size res_names×9 to save at most 8 characters of each unique name)

(2) go through the vsf again to count molecules and save all a[tom] and b[ond] lines into:

- (struct)atom[i] for i-th a[tom] line, i.e., an array of structures (a default line, if present, is the last element of the array), containing members:

(int)index	a[tom] <int> keyword
(int)name	n[ame] <char(16)> keyword; corresponds to an element in the atom_name array holding the actual names
(int)resid	resid <int> keyword
(int)resname	res[name] <char(8)> keyword; corresponds to an element in the res_name array holding the actual names
(double)charge	charge|q <double> keyword
(double)mass	m[ass] <double> keyword
(double)radius	r[adius] <double> keyword

- (int)atom_id array connecting atom[i] elements with vsf bead indices; i.e., atom_id[<vsf id>]=i (so that atom[atom_id[<vsf id>]] structure contains data from i-th a[tom] line)
- (struct)bond[j] for the two connected beads in j-th bond (members (int)index1 and (int)index2 contain vsf bead indices); i.e., an array of structures, so that the two atom[atom_id[bond[j].<member>]] structures contain data from the two connected beads' a[tom] lines

(3) go through the atom array to identify bead types and populate (struct BeadType)bt – two modes

- detailed=true: bead types are distinguished based on their name, mass, charge, and radius – if there's only one value of charge/mass/radius, even beads from a[tom] lines missing that keyword are of the same type; conversely, if there are two values of charge/mass/radius and some a[tom] lines are missing that keyword, three bead types are created with one bead type that has the charge/mass/radius undefined
- detailed=false: bead types are distinguished only according to their names and their charge/mass/radius is set according to the first a[tom] line with that name that's not missing the appropriate keyword

(4) go through the atom array to populate the (struct Bead)bead_all and construct (int)index_all array connecting in-code bead indices and vsf indices, i.e., index_all[<vsf id>]=<in-code id> and thus bead_all[index_all[<vsf id>]].Index=<vsf id>

- internally, unbonded beads are placed before bonded beads, i.e., in all utilities, first `Counts.Unbonded` beads in a `(struct Bead)` array are unbonded and beads in molecules are behind those
- `default` beads are assigned `vsf` indices according to left-out numbers in the `vsf` file

(5) go through the `atom` and `bond` arrays to identify different molecule types and populate the `(struct MoleculeType)mt` – two modes

- `detailed=true`: molecule types are distinguished based on their name, connectivity, numbers and order of beads in the molecule
- `detailed=false`: molecule types are distinguished only according to their names and their connectivity and bead order are determined according to the first molecule with that name encountered in the `vsf` , i.e., if multiple molecules share a name, only the first molecule must have specified bonds

(6) copy all in-function arrays and structure to output arrays and structures

Reducing the system

```
bool CheckVtfTimestep(FILE *vcf, char *vcf_file, COUNTS *Counts,
BEADTYPE **BeadType, BEAD **Bead, int **Index,
MOLECULETYPE **MoleculeType, MOLECULE **Molecule)
```

variable	input/output?	explanation
`vcf`	in	pointer to a `vcf` file open at the beginning of a timestep
`vcf_file`	in	name of the open `vcf` file
`Counts`	out	broad system information
`BeadType`	out	information about bead types
`Bead`	out	information about individual beads
`Index`	out	connection between in-code bead indices and `vsf` indices
`MoleculeType`	out	information about molecule types
`Molecule`	out	information about individual molecules
`CheckVtfTimestep` returns a `true`/`false` value whether the `vcf` contains indexed timesteps		

The procedure to get all information is as follows:

(1) skip timestep preamble (i.e., anything up to the first coordinate line) and determing the timestep type (ordered or indexed)

(2) count coordinate lines and save the bead indices in the timestep as `Bead[i].Flag=true`

- for an ordered timestep, `i` equals line number; for indexed timestep, `i=Index[<vsf index>]` (`<vsf index>` is the first number of the coordinate line)

(3) for an ordered timestep and for an indexed timestep with all beads from the `vsf` file present, do nothing more; otherwise, use the `Bead[i].Flag` values to reduce the system, copying the information about the new system into new structure arrays

- for `struct Counts`, all members except for `BeadsInVsf` are adjusted to contain only numbers relevant to the beads present in the `vcf` file

- for `struct BeadType`, `struct Bead`, `struct MoleculeType`, and `struct Molecule`, only the entities present in the `vcf` file are copied to the new system
- for molecule types, it is possible to remove only part(s) of the molecules (based on the in-molecule bead types); the numbers of bonds and angles are adjusted so that all those containing beads not in the `vcf` file are removed, possibly creating disjointed molecules (or even molecules with no bonds)

(4) copy the new structure arrays back into the original ones, reducing the sizes of those accordingly