Analysis Software Package

Generated by Doxygen 1.8.14

Contents

1 Format of input files for all utilities 1

1.1 Structure file 1

1.2 Indexed coordinate file 2

1.3 Aggregate file 2

2 Options for all utilities 3

3 Common utilities 3

3.1 Aggregates & Aggregates-NotSameBeads utility 3

3.2 Average utility 4

3.3 BondLength utility — not extensively used or tested 5

3.4 Config utility 5

3.5 DensityAggregates 6

3.6 DensityMolecules 7

3.7 DistrAgg utility 7

3.8 GyrationAggregates utility 9

3.9 GyrationMolecules utility 10

3.10 JoinAggregates utility 11

3.11 JoinRuns utility 11

3.12 PairCorrel utility 12

3.13 PairCorrelPerAgg utility 13

3.14 SelectedVcf utility 14

3.15 traject utility 14

3.16 TransformVsf utility 15

3.16.1 Format of output structure file 15

4 Utilities for linear chains 16

4.1 EndToEnd utility 16

4.2 PersistenceLength utility 16

5 Todo List 17

6 Data Structure Index **17**

6.1 Data Structures 17

7 File Index **18**

7.1 File List 18

8 Data Structure Documentation **18**

8.1 Aggregate Struct Reference 18

8.2 Bead Struct Reference 18

8.3 BeadType Struct Reference 19

8.4 Counts Struct Reference 19

8.5 IntVector Struct Reference 20

8.6 LongVector Struct Reference 20

8.7 Molecule Struct Reference 20

8.8 MoleculeType Struct Reference 21

8.9 Vector Struct Reference 21

9 File Documentation **22**

9.1 AnalysisTools.h File Reference 22

9.1.1 Function Documentation 23

9.2 Options.h File Reference 31

9.2.1 Function Documentation 32

9.3 Structs.h File Reference 37

1 Format of input files for all utilities

All the utilities read information about studied system from `vsf/vcf files` (formatted as described below) and `FIELD` file (input file for `DL_MESO simulation package`). Coordinates are read from a `vcf` file with either indexed timesteps. Structure of the system (names and numbers of beads and molecules, etc.) is read from `FIELD` file and `vsf` files, but only bead types that are in the above mentioned `vcf` file are considered.

Aggregate file is of my own format and is used by every utility doing calculation on whole aggregates (as opposed to calculations on individual molecules).

1.1 Structure file

The software package is designed with file `dl_meso.vsf` in mind, which is generated by the `traject` utility provided in DL_MESO software (and modified by me). Generally, the utilities are tested only against files generated by `traject`, but other `vsf` files (such as the one generated by `TransformVsf` utility) should work fine, if formatted according to the following guidelines.

First line may specify default bead (or atom, as called by `vsf`) type, meaning that all beads not specified by their own line are of the default type. No bead of the default type can be in a molecule. If the default line is not present, the number of atom lines must be the same as the total number of beads.

All atom lines in a `vsf` must specify bead index number as `atom <int>` (starting from 0) and name as `name <char[8]>`. If an atom is in a molecule, the name of the molecule type is specified as `segid <char[8]>` and its molecule number (starting from 1) is specified as `resid <int>`. The following is an example of atom lines:

```
atom default 1.0 name <char[8]>
...
atom <int> name <char[8]> segid <char[8]> resid <int>
...
```

While `atom` keyword (or its short version, `a`) must be at the beginning of the line, the order of other keywords do not matter; `radius 1.0` The bead indices must go from the lowest to the highest and the highest index number corresponds to the total number of beads in the systems.

Bond lines follow after atom lines. Only simple bond lines are allowed, i.e. one bond per line in the form: `bond <int>:<int>`.

All molecules with the same name (and different index number) have to be the same, that is, consist of the same number of beads of the same types and have the same structure (i.e., the same bonds).

Blank lines and commented lines (beginning with `#`) are allowed in the `vsf` file.

1.2 Indexed coordinate file

First line of a `vcf` file that is read is a box size line, that is, it contains `pbc <double> <double> <double>` (the three numbers correspond to the side lengths of a cuboid simulation box). Everything up to the `pbc` keyword is ignored. Each timestep starts with a comment line (i.e., line starting with `#`), the second line contains `timestep indexed` (or the short version, `t i`), `c(oordinate) i(ndexed)`, or just `i(ndexed)` and each following line contains index and coordinates of a single bead. Not all beads from the `vsf` file must be present in the `vcf`, instead only selected bead types can be present (although all beads of the selected type(s) must be in all timesteps). Example of indexed coordinate file:

```
pbc <float> <float> <float>
<blank line>
# 1
indexed
<int> <float> <float> <float>
...
```

1.3 Aggregate file

The aggregate file with `.agg` ending is generated using Aggregates utility. It contains information about the number of aggregates in the system in every simulation timestep and therefore is linked to the `vcf` file used to calculate the aggregates. For every aggregate in each timestep there is a number and ids of molecules in that aggregate as well as a number and ids of monomeric beads near the aggregate.

The first line of an aggregate file contains the command used to generate it. The subsequent lines contain information on individual timesteps starting with `Step` keyword, followed by the number of aggregates in the timestep and followed by individual aggregates. Every aggregate is spread over two lines - the first one contains the number of molecules in the aggregate followed by their ids (according to a corresponding `vsf` structure file) and the second line contains the number of monomeric beads in the aggregate followed by its ids (again, the ids correspond to the `vsf` file). The line with monomeric beads is indented for easier reading. The file ends with `Last step: <number of timesteps>`; the `L` marks the end of `agg` file for any utility using it.

Example of an aggregate file:

```
<command used to generate it>
<blank line>
Step: 1
<number of aggregates in step 1>
<blank line>
2 : 1 34
3 : 230 40000 41003
<number of molecules in the second aggregate> : <molecule ids>
<number of monomeric beads near the aggregate : <bead ids>
<blank line>
Step: 2
...
<blank line>
Last Step: <number>
```

2 Options for all utilities

Most of the utilities have several options that are the same. These options are described here, but can be used with any utility unless stated otherwise. These options must be specified after any mandatory arguments.

`-i <name>`
: use custom `.vsf` structure file instead of the default `dl_meso.vsf` (must end with `.vsf`)

`-v`
: verbose output providing information about the system

`-V`
: more detailed verbose output

`-s`
: run silently, that is, without any output at all (overrides verbose options)

`--script`
: do not rewrite terminal line (useful if output is routed to file)

`-h`
: print help and exit

3 Common utilities

Utilies that are not specific to any given system and are used for all simulations.

3.1 Aggregates & Aggregates-NotSameBeads utility

These utilities determine which molecules belong to which aggregates 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 certain distance. Both the distance and the number of needed contact pairs are arguments of the command as well as bead types to consider. Specified molecule(s) can be excluded from aggregate calculation (both from aggregate calculation and the output `.agg` file).

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 bead type pairs, that is `A-A`, `A-B` and `B-B` (provided the beads are not in the same molecules), but Aggregates-NotSameBeads will not use `A-A` or `B-B`. Therefore, at least two bead types must be provided in Aggregates-NotSameBeads argument.

Usage:

`Aggregates` (or `Aggregates-NotSameBeads`) `<input.vcf> <distance> <contacts> <output.agg> <type names> <options>`

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<distance>`

minimum distance for two beads to be in contact (constituting one contact pair)

`<contacts>`

minimum number of contact pairs to consider two molecules to be in one aggregate

`<output.agg>`

output filename (must end with `.agg`) containing information about aggregates

`<type names>`

names of bead types to use for calculating contact pairs

`<options>`

`-x <name(s)>`

exclude specified molecule(s) from calculation of aggregates

`-j <joined.vcf>`

filename for coordinates of joined aggregates (must end with `.vcf`)

3.2 Average utility

Average uses binning method to analyse data stored in a supplied file. It prints average, statistical error and estimate of integrated autocorrelation time (tau). Empty lines and lines beginning with '#' are skipped. The program prints to the screen four numbers: `<n_blocks> <simple average> <statistical error> <estimate of tau>`.

A way to estimate a 'real' value of tau is to use a wide range of `<n_blocks>` and then plot `<tau>` as a function of `<n_blocks>`. Since the number of data points in a block has to be larger than tau (say, 10 times larger), plotting `<number of data lines>/10/<n_blocks>` vs. `<n_blocks>` will produce an exponential function that will intersect the plotted `<tau>`. A value of tau near the intersection (but to the left where the exponential is above `<tau>`) can be considered a good estimate for tau.

Usage:

```
Average <filename> <column> <discard> <n_blocks>
```

`<filename>`

name of data filel

`<column>`

column number in the file containing the data to analyze

`<discard>`

number of data values considered as equilibrium

`<n_blocks>`

number of blocks for binning analysis

3.3 BondLength utility — not extensively used or tested

BondLength utility calculates normalised distribution of bond length for specified molecule types.

Usage:

```
BondLength <input.vcf> <width> <output file> <molecule names> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<width>`

width of each bin for the distribution

`<output file>`

output filename containing distribution of bond lengths

`<molecule names>`

names of molecule types to calculate the distribution for

3.4 Config utility

This utility takes `.vcf` file containing all beads (such as `All.vcf` created by DL_MESO `traject` utility which was modified by me) and creates `CONFIG` file (file containing initial coordinates for a simulation via `DL_MESO simulation package`). If a `vcf` file that does not contain all beads, Config will still run, but the generated file will not contain coordinates for all beads and thus will not be able to be used to start a simulation rune using DL_MESO software.

Usage:

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

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<options>`

`-st <int>`

timestep for creating the CONFIG file (if the number is higher than the number of steps, the last step is used)

xxx

3.5 DensityAggregates

This utility calculates number bead density for aggregates of specified size from their centre of mass. During the calculation, only the current aggregate is taken into account, so there is no possibility of getting 'false' densities from adjacent aggregates. Therefore if some bead type is never present in an aggregate of specified size (but is in the `.vcf` file), its density will always be 0.

Instead of true aggregate size, a number of molecules of specified name can be used, i.e. an aggregate with 1 `A` molecule and 2 `B` molecules can be specified with `<agg sizes>` of 3 without `-m` option or 1 if `-m A` is used (or 2 if `-m B` is used).

Also specified molecule type(s) can be excluded via the `-x` option. This is useful in case of several molecules sharing the same bead type. Calculated densities take into account only name of a bead type, not in which molecule(s) it occurs. The density from the bead type in different molecule types will therefore be the sum of the densities from those molecules.

Usage:

```
DensityAggregates <input.vcf> <input.agg> <width> <output.rho> <agg sizes>
<options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<input.agg>`

input filename (must end with `.agg`) containing information about aggregates

`<width>`

width of each bin for the distribution

`<output.rho>`

output density file (automatic ending `agg#.rho` added)

`<agg sizes>`

aggregate sizes for density calculation

`<options>`

`--joined`
specify that the `<input.vcf>` contains aggregates with joined coordinates

`-n <int>`
number of bins to average

`-st <int>`
starting timestep for calculation

`-m <molecule type name>`
instead of aggregate size, use number of molecules of specified molecule types

`-x <name(s)>`
exclude specified molecule(s)

Todo DensityAggregates: check if only chains in one aggregate are used – anomalies in VanDerBurgh/AddedPol/

JoinRuns: implement wholy `--script` common option

Completely change this - either implement `-x` option or remove function `WriteCoorIndexed` and hard code the writing to file

3.6 DensityMolecules

DensityMolecules works in similar way as the DensityAggregates, only instead of aggregates, the densities are calculated for specified molecule types. Care must be taken with beadtype names in various molecules types, because if one beadtype appears in more molecule types, the resulting density for that beadtype will be averaged without regard for the various types of molecule it appears in.

It is possible to use specified bead instead of the centre of mass for the coordinates to calculate densities from. Care must be taken, because the order of molecule types is taken from `FIELD` rather then from `DensityMolcules` arguments. For example: whether bead 1 will be connected with `NameA` or `NameB` in `DensityMolecules ... NameA NameB -c 1 2` depends on molecules' order in `FIELD` file; that is if `NameA` is first in `FIELD`, 1 will be associated with `NameA` and 2 with `NameB`, but if `NameB` is first, the associations are reverse, regardless of the order of names in the command's arguments. If the centre of mass should be used, `x` is given as argument. In the above example (assuming `NameA` is first in `FIELD`) if bead 1 is intended to be used for `NameB`, but centre of mass for `NameA`, then an argument of the form `-c x 1` must be used.

Usage:

```
DensityMolecules <input.vcf> <width> <output.rho> <mol name(s)> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<width>`

width of each bin for the distribution

`<output.rho>`

output density file (automatic ending `agg#.rho` added)

`<mol name(s)>`

names of molecule types to calculate density for

`<options>`

`--joined`

specify that the `<input.vcf>` contains aggregates with joined coordinates

`-n <int>`

number of bins to average

`-c x/<int>`

use specified molecule bead instead of centre of mass

3.7 DistrAgg utility

DistrAgg calculates number and weight average aggregation masses for each timestep (i.e. their time evolution) as well as their average over the whole simulation. The number average and weight average aggregation mass, $\langle M\rangle_{\mathrm{n}}$ and $\langle M\rangle_{\mathrm{w}}$ respectively, are defined as:

$$\langle M\rangle_{\mathrm{n}} = \frac{\sum_{A_{\mathrm{S}}} m_i N_i}{N}\,, \quad \text{and} \quad \langle M\rangle_{\mathrm{w}} = \frac{\sum_{A_{\mathrm{S}}} m_i^2 N_i}{\sum_{A_{\mathrm{S}}} m_i N_i}, \tag{1}$$

where the sums go over all aggregate sizes. m_i is the mass of an aggregate with aggregation number $A_{\mathrm{S}} = i$, N_i is the number of such aggregates and N is total number of aggregates. The equations can be written more conveniently (for programming purposes) as sums over aggregates themselves instead of their sizes:

$$\langle M\rangle_{\mathrm{n}} = \frac{\sum_{i=1}^{N} m_i}{N}\,, \quad \text{and} \quad \langle M\rangle_{\mathrm{w}} = \frac{\sum_{i=1}^{N} m_i^2}{\sum_{i=1}^{N} m_i}. \tag{2}$$

The utility also calculates number, weight and z distribution function of aggregation numbers. The distributions, $F_{\mathrm{n}}(A_{\mathrm{S}})$, $F_{\mathrm{w}}(A_{\mathrm{S}})$, and $F_{\mathrm{z}}(A_{\mathrm{S}})$ respecively, are defined as:

$$F_{\mathrm{n}}(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}}}{\sum_{A_{\mathrm{S}}} N_i} = \frac{N_{A_{\mathrm{S}}}}{N}\,, \quad F_{\mathrm{w}}(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{\sum_{A_{\mathrm{S}}} N_i m_i}\,, \quad \text{and} \quad F_{\mathrm{z}}(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}^2}{\sum_{A_{\mathrm{S}}} N_i m_i^2}, \tag{3}$$

where $N_{A_{\mathrm{S}}}$ and $m_{A_{\mathrm{S}}}$ stand for the number and mass, respectively, of aggregates with aggregation number A_{S}. The equations are normalized so that $\sum F_x(A_{\mathrm{S}}) = 1$. Equations for F_{w} and F_{z} can again be transformed to contain sums over aggregates, not their sizes:

$$F_{\mathrm{w}}(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}}{\sum_{i=1}^{N} m_i} \quad \text{and} \quad F_{\mathrm{z}}(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} m_{A_{\mathrm{S}}}^2}{\sum_{i=1}^{N} m_i^2}, \tag{4}$$

Lastly, the utility calculates distribution of volume fractions of aggregates under the assumption that all beads have the same volume, which is essential in dissipative particle dynamics simulation method. Since in DPD the volumes of all beads are identical, volume fraction of an aggregate with aggregation number A_{s} is calculated as:

$$\phi(A_{\mathrm{S}}) = \frac{N_{A_{\mathrm{S}}} n_{A_{\mathrm{S}}}}{\sum_{i=1}^{N} n_i}, \tag{5}$$

where n_i is the number of beads in an aggregate with $A_{\mathrm{s}} = i$.

The definition of aggregation number , A_S, is somewhat flexible. The `-m <name>` option can be used to specify that the aggregation number is not the number of all molecules in an aggregate, but rather only the number molecules of the specified type(s). Two values are then taken as the mass and aggregation number for calculations of distributions and averages – the 'true' value (noted as `whole agg mass` in output files) and the '`-m` option value' (noted as `options mass`). For example, let's assume an aggregate contains `A` and `B` molecules. Using `-m A` will count the aggregate size as the number of `A`s, but its mass as both the mass of only `A` molecules and as the sum of `A` and `B` molecules' masses. The resulting distributions will therefore be the functions of aggregation number specified by the `-m` option, but for every distribution function, there will be two data columns in the output file (and two sets of overall averages).

Also using the `--only <name>` uses only aggregates composed exclusively of a specified molecule type. On the contrary, `-x <name(s)>` option discounts aggregates containing only the specified molecules.

Lastly, the calculations can be made only for a given range of aggregation numbers (specified by the `-m` option if present) if `-n` option is used.

The utility reads information about aggregate from input file with Aggregate format. This file can be generated using Aggregates utility.

Usage:

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

`<input>`

  input filename with information about aggregates

`<distr file>`

  output filename with weight and number distribution functions

`<avg file>`

  output filename with weight and number average aggregation number in each timestep

`<options>`

- `-st <int>`

  starting timestep for calculation (does not affect calculation of time evolution)

- `-n <int> <int>`

  range of aggregation numbers to calculate distributions and averages for

- `-m <name(s)>`

  instead of aggregate size, use number of molecules of specified molecule types

- `--only <molecule type name>`

  use only aggregates composed of specified molecule type

- `-x <name(s)>`

  exclude aggregates containing only specified molecule(s)

3.8 GyrationAggregates utility

This utility calculates a gyration tensor and its eigenvalues (as roots of the tensor's characteristic polynomial) for all aggregates. Using the eigenvalues, various shape descriptors are determined.

It calculates radius of gyration, R_{G}, as:

$$R_{\mathrm{G}}^2 = \lambda_x^2 + \lambda_y^2 + \lambda_z^2, \tag{6}$$

where λ_i^2 is the i-th eigenvalue of the gyration tensor (or its i-th principle moment). The eigenvalues are sorted so that $\lambda_x^2 \le \lambda_y^2 \le \lambda_z^2$. Then it calculates the asphericity, b:

$$b = \lambda_z^2 - \frac{1}{2}\left(\lambda_x^2 + \lambda_y^2\right) = \frac{3}{2}\lambda_z^2 - \frac{R_{\mathrm{G}}^2}{2}, \tag{7}$$

the acylindricity, c:

$$c = \lambda_y^2 - \lambda_x^2 \tag{8}$$

and the relative shape anisotropy , κ:

$$\kappa^2 = \frac{b^2 + 0.75c^2}{R_{\mathrm{G}}^4} = \frac{3}{2}\frac{\lambda_x^4 + \lambda_y^4 + \lambda_z^4}{\left(\lambda_x^2 + \lambda_y^2 + \lambda_z^2\right)^2} \tag{9}$$

Number averages of all the properties and weight and z averages for radius of gyration are calculated. The number average for a quantity $\mathcal{O}$ is defined as:

$$\langle \mathcal{O} \rangle_{\mathrm{n}} = \frac{\sum_i \mathcal{O}_i}{N}, \tag{10}$$

where N is the total number of aggregates. The weight average is:

$$\langle \mathcal{O} \rangle_{\mathrm{w}} = \frac{\sum_i m_i \mathcal{O}_i}{\sum_i m_i}, \tag{11}$$

where m_i is mass of an aggregate i. Lastly, the z-average is:

$$\langle \mathcal{O} \rangle_{\mathrm{z}} = \frac{\sum_i m_i^2 \mathcal{O}_i}{\sum_i m_i^2}. \tag{12}$$

All the sums go over individual aggregates. Both number and weight average according to these equations were already used in the DistrAgg utility.

Averages of the shape descriptors during the simulation (or their time evolution) are written to an output file and overall averages are appended to that file.

Sums dependent on aggregation number can be written to output file with `-ps <name>` option. The aggregation number can be either the number of all molecules in an aggregate or a number of molecules of the type specified by `-m <name>` option.

Usage:

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

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<input.agg>`

input filename (must end with `.agg`) containing information about aggregates

`<output.vcf>`

output filename with shape descriptors for chosen sizes throughout simulation

`<agg sizes>`

aggregate sizes for gyration calculation

`<options>`

- `--joined`

  specify that the `<input.vcf>` contains aggregates with joined coordinates

- `-bt`

  specify bead types to be used for calculation (default is all)

- `-m <name>`

  take as an aggregate size the number of `<name>` molecules in aggregates instead of the number of all molecules

- `-ps <name>`

  output filename with per-size (or per-aggregation-number) averages

- `-n <int> <int>`

  range of aggregation numbers to calculate distributions and averages

3.9 GyrationMolecules utility

This utility function in the same way as GyrationAggregates, but it calculates radii of gyration for specified molecule types instead of aggregate sizes.

Right now it calculates gyration for all beads in the specified molecule types.

Usage:

```
GyrationMolecules <input.vcf> <input.agg> <output> <molecule names> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<input.agg>`

input filename (must end with `.agg`) containing information about aggregates

`<output.vcf>`

output filename with radii of gyration throughout simulation (automatic ending #.txt)

`<molecule names>`

molecule types for gyration calculation

`<options>`

- `-bt`

  specify bead types to be used for calculation (default is all)

- `--joined`

  specify that the `<input.vcf>` contains joined coordinates

3.10 JoinAggregates utility

This utility reads input `.vcf` and `.agg` files and removes periodic boundary conditions from aggregates - e.i. it joins the aggregates. The distance and the bead types for closeness check are read from the first line of `.agg` file with contains full `Aggregates` command used to generate the file. JoinAggregates is meant for cases, where `-j` flag was omitted in Aggregates utility.

Usage:

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

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<input.agg>`

input filename (must end with `.agg`) containing information about aggregates

`<output.vcf>`

output filename (must end with `.vcf`) with joined coordinates

3.11 JoinRuns utility

MOST LIKELY NOT WORKING – IT'S NOT USED.

This program is to be used if two simulation runs with different initial seeds (that is, two simulations with different bead id numbers `.vsf` files, but identical `FIELD` files) should be joined. Two `.vcf` files that contain the same bead types must be provided as well as the `.vsf` structure file for the second simulation. The output `.vcf` coordinate files has bead ids according to the structure file of the first simulation. The program is, however, extremely inefficient with unbonded beads, while bonded beads are always sorted in the same way by DL_MESO simulation software. The usefullness of such utility is confined to cases with more then one type of unbonded beads and under those conditions the utility may take around 1 minute per step (of the second simulation run) for system in box of side length 40.

Usage:

```
JoinRuns <1st input.vcf> <2nd input.vcf> <2nd input.vsf> <output.vcf>
<type names> <options>
```

`<1st input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps for the first simulation

`<2nd input.vcf>`

input coordinate filename for the second sumation in the same format as the first coordinate file

`<2nd input.vsf>`

`.vsf` structure file for the second simulation (must end with `.vsf`)

`<output.vcf>`

output filename with indexed coordinates (must end with `.vcf`)

`<type names>`

names of bead types to save

`<options>`

`--joined`

join individual molecules by removing periodic boundary conditions

`-n1 <int>`

number of timestep to start the first simulation from

`-n2 <int>`

number of timestep to start the second simulation from

`-u1 <int>`

leave out every `skip` steps in the first simulation

`-u2 <int>`

leave out every `skip` steps in the second simulation

Todo JoinRuns: base reindexing of beads in the second simulation on comparison between the two `.vsf` files

3.12 PairCorrel utility

This utility calculates pair correlation function (pcf) between specified bead types. All pairs of bead types (including same pair) are calculated - given `A` and `B` types, pcf between `A-A`, `A-B` and `B-B` are calculated.

Currently, the utility cannot recognise between beads of the same type that are in different molecule types - i.e. if bead type `A` is both in molecule type `1` and molecule type `2`, only one pcf will be calculated regardless of the molecule type `A` is in.

Usage:

```
PairCorrel <input.vcf> <width> <output.pcf> <bead type(s)> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<width>`

width of each bin for the distribution

`<output.pcf>`

output file with pair correlation function(s)

`<bead type(s)>`

bead type name(s) for pcf calculation

`<options>`

`-n <int>`

number of bins to average

`-st <int>`

starting timestep for calculation

3.13 PairCorrelPerAgg utility

DO NOT USE.

What would this be for? Per aggregate densities are calculated via DensityAggregates utility.

DISREGARD THE REST OF THIS ENTRY.

PairCorrelPerAgg utility calculates pair correlation function per aggregates - that is only beads in the same aggregate are used. If aggregate size(s) is not specified, average pcf is calculated (that is, regardless of aggregate size). In all probability the utility is working, but since it is not really useful, it has never been thouroughly tested.

Usage:

`PairCorrelPerAgg <input.vcf> <input.agg> <width> <output.pcf> <bead type(s)> <options>`

`<input.vcf>`
: input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<input.agg>`
: input filename (must end with `.agg`) containing information about

`<width>`
: width of each bin for the distribution

`<output.pcf>`
: output file with pair correlation function(s)

`<bead type(s)>`
: bead type name(s) for pcf calculation

`<options>`
: `-n <int>`
  number of bins to average
: `-st <int>`
  starting timestep for calculation

3.14 SelectedVcf utility

This utility takes `.vcf` file containing either ordered timesteps or indexed timesteps and creates a new `.vcf` coordinate file containing only beads of selected types with an option of removing periodic boundary condition and thus joining molecules. The otput `.vcf` file contains indexed timesteps.

Specified molecules can be excluded which is useful when the same bead type is shared between more molecule types.

The selected `<type names>` are printed at the beginning of the output file.

Usage:

`SelectedVcf <input.vcf> <output.vcf> <type names> <options>`

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps

`<output.vcf>`

output filename with indexed coordinates (must end with `.vcf`)

`<type names>`

names of bead types to save

`<options>`

`--join`

join individual molecules by removing periodic boundary conditions

`-st <int>`

starting timestep for calculation

`-sk <int>`

number of steps to skip per one used

`-x <name(s)>`

exclude specified molecule(s)

3.15 traject utility

This utility is from the `DL_MESO simulation package`. While originally it creates a `.vtf` file containing both structure and coordinates, I have changed it to create a separate `dl_meso.vsf` structure file and `All.vcf` coordinate file containing ordered timesteps.

There are two versions from two versions of the `DL_MESO simulation package`, namely versions 2.5 and 2.6.

Usage:

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

`<cores>`

number of computer cores used for the simulation run (or the number of `HISTORY` file)

The standard options cannot be used with this utility.> }}}

3.16 TransformVsf utility

This utility takes `.vsf` structure file and DL_MESO input file `FIELD` and transforms them into a different `.vsf` structure file that is better suited for visualisation using VMD software.

Usage:

`TransformVsf <output.vsf> <options>`

`<output.vsf>`

output structure file that must end with `.vsf`

3.16.1 Format of output structure file

Every atom line in the generated structure file contains bead's index number, mass, charge and name. Atom lines for beads in molecules also contain molecule's id number and the name of the type of molecule. The bond section of `output.vsf` lists all bonds one by one (i.e. no chains of bonds in the format `<id1>:: <id2>` are used). Information about which bonds belong to which molecule is provided as a comment. The file has the following format:

```
atom default name <name> mass <m> charge <q>

...

atom <id> name <name> mass <m> charge <q>

...

atom <id> name <name> mass <m> charge <q> segid <name> resid
<id>

...

# resid <id>

<bonded bead id1>: <bonded bead id2>

...
```

For VMD atom selection:

`segid <name>`

selects all molecules with given name(s)

`resid <id>`

selects molecule(s) with given index number(s)

`charge <q>`

selects all beads with given charge(s) (double quotes are required for negative charge)

`mass <m>`

selects all beads with given mass(es)

4 Utilities for linear chains

This section provides information about utilities with calculations that are sensible to do only on linear polymer chains. No check whether the molecules are linear is done.

4.1 EndToEnd utility

This utility calculates end-to-end distance of specified molecules. End-to-end distance makes sense only for linear chains, therefore it is assumed that the provided molecule names are linear chains. No check is performed. The distance is calculated between the first and the last bead of the molecule; that is, between the first and the last bead in the `FIELD` entry for the given molecule. Also the use of joined coordinates (that is, without periodic boundary condition) is required, because the utility does not remove periodic boundary conditions.

The output is a file containing average end-to-end distance for every molecule type for each timestep.

Usage:

```
EndToEnd <input.vcf> <output.vcf> <molecule names> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps (with joined coordinates)

`<output.vcf>`

output filename with indexed coordinates (must end with `.vcf`)

`<molecule names>`

names of molecule types (linear chains) to use

4.2 PersistenceLength utility

WARNING! Probably does not work as it should. Although I never needed it, some preliminary tests produced negative values, which is plain wrong.

This utility calculates persistence length of specified molecules. It is assumed that the provided molecules are linear chains, but no check is performed. Also the use of joined coordinates (that is, without periodic boundary condition) is required, because the utility does not remove periodic boundary conditions.

The calculation of the persistence length is based on the projection of angles between bonds vectors (see e.g. `this paper`). The following formula for the persistence length, l_P is used:

$$l_\mathrm{P} = \langle b \rangle \sum_{i=0}^{i=N_b} \langle \cos\theta_i \rangle, \tag{13}$$

where $\langle b \rangle$ is the average bond length in a molecule, $\langle \theta_i \rangle$ is the average angle between two bond vectors separated by i bonds. N_b is the number of bonds in the given molecule.

The output is a file containing average persistence length for every molecule type for each timestep.

Usage:

```
PersistenceLength <input.vcf> <output> <molecule names> <options>
```

`<input.vcf>`

input coordinate filename (must end with `.vcf`) containing either ordered or indexed timesteps (with joined coordinates)

`<output.vcf>`

output filename with indexed coordinates (must end with `.vcf`)

`<molecule names>`

names of molecule types (linear chains) to use

5 Todo List

Page Common utilities

DensityAggregates: check if only chains in one aggregate are used – anomalies in VanDerBurgh/AddedPol/

JoinRuns: implement wholy `--script` common option

Completely change this - either implement `-x` option or remove function `WriteCoorIndexed` and hard code the writing to file

JoinRuns: base reindexing of beads in the second simulation on comparison between the two `.vsf` files

6 Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

Aggregate
Information about every aggregate 18

Bead
Information about every bead 18

BeadType
Information about bead types 19

Counts
Total numbers of various things 19

IntVector
3D vector of integers 20

LongVector
3D vector of floats 20

Molecule
Information about every molecule 20

MoleculeType
Information about molecule types 21

Vector
3D vector of floats 21

7 File Index

7.1 File List

Here is a list of all documented files with brief descriptions:

AnalysisTools.h
Functions common to all analysis utilities 22

Options.h
Options usable in utilities **31**

Structs.h
Structures for utilities **37**

Common/Aggregates.h **??**

8 Data Structure Documentation

8.1 Aggregate Struct Reference

Information about every aggregate.

```
#include <Structs.h>
```

Data Fields

- int nMolecules

  number of molecules in aggregate
- int ∗ Molecule

  ids of molecules in aggregate
- int nBeads

  number of bonded beads in aggregate
- int ∗ Bead

  ids of bonded beads in aggregate
- int nMonomers

  number of monomeric beads in aggregate
- int ∗ Monomer

  ids of monomeric beads in aggregate
- double Mass

  total mass of the aggregate
- bool Use

  should aggregate be used for calculation?

8.2 Bead Struct Reference

Information about every bead.

```
#include <Structs.h>
```

Data Fields

- int Type

  type of bead corresponding to index in BeadType struct
- int Molecule

  index number of molecule corresponding to Molecule struct (-1 for monomeric bead)
- int nAggregates

  number of aggregates the bead is in (only monomeric beads can be in more aggregates - allocated memory for 10)
- int * Aggregate

  index numbers of aggregates corresponding to Aggregate struct (-1 for bead in no aggregate)
- int Index

  index of the bead according to .vsf file (needed for indexed timesteps)
- Vector Position

  cartesian coordinates of the bead

8.3 BeadType Struct Reference

Information about bead types.

```
#include <Structs.h>
```

Data Fields

- char Name [16]

  name of given bead type
- int Number

  number of beads of given type
- bool Use

  should bead type in .vcf file be used for calculation?
- bool Write

  should bead type in .vcf file be written to output .vcf?
- double Charge

  charge of every bead of given type
- double Mass

  mass of every bead of given type

8.4 Counts Struct Reference

Total numbers of various things.

```
#include <Structs.h>
```

Data Fields

- int TypesOfBeads

  number of bead types
- int TypesOfMolecules

  number of molecule types
- int Beads

  total number of beads in all molecules
- int Bonded

  total number of beads in all molecules (TO BE REMOVED)
- int Unbonded

  total number of monomeric beads (TO BE REMOVED)
- int BeadsInVsf

  total number of all beads in .vsf file (not necessarily in .vcf)
- int Molecules

  total number of molecules
- int Aggregates

  total number of aggregates

8.5 IntVector Struct Reference

3D vector of integers.

```
#include <Structs.h>
```

Data Fields

- int **x**
- int **y**
- int **z**

8.6 LongVector Struct Reference

3D vector of floats.

```
#include <Structs.h>
```

Data Fields

- long double **x**
- long double **y**
- long double **z**

8.7 Molecule Struct Reference

Information about every molecule.

```
#include <Structs.h>
```

Data Fields

- int Type

  type of molecule corresponding to index in MoleculeType struct
- int ∗ Bead

  ids of beads in the molecule
- int Aggregate

  id of aggregate molecule is in (corresponding to index in Aggregate struct)

8.8 MoleculeType Struct Reference

Information about molecule types.

```
#include <Structs.h>
```

Data Fields

- char Name [16]

  name of given molecule type
- int Number

  number of molecules of given type
- int nBeads

  number of beads in every molecule of given type
- int nBonds

  number of bonds in every molecule of given type
- int ∗∗ Bond

  pair of ids for every bond (with relative bead numbers from 0 to nBeads)
- int nBTypes

  number of bead types in every molecule of given type
- int ∗ BType

  ids of bead types in every molecule of given type (corresponds to indices in BeadType struct)
- double Mass

  total mass of every molecule of given type
- bool InVcf

  is molecule type in vcf file?
- bool Use

  should molecule type be used for calculation?
- bool Write

  should molecule type be used for calculation?

8.9 Vector Struct Reference

3D vector of floats.

```
#include <Structs.h>
```

Data Fields

- double **x**
- double **y**
- double **z**

9 File Documentation

9.1 AnalysisTools.h File Reference

Functions common to all analysis utilities.

`#include "Structs.h"`

Functions

- void CommonHelp (bool error)

  Function printing help for common options.
- void VerboseOutput (bool Verbose2, char ∗input_vcf, char ∗bonds_file, Counts Counts, BeadType ∗BeadType, Bead ∗Bead, MoleculeType ∗MoleculeType, Molecule ∗Molecule)

  Function printing basic information about system if $-v$ *or* $-V$ *option is provided.*
- bool ReadStructure (char ∗vsf_file, char ∗vcf_file, char ∗bonds_file, Counts ∗Counts, BeadType ∗∗BeadType, Bead ∗∗Bead, MoleculeType ∗∗MoleculeType, Molecule ∗∗Molecule)

  Function reading information from dl_meso FIELD and vsf structure files.
- int ReadCoorOrdered (FILE ∗vcf_file, Counts Counts, Bead ∗∗Bead, char ∗∗stuff)

  Function reading ordered coordinates from .vcf coordinate file.
- int ReadCoorIndexed (FILE ∗vcf_file, Counts Counts, Bead ∗∗Bead, char ∗∗stuff)

  Function reading ordered coordinates from .vcf coordinate file.
- bool SkipCoor (FILE ∗vcf_file, Counts Counts, char ∗∗stuff)

  Function to skip one timestep in coordinates file.
- bool ReadAggregates (FILE ∗agg_file, Counts ∗Counts, Aggregate ∗∗Aggregate, MoleculeType ∗MoleculeType, Molecule ∗Molecule)

  Function reading information about aggregates from `.agg` *file.*
- void WriteCoorIndexed (FILE ∗vcf_file, Counts Counts, BeadType ∗BeadType, Bead ∗Bead, MoleculeType ∗MoleculeType, Molecule ∗Molecule, char ∗stuff)

  Function writing indexed coordinates to a .vcf file.
- int FindBeadType (char ∗name, Counts Counts, BeadType ∗BeadType)

  Function to identify type of bead from its name.
- int FindMoleculeType (char ∗name, Counts Counts, MoleculeType ∗MoleculeType)

  Function to identify type of molecule from its name.
- Vector Distance (Vector id1, Vector id2, Vector BoxLength)

  Function to calculate distance vector between two beads.
- void RemovePBCMolecules (Counts Counts, Vector BoxLength, BeadType ∗BeadType, Bead ∗∗Bead, MoleculeType ∗MoleculeType, Molecule ∗Molecule)

  Function to join all molecules.
- void RemovePBCAggregates (double distance, Aggregate ∗Aggregate, Counts Counts, Vector BoxLength, BeadType ∗BeadType, Bead ∗∗Bead, MoleculeType ∗MoleculeType, Molecule ∗Molecule)

  Funcion to join all aggregates.
- void RestorePBC (Counts Counts, Vector BoxLength, Bead ∗∗Bead)

Function to restore pbc.
- Vector CentreOfMass (int n, int ∗list, Bead ∗Bead, BeadType ∗BeadType)

  Function to calculate centre of mass for a collection of beads.
- Vector Gyration (int n, int ∗list, Counts Counts, Vector BoxLength, BeadType ∗BeadType, Bead ∗∗Bead)
- double Min3 (double x, double y, double z)

  Function returning the lowest number from three floats.
- Vector Sort3 (Vector in)

  Function returning sorted numbers $x < y < z$.
- void FreeBead (Counts Counts, Bead ∗∗Bead)

  Free memory allocated for Bead struct array.
- void FreeMolecule (Counts Counts, Molecule ∗∗Molecule)

  Free memory allocated for Molecule struct array.
- void FreeMoleculeType (Counts Counts, MoleculeType ∗∗MoleculeType)

  Free memory allocated for MoleculeType struct array.
- void FreeAggregate (Counts Counts, Aggregate ∗∗Aggregate)

  Free memory allocated for MoleculeType struct array.

9.1.1 Function Documentation

9.1.1.1 CentreOfMass()

```
Vector CentreOfMass (
            int n,
            int ∗ list,
            Bead ∗ Bead,
            BeadType ∗ BeadType )
```

Parameters

in	*n*	number of beads
in	*list*	list of bead ids (corresponding to indices in Bead struct)
in	*Bead*	information about individual beads (coordinates)
in	*BeadType*	information about beadtypes (masses)

Returns

coordinates of centre of mass of a given aggregate

Function to calculate centre of mass for a given list of beads.

9.1.1.2 CommonHelp()

```
void CommonHelp (
            bool error )
```

Parameters

in	*error*	`true` or `false` whether to use stderr or stdout

Function to print help for common options, either for `-h` help option or program error.

9.1.1.3 Distance()

```
Vector Distance (
            Vector id1,
            Vector id2,
            Vector BoxLength )
```

Parameters

in	*id1*	first coordinate vector
in	*id2*	second coordinate vector
in	*BoxLength*	dimensions of simulation box

Returns

distance vector between the two provided beads (without pbc)

Function calculating distance vector between two beads. It removes periodic boundary conditions and returns x, y, and z distances in the range <0, BoxLength/2).

9.1.1.4 FindBeadType()

```
int FindBeadType (
            char * name,
            Counts Counts,
            BeadType * BeadType )
```

Parameters

in	*name*	bead name
in	*Counts*	numbers of beads, residues, etc.
in	*BeadType*	information about bead types

Returns

bead type id corresponding to index in BeadType struct (or -1 if non-existent bead name)

9.1.1.5 FindMoleculeType()

```
int FindMoleculeType (
            char * name,
            Counts Counts,
            MoleculeType * MoleculeType )
```

Parameters

in	*name*	molecule name
in	*Counts*	numbers of beads, residues, etc.
in	*MoleculeType*	information about bead types

Returns

molecule type id corresponding to index in BeadType struct (or -1 for non-existent molecule)

9.1.1.6 FreeAggregate()

```
void FreeAggregate (
            Counts Counts,
            Aggregate ** Aggregate )
```

Parameters

in	*Counts*	number of beads, molecu.es, etc.
out	*Aggregate*	information about individual molecules

Free memory allocated for Aggregate struct array. This function makes it easier other arrays to the Aggregate struct in the future

9.1.1.7 FreeBead()

```
void FreeBead (
            Counts Counts,
            Bead ** Bead )
```

Parameters

in	*Counts*	number of beads, molecu.es, etc.
out	*Bead*	information about individual beads

Free memory allocated for Bead struct array. This function makes it easier to add other arrays to the Bead struct in the future

9.1.1.8 FreeMolecule()

```
void FreeMolecule (
            Counts Counts,
            Molecule ** Molecule )
```

Parameters

in	*Counts*	number of beads, molecu.es, etc.
out	*Molecule*	information about individual molecules

Free memory allocated for Molecule struct array. This function makes it easier other arrays to the Molecule struct in the future

9.1.1.9 FreeMoleculeType()

```
void FreeMoleculeType (
            Counts Counts,
            MoleculeType ** MoleculeType )
```

Parameters

in	*Counts*	number of beads, molecu.es, etc.
out	*MoleculeType*	information about individual molecules

Free memory allocated for MoleculeType struct array. This function makes it easier other arrays to the MoleculeType struct in the future

9.1.1.10 Gyration()

```
Vector Gyration (
             int n,
             int * list,
             Counts Counts,
             Vector BoxLength,
             BeadType * BeadType,
             Bead ** Bead )
```

Function to calculate the principle moments of the gyration tensor.

9.1.1.11 Min3()

```
double Min3 (
             double x,
             double y,
             double z )
```

Parameters

in	*x*	first double precision number
in	*y*	second double precision number
in	*z*	third double precision number

Returns

lowest of the supplied numbers

Function returning the lowest number from three floats.

9.1.1.12 ReadAggregates()

```
bool ReadAggregates (
             FILE * agg_file,
             Counts * Counts,
             Aggregate ** Aggregate,
             MoleculeType * MoleculeType,
             Molecule * Molecule )
```

Parameters

in	*agg_file*	name of input aggregate file
in	*Counts*	numbers of beads, molecules, etc.
out	*Aggregate*	information about aggregates
in	*MoleculeType*	information about molecule types
in	*Molecule*	information about individual molecules

Returns

1 if 'Last Step' detected or 0 for no error

Function reading information about aggregates from `.agg` file (Aggregate file) generated by Aggregates utility.

9.1.1.13 ReadCoorIndexed()

```
int ReadCoorIndexed (
            FILE * vcf_file,
            Counts Counts,
            Bead ** Bead,
            char ** stuff )
```

Parameters

in	*vcf_file*	name of input .vcf coordinate file
in	*Counts*	numbers of beads, molecules, etc.
out	*Bead*	coordinates of individual beads
out	*stuff*	first line of a timestep

Returns

0 for no errors or index number of bead (starting from 1) for which coordinates cannot be read

Function reading coordinates from .vcf file with indexed timesteps (Indexed coordinate file).

9.1.1.14 ReadCoorOrdered()

```
int ReadCoorOrdered (
            FILE * vcf_file,
            Counts Counts,
            Bead ** Bead,
            char ** stuff )
```

Parameters

in	*vcf_file*	name of input .vcf coordinate file
in	*Counts*	numbers of beads, molecules, etc.
out	*Bead*	coordinates of individual beads
out	*stuff*	first line of a timestep

Returns

0 for no errors or index number of bead (starting from 1) for which coordinates cannot be read

Function reading coordinates from .vcf file with ordered timesteps (OrderedCoorFile).

9.1.1.15 ReadStructure()

```
bool ReadStructure (
            char * vsf_file,
            char * vcf_file,
            char * bonds_file,
            Counts * Counts,
            BeadType ** BeadType,
            Bead ** Bead,
            MoleculeType ** MoleculeType,
            Molecule ** Molecule )
```

Parameters

in	*vsf_file*	.vsf structure file
in	*vcf_file*	.vcf coordinate file
in	*bonds_file*	filename with bonds
out	*Counts*	numbers of beads, molecules, etc.
out	*BeadType*	information about bead types
out	*Bead*	informationn about individual beads
out	*MoleculeType*	information about molecule types
out	*Molecule*	information about individual molecules

Returns

'true' or 'false' for .vcf file with indexed or ordered timesteps, respectively

Function reading information about beads and molecules from DL_MESO `FIELD` file, `.vsf` structure file, and `.vcf` coordinate file. Charge and mass of beads is read from `FIELD` file, but all other information is read from `vsf` structure file. If `vcf` coordinate is passed to `ReadStructure()`, bead types not present in the `vcf` file get ignored.

BOND FILE NOT IMPLEMENTED YET.

Optional file with bond declarations provides an alternative for bonds of any molecule type in `FIELD`. If optional bond file is not used, an empty string is passed to this function.

9.1.1.16 RemovePBCAggregates()

```
void RemovePBCAggregates (
            double distance,
            Aggregate * Aggregate,
            Counts Counts,
            Vector BoxLength,
            BeadType * BeadType,
            Bead ** Bead,
            MoleculeType * MoleculeType,
            Molecule * Molecule )
```

Parameters

in	*distance*	distance for closeness check (taken from agg file)
in	*Aggregate*	information about aggregates
in	*Counts*	number of beads, molecu.es, etc.
in	*BoxLength*	dimensions of the simulation box
in	*BeadType*	information about bead types
out	*Bead*	information about individual beads (coordinates)
in	*MoleculeType*	information about molecule types
in	*Molecule*	information about individual molecules

Function to remove periodic boundary conditions from all aggregates, thus joining them.

9.1.1.17 RemovePBCMolecules()

```
void RemovePBCMolecules (
            Counts Counts,
            Vector BoxLength,
            BeadType * BeadType,
            Bead ** Bead,
            MoleculeType * MoleculeType,
            Molecule * Molecule )
```

Parameters

in	*Counts*	numbers of beads, molecules, etc.
in	*BoxLength*	dimension of the simulation box
in	*BeadType*	information about bead types
out	*Bead*	information about individual beads (coordinates)
in	*MoleculeType*	information about molecule types
in	*Molecule*	information about individual molecules

Function to remove periodic boundary conditions from all individual molecules, thus joining them

9.1.1.18 RestorePBC()

```
void RestorePBC (
            Counts Counts,
            Vector BoxLength,
            Bead ** Bead )
```

Parameters

in	*Counts*	numbers of beads, molecules, etc.
in	*BoxLength*	dimension of the simulation box
out	*Bead*	information about individual beads (coordinates)

Function to restore removed periodic boundary conditions. Used in case of cell linked list, because it needs coordinates <0, BoxLength>.

9.1.1.19 SkipCoor()

```
bool SkipCoor (
            FILE * vcf_file,
            Counts Counts,
            char ** stuff )
```

Parameters

in	*vcf_file*	file with vcf coordinates
in	*Counts*	number of beads in vcf file
out	*stuff*	first line of a timestep

Returns

1 if premature end of file or 0 for no error

Function to skip one timestep in coordinates file. It works with both indexed and ordered vcf files.

9.1.1.20 Sort3()

```
Vector Sort3 (
            Vector in )
```

Parameters

in	*in*	first double precision number

Returns

sorted vector

Function returning sorted numbers x $<$ y $<$ z.

9.1.1.21 VerboseOutput()

```
void VerboseOutput (
            bool Verbose2,
            char * input_vcf,
            char * bonds_file,
            Counts Counts,
            BeadType * BeadType,
            Bead * Bead,
            MoleculeType * MoleculeType,
            Molecule * Molecule )
```

Parameters

in	*Verbose2*	print extra information if 'true'
in	*input_vcf*	.vcf structure file
in	*bonds_file*	filename with bonds
in	*Counts*	numbers of beads, molecules, etc.
in	*BeadType*	information about bead types
in	*Bead*	informationn about individual beads
in	*MoleculeType*	information about molecule types
in	*Molecule*	information about individual molecules

Function providing standard verbose output (for cases when verbose option is used). It prints most of the information about used system.

9.1.1.22 WriteCoorIndexed()

```
void WriteCoorIndexed (
            FILE * vcf_file,
            Counts Counts,
            BeadType * BeadType,
            Bead * Bead,
            MoleculeType * MoleculeType,
            Molecule * Molecule,
            char * stuff )
```

Parameters

in	*vcf_file*	name of output .vcf coordinate file
in	*Counts*	numbers of beads, molecules, etc.
in	*BeadType*	information about bead types
in	*Bead*	coordinates of individual beads
in	*MoleculeType*	information about molecule types
in	*Molecule*	coordinates of individual molecules
in	*stuff*	array of chars containing comment line to place at the beginning

Function writing coordinates to a `.vcf` file. According to the Use flag in BeadType structure only certain bead types will be saved into the indexed timestep in .vcf file (Indexed coordinate file).

9.2 Options.h File Reference

Options usable in utilities.

`#include "Structs.h"`

Functions

- bool VsfFileOption (int argc, char **argv, char **vsf_file)

  Option whether to use `.vsf` file different from `dl_meso.vsf` (`-i <name.vsf>`).
- bool BondsFileOption (int argc, char **argv, char **bonds_file)

  Option whether to use bonds file (`-b <name>`).
- void VerboseLongOption (int argc, char **argv, bool *verbose, bool *verbose2)

  Option whether to use long verbose output (overrides VerboseShortOutput) (`-V`).
- void SilentOption (int argc, char **argv, bool *verbose, bool *verbose2, bool *silent)

  Option whether not to print to stdout (overrides Verbose options) (`-s`).
- bool ExcludeOption (int argc, char **argv, Counts Counts, MoleculeType **MoleculeType)

  Option whether to exclude molecule types (`-x <name(s)>`).
- bool JoinCoorOption (int argc, char **argv, char *joined_vcf)

  Option whether to write joined aggregate coordinates to file (`-j <joined.vcf>`).
- bool BeadTypeOption (int argc, char **argv, Counts Counts, BeadType **BeadType)

  Option to choose which bead types to use in calculations (`-bt <name(s)>`).
- bool BoolOption (int argc, char **argv, char *opt)

  Option whether not to print rewrite stdout line (`--script`).
- bool IntegerOption (int argc, char **argv, char *opt, int *value)

  Function for any option with integer argument.
- bool TwoIntegerOption (int argc, char **argv, char *opt, int *values)

  Function for any option with two integer arguments.
- bool FileOption (int argc, char **argv, char *opt, char **name)

  Function for any option with filename.

- bool MoleculeTypeOption (int argc, char **argv, char *opt, int *moltype, Counts Counts, MoleculeType **MoleculeType)

    Function for any option with molecule type name.
- bool MoleculeTypeOption2 (int argc, char **argv, char *opt, int **moltype, Counts Counts, MoleculeType **MoleculeType)

    Function for any option with molecule type name(s).

9.2.1 Function Documentation

9.2.1.1 BeadTypeOption()

```
bool BeadTypeOption (
             int argc,
             char ** argv,
             Counts Counts,
             BeadType ** BeadType )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*Counts*	numbers of beads, molecules, etc.
out	*BeadType*	information about bead types

Returns

`true` or `false` error or not error

Option to choose which bead types to use for calculation. If the option is absent, all bead types are switched to `Use = true`. Argument: `-bt <name(s)>`

9.2.1.2 BondsFileOption()

```
bool BondsFileOption (
             int argc,
             char ** argv,
             char ** bonds_file )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
out	*bonds_file*	filename with bonds

Returns

`true` or `false` error or not error

Option whether to use bonds file with alternative bond definitions.

9.2.1.3 BoolOption()

```
bool BoolOption (
            int argc,
            char ** argv,
            char * opt )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*opt*	option switch (e.g. array containing `-n`)

Returns

`true` if `opt` present, `false` otherwise

Function for any boolean option (i.e. without argument). The option (e.g. `--script`) is an argument of this function.

9.2.1.4 ExcludeOption()

```
bool ExcludeOption (
            int argc,
            char ** argv,
            Counts Counts,
            MoleculeType ** MoleculeType )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*Counts*	numbers of beads, molecules, etc.
out	*MoleculeType*	information about molecule types

Returns

`true` or `false` error or not error

Option to exclude specified molecule types from calculations. Gives specified molecule types `Use = false` and the rest `Use = true`. Arguments: `-x <name(s)>`

9.2.1.5 FileOption()

```
bool FileOption (
            int argc,
            char ** argv,
            char * opt,
            char ** name )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*opt*	option switch (e.g. array containing `-n`)
out	*name*	array containing the filename

Returns

`true` or `false` for error

Generic option for file name. The option is an argument of this function.

9.2.1.6 IntegerOption()

```
bool IntegerOption (
            int argc,
            char ** argv,
            char * opt,
            int * value )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*opt*	option switch (e.g. array containing `-n`)
out	*value*	integer value of given option

Returns

`true` or `false` for error

Function for any option with integer argument. The option (e.g. `-n`) is an argument of this function.

9.2.1.7 JoinCoorOption()

```
bool JoinCoorOption (
            int argc,
            char ** argv,
            char * joined_vcf )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*Counts*	numbers of beads, molecules, etc.
out	*MoleculeType*	information about molecule types

Returns

`true` or `false` error or not error

Option whether to join aggregates and save joined coordinates into a specified file. Arguments: `-j <joined.↩ vcf>`

9.2.1.8 MoleculeTypeOption()

```
bool MoleculeTypeOption (
            int argc,
            char ** argv,
            char * opt,
            int * moltype,
            Counts Counts,
            MoleculeType ** MoleculeType )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*opt*	option switch (e.g. array containing `-n`)
in	*Counts*	numbers of beads, molecules, etc.
out	*moltype*	id of the molecule type
in	*MoleculeType*	information about molecule types

Returns

`true` or `false` for error

Generic option for molecule type that can take more than one argument. The option is an argument of this function.

9.2.1.9 MoleculeTypeOption2()

```
bool MoleculeTypeOption2 (
            int argc,
            char ** argv,
            char * opt,
            int ** moltype,
            Counts Counts,
            MoleculeType ** MoleculeType )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
in	*opt*	option switch (e.g. array containing `-n`)
in	*Counts*	numbers of beads, molecules, etc.
out	*moltype*	array for the molecule type
in	*MoleculeType*	information about molecule types

Returns

`true` or `false` for error

Generic option for molecule types. The option is an argument of this function.

9.2.1.10 SilentOption()

```
void SilentOption (
            int argc,
            char ** argv,
            bool * verbose,
            bool * verbose2,
            bool * silent )
```

Parameters

`in`	*argc*	number of program's arguments
`in`	*argv*	program's arguments
`out`	*verbose*	bool for `-v` option (verbose output)
`out`	*verbose2*	bool for `-V` option (detailed verbose output)
`out`	*silent*	bool for this option

Returns

`true` or `false` for error on common options

Option to not print anything to stdout (or at least no system definitions and no Step: #). Overrides VerboseShort↩
Option and VerboseLongOption. Argument: `-s`

9.2.1.11 TwoIntegerOption()

```
bool TwoIntegerOption (
            int argc,
            char ** argv,
            char * opt,
            int * values )
```

Parameters

`in`	*argc*	number of program's arguments
`in`	*argv*	program's arguments
`in`	*opt*	option switch (e.g. array containing `-n`)
`out`	*values*	array of two integer values of given option

Returns

`true` or `false` for error

Function for any option with two integer arguments. The option (e.g. `-n`) is an argument of this function.

9.2.1.12 VerboseLongOption()

```
void VerboseLongOption (
             int argc,
             char ** argv,
             bool * verbose,
             bool * verbose2 )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
out	*verbose*	bool for `-v` option (verbose output)
out	*verbose2*	bool for `-V` option (detailed verbose output)

Returns

`true` or `false` for error on common options

Option whether to print detailed data to stdout. Data are printed via VerboseOutput() function (and possibly some in-program code). Argument: `-V`

9.2.1.13 VsfFileOption()

```
bool VsfFileOption (
             int argc,
             char ** argv,
             char ** vsf_file )
```

Parameters

in	*argc*	number of utility's arguments
in	*argv*	utility's arguments
out	*vsf_file*	filename with structure information

Returns

`true` or `false` error or not error

Option whether to use `.vsf` file different from the default `dl_meso.vsf`.

9.3 Structs.h File Reference

Structures for utilities.

Data Structures

- struct Vector

    3D vector of floats.

- struct LongVector

  3D vector of floats.
- struct IntVector

  3D vector of integers.
- struct Counts

  Total numbers of various things.
- struct BeadType

  Information about bead types.
- struct MoleculeType

  Information about molecule types.
- struct Bead

  Information about every bead.
- struct Molecule

  Information about every molecule.
- struct Aggregate

  Information about every aggregate.

Macros

- #define PI 3.141593

  value of pi
- #define SQR(x) ((x)∗(x))

  macro for algebraic square
- #define CUBE(x) ((x)∗(x)∗(x))

  macro for algebraic cube

Typedefs

- typedef struct Vector Vector

  3D vector of floats.
- typedef struct LongVector LongVector

  3D vector of floats.
- typedef struct IntVector IntVector

  3D vector of integers.
- typedef struct Counts Counts

  Total numbers of various things.
- typedef struct BeadType BeadType

  Information about bead types.
- typedef struct MoleculeType MoleculeType

  Information about molecule types.
- typedef struct Bead Bead

  Information about every bead.
- typedef struct Molecule Molecule

  Information about every molecule.
- typedef struct Aggregate Aggregate

  Information about every aggregate.