# AnalysisTools user manual

Karel Šindelka
k.sindelka@gmail.com

September 5, 2019

# Contents

# 1. Introdution

This software package is a set of utilities aimed to analyse trajectories of coarse-grained simulations. It was initially developed to work with DL_MESO simulation package. However, it works with vtf trajectory files, so it can be used to analyse any `vtf` trajectories.

The emphasis is placed on assembly of molecules, e.i., self-assembly of polymers. It therefore include utilities to determine if molecules are in aggregates and to calculate various properties of aggregates. The utilities only calculate desired quantities and write them to output text files, there is no plotting or visualisation.

Examples of the resulting data can be seen in the author's thesis (Šindelka, Karel: The study of the association behavior of the amphiphilic copolymers in solutions containing low molar compounds by means of computer simulations, dissertation thesis, Charles University, 2018) as well as in papers in impacted journals (e.g., doi:10.1039/C8CP05907A, 10.1134/S1811238217010052, or 10.1007/s00396-017-4090-0).

# 2. Installation

All programs can be compiled using `cmake` which generates `Makefile` and subsequently running `make`. It requires `C` and `FORTRAN` compilers. The compilation should be done in a separate directory, such as `build`.

To create the `Makefule` and compile all utilities, simply run the following command (assuming `build` is a subdirectory of `AnalysisTools` root directory):

```
cmake -G "Unix Makefiles" ../
```

The binaries will be in ‘bin‘ subdirectory of ‘build‘.

Cmake variable `-DCMAKE_BUILD_TYPE=Debug` can be used to compile the version of utilities for debugging.

To compile individual `C` programs using `gcc`, run (assuming you are in a subdirectory of `AnalysisTools` root directory):

`gcc -O3 -lm ../AnalysisTools.c ../Error.c ../Options.c ../Utility/program.c`.

# 3. Format of input/output files

All utilities read information about the system from vsf/vcf files (formatted as described below) and `FIELD` file (input file for DL_MESO simulation package). All system information is read from the `vsf` structure file (Section 3.1) and from the `vcf` coordinate file (Section 3.2). One `vtf` file containing a structure and coordinate sections can also be used. The `FIELD` file is only used when bead charge and/or mass is missing from the `vsf` structure file. The utilities consider only bead types that are present in the `vcf` coordinate file (i.e., bead types present in the `vsf` file but not in the `vcf` file are not seen by the utilities). A description of how the system data are read is shown in Chapter 5.

Both `vsf` and `vcf` files can be generated using a `traject` utility provided by the DL_MESO simulation package (`traject-v2_5` and `traject-v2_6` provided here are modified versions from earlier DL_MESO simulation package versions). Both structure and coordinates can be in one file (with `vtf` extension) – this file can be used instead of separate `vsf` and `vcf` files.

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).

## 3.1 `vsf` structure file

The structure file contains all information about all beads and bonds except for their Cartesian coordinates. The utilities are written for a `vsf` file created by a `traject` utility (for DL_MESO versions from 2.5 to 2.7), but other `vsf` files should work as long as they adhere to the following format.

A `vsf` file is divided into two parts. The first part contains bead definitions. Each line contains the description of a single bead and follows these rules:

- the line starts with `atom` (or just `a`)
- the second string is a bead index number that starts from 0 and increases with every subsequent line (the last bead definition line therefore shows the total number of beads in the simulation)
- the line contains bead name as `name <char(8)>`
- the first bead definition line may contain `default` instead of the index number; every bead that is not explicitly written in the bead definition lines is has the

default name

- if the bead is in a molecule, the line contains molecule name (`resname <char(8)>`) and molecule id (`resid <int>`) that starts from 1
- `mass` and `charge` keywords are read if present (otherwise the mass and charge of beads is read from `FIELD`)
- other keywords are ignored

The following is an example of bead definition lines containing all required data:

```
atom default name Bead_A
atom 0 name Bead_B
atom 3 name Bead_C resname Mol_A 1
atom 4 name Bead_C resname Mol_A 1
atom 6 name Bead_C resname Mol_A 2
atom 7 name Bead_C resname Mol_A 2
atom 8 name Bead_D resname Mol_B 3
atom 9 name Bead_D resname Mol_B 3
```

In this example, there are four bead types (named `Bead_A`, `Bead_B`, and `Bead_C`, and `Bead_D`) and 10 beads in all (with indices from 0 to 9). Beads with indices 1, 2, and 5 are of the default type (`Bead_A`). There are two molecule types named `Mol_A` and `Mol_B` with molecule indices 1 to 3. All molecules with the same name must have the same structure, i.e., the same number of beads and the same bond connectivity.

The second part of a `vsf` file contains bonds definitions and must be preceded by a blank line. Each bond definition line follows these rules:

- the line starts with `bond` (or just `b`)
- bond between two beads is specified by their indices separated by a colon (there cannot be a space between the first number and the colon)

The following is an example of bead definition lines that complement the above-shown bead definition lines:

```
bond 3: 4
# possible comment
bond 6: 7
bond 8: 9
```

In this example, there three bonds between beads with indices 3 and 4, 6 and 7, 8 and 9.

Blank lines and comments (lines beginning with `#`) are allowed in both parts of the `vsf` file.

## 3.2 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 (Section 3.2.1) or indexed timesteps (Section 3.2.2).

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 ordered` or `timestep indexed` (the keyword `timestep` can be omitted). In both ordered and indexed `vcf` files, the size of the simulation box is given by a line `pbc <float> <float> <float>` which is located before the first coordinate block. Only `timestep` and `pbc` lines are read before the first coordinates (everything else is ignored), so `vtf` file can be used instead of a `vcf` file.

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

### 3.2.1 Ordered coordinate file

Coordinate lines in ordered `vcf` file contain only the Cartesian coordinates of the beads in the form `<float> <float> <float>`. The beads are written in ascending order of their indices as defined in the `vsf` file. The following is an example of an ordered `vcf` file:

```
# any number of comments or blank lines

timestep ordered
pbc 10 10 10
0.0 0.0 0.0
0.5 0.5 0.5
...

# comments between timesteps
timestep ordered
# another comment
1.0 1.0 1.0
1.5 1.5 1.5
...
```

In this example, the simulation box is cubic with side length of 10. Beginnings of two timesteps are represented by coordinates of the first two beads with indices 0 and 1 (as defined in the `vsf` file).

### 3.2.2 Indexed coordinate file

Indexed coordinate files contains not only Cartesian coordinates, but also bead indices (preceding the coordinates). Therefore an indexed timestep does not have to contain all beads in the `vsf` structure file. Moreover, the beads do not have to be ordered according to their ascending indices. The following is an example of an ordered `vcf` file:

```
# any number of comments or blank lines

timestep indexed
pbc 10 10 10
 2 0.5 0.5 0.5
21 0.0 0.0 0.0
...

# comments between timesteps
timestep indexed
# another comment
21 1.0 1.0 1.0
 2 1.5 1.5 1.5
...
```

This example is similar to that for the ordored `vcf` file, but two beads have indices 2 and 21 instead of 0 and 1.

## 3.3 Aggregate file (agg)

The aggregate file with `agg` extension is generated using `Aggregates` utility. The file contains information about the number of aggregates in each timestep and which molecules and monomeric beads belong to which aggregate. It serves as an additional input file for utilities that calculate properties of whole aggregates – `agg` file is therefore linked to the `vcf` 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 can be necessary for subsequent analysis of aggregates. The second line is blank and from the third line the data for individual timesteps are shown. Each timestep follows these rules:

- each timestep starts with `Step: <int>` (only `Step` keyword is read by the utilities)
- the second line contains the number of aggregates in the given timestep and 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
  (2) number of monomeric beads in the aggregate followed by their indices taken from the vsf file
- no blank or comment lines are allowed inside the aggregate block
- all molecules present in the vcf file used to generate this file must be present in every timestep; here, aggregate can also refer to dissolved molecules

Following is an example of an agg file:

```
Aggregates in.vcf 1 1 out.agg A

Step: 1
2

2 : 1 3
  3 : 10 100 1000
1 : 2
  1 : 20

Step: 2
1

3 : 1 2 3
  4 : 10 20 100 2000

Last Step: 2
```

In this example, command `Aggregates in.vcf 1 1 out.agg A` was used to generate the file (see Section 4.3 for details about this utility). There are two timesteps here – the first contains two aggregates (although one of them is a free, dissolved molecule) and the second a single aggregate. As an example, the aggregate in the second step contains three molecules with indices 1, 2, and 3 (taken from the vsf file) and four monomeric beads (i.e., solvent or counterions) with indices 10, 20, 100, and 2000 (again, taken from the vsf file).

Besides using this file for further analysis using other utilities, the indices can be used in vmd to visualize, e.g., only a specific aggregate.

# 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 `vsf` file instead of `traject.vsf` |
| `-v` | verbose output that provides information about all bead and molecule types |
| `-V` | detailed verbose output that provides information about all individual molecules as well as about bead and molecule types |
| `-s` | run silently, i.e., without any output at all (overrides `-v` and `-V` options) |
| `--script` | do not rewrite terminal line (useful if output is routed to a file) |
| `-h` | print help and exit |

## 4.1 AddToSystem

This utility takes an existing system specified by `vcf` coordinate and `vsf` structure files and adds new beads into it. The new beads replace neutral unbonded ones with the lowest indices (as ordered in the `vsf` file) from the original system. If molecules are added, `AddToSystem` places them at the end (for the sake of `DL_MESO` which requires molecules to be after unbonded beads). The utility generates `vcf` and `vsf` files for the new system.

`AddToSystem` does not check whether there are enough unbonded neutral beads to be replaced by the new beads; the utility will most likely not show an error, but the resulting `vcf` and `vsf` files won't be right.

Coordinates of the new unbonded beads are ruled by the `-ld`, `-hd`, and `-bt` options (either the first bead or the geometric centre of the new molecules obey these options). The coordinates of the remaining beads in a molecule are governed by the provided coordinates. The molecules are added with a random orientation.

If `-ld` and/or `-hd` options are used, they must accompanied by the `-bt` option.

The structure and number of added molecules and monomeric beads are read

from a FIELD-like file. This file must contain species section followed by molecule section as described in the DL_MESO simulation package.

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 beads).

The molecule section that must be behind 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 these molecules |
| `beads <int>` | Number of beads in these molecules |
| `<bead name> <float> <float> <float>` | One line for each of the `<int>` beads |
| `...` | specifying bead name and its |
| `<bead name> <float> <float> <float>` | Cartesian coordinates |
| `bonds <int>` | Number of bonds in these molecules |
| `<string> <int> <int>` | One line for each of the `<int>` bonds |
| `...` | containing arbitrary string and |
| `<string> <int> <int>` | indices connected beads |
| `...` | Anything beyond here is ignored |
| `finish` | Description of a molecule is finished |

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 arbitrary <string> in the bonds is ignored by AddToSystem (it is a relic from the DL_MESO simulation package, where the <string> specifies a type of bond). The indices in bond lines run from 1 to the number of beads in the molecules and are ordered according to the beads part of the section. Because molecule section in the FIELD file from DL_MESO can also include bond angles and dihedral angles, anything beyond the last bond line is ignored (until the finish keyword is read).

If no molecules are to be added, the line molecule 0 must be still be present in the file.

The following is an example of the FIELD-like file:

```
species 3
```

```
A   1.0  1.0  0
B   1.0  0.0  0
CI  1.0 -1.0 30

molecule 2
Dimer
nummols 10
beads 2
A 0.0 0.0 0.0
A 0.5 0.0 0.0
bonds 1
harm 1 2
finish
surfact
nummols 10
beads 3
A 0.0 0.0 0.0
B 0.5 0.0 0.0
B 1.0 0.0 0.0
bonds 2
harm 1 2
harm 2 3
angles 1
harm 1 2 3
finish
```

In this example, 30 unbonded (or monomeric) negatively charged beads called `CI` are added as well as 20 molecules – 10 molecules called `Dimer` and 10 molecules called `surfact`. `Dimer` molecules contain two `A` beads and one bond each; `surfact` molecules contain three beads and two bonds each. The part starting with `angles` and ending with `finish` is ignored. All in all, 80 beads are added – 30 `CI`, 30 `A`, and 20 `B` beads.

The utility creates the `vcf` and `vsf` files with the new system and can also write the coordinates into a `xyz` file.

Usage:

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

| Mandatory arguments | |
|---|---|
| `<input>` | input coordinate file (either `vcf` or `vtf` format) |

| | |
|---|---|
| `<input add>` | FIELD-like file specifying additions to the system |
| `<out.vcf>` | output `vcf` coordinate file for the new system |
| `<out.vsf>` | output `vsf` structure file for the new system |
| Non-standard options | |
| `-st <int>` | timestep to add new beads to (default: 1) |
| `-xyz <name>` | save coordinates to `xyz` file |
| `-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 |

## 4.2 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 order of beads 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> <mol name(s)> <options>
```

| |
|---|
| Mandatory arguments |

| | |
|---|---|
| `<input>` | input coordinate file (either `vcf` or `vtf` format) |
| `<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) |

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 $n$th 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

## 4.3 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 3.3.

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) |

## 4.4 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 $\tau$. Empty lines and comments (lines beginning with `#`) are skipped. `Average` prints to standard output (i.e., the screen) four numbers: `<n_blocks> <average> <std error> <tau estimate>`:

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

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, $\epsilon$, 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, $\tau$, representing the number of steps between two uncorrelated measurements must be determined. Every $\tau$-th measurement is uncorrelated, so the equation (4.2) can then be used to estimate the error.

A commonly used method to estimate $\tau$ is the binning (or block) method. In this method, the correlated data are divided into $N_\mathrm{B}$ 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:

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

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

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

A way to quickly get $\tau$ estimate is to use a wide range of `<n_blocks>` value and plot the `<tau estimate>` values as a function of `<n_blocks>`. Because the number of data points in one block of the binning analysis should be significantly larger than $\tau$ (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 considere a safe estimate of $\tau$.

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) and it also can calculate 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 4.2). 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) |
| `-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 incomplete `CONFIG`.

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>` | 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`, $\mathrm{d}r$ is the `<width>` argument) divided by the volume of this shell.

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> <output.agg> <width> <output.rho> <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.rho>` | 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) |
| `-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-$\sigma$ 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

## 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 recognise 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.

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) |

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 from `vsf` – similar to `-n` option in `AngleMolecules`, Section 4.2) 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(s) (one per molecule type) with automatic `<mol_name>.rho` ending |
| `<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) |
| `-c <name> <int>` | use specified bead in a molecule `<name>` instead of its centre of mass |

Format of output files:
(1) `<output>` – 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` radial number profile and `stderr` are one-$\sigma$ 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 specified 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 4.2). 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 have to be supplied to the `-n` option.).

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 `<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:

```
DihedralMolecules <input> <mol name(s)> <options>
```

| Mandatory arguments | |
|---|---|
| `<input>` | input coordinate file (either `vcf` or `vtf` format) |
| `<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) |

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 $n$th 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 <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: 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. 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 $A_{\mathrm{S}}$; $M$ 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 the equal to the number distribution, $F_{\mathrm{n}}(A_{\mathrm{S}})$. These distribution are written into the `<output distr>` file.

Lastly, `DistrAgg` can calculate number distribution of composition for aggregates with specified size(s) (`-c` option). This is a number distribution of the numbers of different molecule types in the aggregate. For example, if the simulation box contains molecule types `Mol_A` and `Mol_B`, aggregates with the same size can contain different numbers of these molecules, or different ratios of the numbers of `Mol_A` to `Mol_B` molecules, $\xi = N_{\mathrm{Mol\_A}}/N_{\mathrm{Mol\_B}}$. For now, `DistriAgg` can calculate this distribution only for aggregates containing two molecule types. 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 aggregate with aggregate size $A_{\mathrm{S}}$ and ratio $\xi$; $N_{A_{\mathrm{S}}}$ is the total number of aggregates with aggregate size $A_{\mathrm{S}}$.

The definition of aggregate size is flexible. If none of `-m`, `-x`, `--only` options is used, aggregate size is the 'true' aggregation number, $A_{\mathrm{S}}$, 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 taken as 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 only 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 composed of only Mol_A and Mol_B molecules

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) |
| -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 |

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 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))
  - a column denoted `<mol name>_n` shows an average number of molecules named `mol name` in an aggregate

(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)
- 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 GenSystem

This simple utility uses modified `FIELD` file to create `vsf` structure file and to generate coordinates that could be used as a simulation's starting point. The utility assumes linear chains and uses equilibrium bond length to construct a prototype molecule that is fully stretched in one direction for each molecule type. 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 (the rest of the file is ignored).

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 |

## 4.13 GyrationAggregates

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

The eigenvalues, $\lambda_x^2$, $\lambda_y^2$, and $\lambda_z^2$, (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, $R_\mathrm{G}$, 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 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. To save averages for aggregate sizes, `-ps` option can be used.

The shape descriptors are by default calculated for all beads in the aggregates, but `-bt` option can be used to specify which bead types to use.

Similarly to `DistrAgg`, the definition of aggregate size is flexible – see Section 4.11 for explanation of the `-m` and `-x` 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) |
| `-n <int> <int>` | use only aggregate sizes in given range |
| `-st <int>` | starting timestep for calculation (default: 1) |

(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` – $\lambda_x^2$, $\lambda_y^2$, and $\lambda_z^2$)
- 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, `As`
  - last column is the total number of aggregates of the given size
  - the rest 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, of acylindricity, of asphericity, and of all three eigenvalues (all denoted by the above-described symbols)

## 4.14 GyrationMolecules

This utility calculates shape descriptors similarly to `GyrationAggregates` (Section 4.13), but for individual molecules instead 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) |

(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)); number averages of relative shape anisotropy (`Anis` – Equation (4.13)), acylindricity and asphericity (`Acyl` and `Aspher`, respectively – Equation (4.12))

## 4.15 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 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 4.3 for details on `Aggregates` utility).

The output file is a `vcf` coordinate file.

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** | |
| `-st <int>` | starting timestep for calculation (default: 1) |

## 4.16 JoinRuns

*This utility probably does not work correctly.*

This utility joins two independent simulation runs of the same system. That is, the system contains 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).

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 |

| Non-standard options | |
|---|---|
| `--join` | join molecules by removing periodic boundary conditions |
| `-st1 <int>` | starting timestep first run (default: 1) |
| `-st2 <int>` | starting timestep second run (default: 1) |
| `-sk1 <int>` | number of steps skip per one used for first run (default: 0) |
| `-sk2 <int>` | number of steps skip per one used for second run (default: 0) |

## 4.17 lmp_data

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

The utility reads information on system composition from DL_MESO file (information on all beads and structure of molecules – although it does not read dihedrals) and coordinates from vcf coordinate file. The utility ignores the interactions part from FIELD. The utility also ignore dihedrals.

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) |
| -st <int> | coordinate timestep for creating the data file |

## 4.18 PairCorrel

This utility calculates pair correlation function (pcf) between specified bead types. All bead type pairs are used – if A and B bead types, A-A, A-B, and B-B bead type pairs are used. Right now, the pcfs are not correctly normalised.

The utility do not recognise between beads of the same type that are in different molecules, so a pcf will be a sum of the beads from different molecule types.

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) |

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 correspond to one pair of bead types

## 4.19 PotentialAggregates

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

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

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

$$U_{ij}^{\text{long}} = \frac{l_{\text{B}} q_i q_j}{r_{ij}}, \tag{4.14}$$

where $l_{\text{B}}$ 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,

$$U_{ij}^{\text{short}} = U_{ij}^{\text{long}} \left[1 - (1 + \beta r_{ij})\, \mathrm{e}^{-2\beta r_{ij}}\right], \tag{4.15}$$

where $\beta = \frac{5 r_{\text{c}}}{8\lambda}$ ($r_{\text{c}}$ is cut off distance and $\lambda$ 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 but the utility must then be recompiled.

The aggregate size can be modified using `-m` options similarly to `DensityAggregates` (Section 4.7).

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(s) (one per aggregate size) with automatic `agg#.txt` ending (`#` is aggregate size) |
| `<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) |
| `-m <mol name(s)>` | instead of 'true' aggregate size, use the number of specified molecule type(s) in an aggregate |

## 4.20 SelectedVcf

This utility creates a new `vcf` coordinate file containing only beads of specified types with output `vcf` file. The selected bead types are printed as comments at the beginning of the output `vcf` file.

There is 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, as of now, no utilities can read a `vcf` file that does not contain all beads of a given type, so this can be used only for vmd visualization.

Lastly, `xyz` coordinate file can also be created from the selected bead type(s).

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 |

## 4.21 traject

This utility is from the DL_MESO simulation package and comes in three version 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` 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) |

## 4.22 TransformVsf

This not-very-useful utility just rewrites `vsf` file for better visualization with `vmd`. The output `vsf` file contains not only bead name and index, but its charge and mass

as well.

Usage:

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

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

# 5. Computational details

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

## 5.1 Read system data

`ReadStructure()` function reads all system information from `vsf` and `vcf` files. `FIELD` file is used only to get mass and charge of beads if the information is not in `vsf` file. Provided `vsf` file is used to get all information about beads and molecules – names and numbers of bead and molecules, bonds in molecules. The first timestep of the `vcf` file is used to determine numbers and ids of beads in that `vcf` file which means that all timesteps must contain the same beads.

The procedure in `ReadStructure()` is as follows:

(1) Go through `atom` section of the `vsf` file to identify default bead type (if `atom default` line is present), to find highest bead and molecule ids, and to find bead type names (and charges and masses if present).

(2) Go again through the `atom` section to read names and ids of beads and molecules as well as numbers of all beads and molecules for each type.

(3) Go through `bond` section of the `vsf` file to calculate number of bonds in each molecule type.

(4) Go again through the `bond` section to read bonds for each molecule type.

(5) Go through the `atom` section (for the third time) to assign bead ids to molecules.

(6) Go through the first timestep of `vcf` file to find which beads are in that `vcf` file (if no `vcf` file is provided – e.g., for `DistrAgg` utility – assume all beads are used).

(7) Modify all arrays to accommodate only the beads that are present in the `vcf` file.

(8) Read charge and mass from the `FIELD` file, if not already known from `vsf` file.