Analysis Software Package

Generated by Doxygen 1.8.13

Contents

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 ordered timesteps or 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.

The first mandatory line specifies default bead type which means all atom lines for beads of this type are unnecessary (provided those beads are not in a molecule). All atom lines in `dl_meso.vsf` specify VDW radius and atom name. If an atom is in a molecule, its molecule number is appended to the atom line as `resid <id>`:

```
atom default radius 1.000000 name <name>
...
atom <id> radius 1.000000 name <name> resid <id>
...
```

Only the bead number and name are read, so both VDW radius and molecule number are not strictly necessary. Short version of `atom` and `name` keywords (`a` and `n` respectively) can be used. Other keywords can be included, because they will be ignored. No comments are allowed in `.vsf` file.

Bond lines of `.vsf` files are not read and are therefore irrelevant to all the utilities.

1.2 Optional bond file

Bonds for each molecule type are specified in `FIELD`, but they can be read from a different file if required.

The file with molecule bonds must contain name of the molecule type (same as in `FIELD`) followed by number of bonds on the next line and on every subsequent line two index numbers corresponding to the two connected beads (bead numbers start from one and are ordered according to beads in `FIELD` section for the given molecule). The bead numbers do not have to be sorted in any way and no blank lines should be present.

Example of bond file:

```
triangle
3
1 2 possible
3 1
2 3 comment
```

This file must be used for molecule types that have only some of its beads in `.vcf` file with indexed timesteps. In such a case, the bead indices correspond to `FIELD` as if the bead types not present in `.vcf` are not present `FIELD`.

Example of the relevant part of `FIELD`:

```
...
beads 3
A <float> <float> <float>
B <float> <float> <float>
A <float> <float> <float>
bonds 3
harm 1 2 <float> <float>
harm 1 3 <float> <float>
harm 2 3 <float> <float>
finish
```

Assuming only bead types `A` are present in `.vcf` file, the now necessary bond file would like like this:

```
name
1
1 2 possible comment
```

Should the bond file not be provided such case, the utilities detect no error, but will not work correctly (and may crash with segmentation fault).

Bond information about molecule types not present in the bond file will be read from `FIELD`.

1.3 Ordered coordinate file

First line of `.vcf` file with ordered timestep(s) contains box size. Each timestep starts with a comment line (i.e. line starting with `#` sign), the second line contains `timestep` (or the short version, `t`) and each following line contains the coordinates of a single bead. Every bead from `.vsf` structure file must be present in each timestep.

Exactly one blank line must be between every two timesteps and no blank lines are allowed at the end of the file.

Example of ordered coordinate file:

```
pbc <float> <float> <float>
<blank line>
# 1
timestep
<float> <float> <float>
...
```

An ordered coordinate file is generated using `traject-v2_5`, but `traject-v2_6` produces indexed coordinate file (due to the way the original `traject` utility prints coordinates).

1.4 Indexed coordinate file

Unlike the `.vcf` file with ordered timesteps, the `.vcf` file with indexed timestep does not contain coordinates for every bead. Only beads of selected bead types are present and their names are written as comments at the beginning of the file (and followed by a blank line). Every bead is prepended by its index number according to the `.vsf` structure file. Keyword `timestep` (or `t`) at the beginning of every timestep is replaced by `indexed` (or its short version, `i`). The beads does not have to be ordered in any way, but the same number of beads must be in every timestep and every of given types must be present. Otherwise the file has the same format as the `.vcf` file with ordered timesteps.

Example of indexed coordinate file:

```
# name
...
<blank line>
pbc <float> <float> <float>
<blank line>
# 1
indexed
<id> <float> <float> <float>
...
```

1.5 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.

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 in the aggregate :  <bead ids>
<blank line>
Step:  2
...
<blank line>
Last Step:  <number>
```

2 Options for all utilities

A great majority of utilities has 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`)

`-b <name>`

file containing bond alternatives to `FIELD` (see Optional bond file for explanation)

`-v`

verbose output providing information about the system

`-V`

more detailed verbose output (also prints comments from `.vcf` file at the start of every timestep)

`-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 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.

Usage:

```
traject <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.

3.2 SelectedVcf utility

This utility takes `.vcf` file containing either ordered timesteps (such as `All.vcf` created by DL_MESO `traject` utility which was modified by me) 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 therefore contains indexed timesteps.

Usage:

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

`<input.vcf>`

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

`<start>`

number of timestep to start from

`<skip>`

leave out every `skip` steps

`<output.vcf>`

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

`<type names>`

names of bead types to save

`<options>`

`-j`

join individual molecules by removing periodic boundary conditions

`-st <int>`

starting timestep for calculation

`-sk <int>`

number of steps to skip per one used

3.3 Config utility

This utility takes `.vcf` file containing either ordered timesteps (such as `All.vcf` created by DL_MESO `traject` utility which was modified by me) or indexed timesteps and creates `CONFIG` file (file containing initial coordinates for a simulation via `DL_MESO simulation package`).

Usage:

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

`<input.vcf>`

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

3.4 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 well suited for visualisation using VMD software.

Usage:

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

`<output.vsf>`

output structure file that must end with `.vsf`

3.4.1 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)

Todo Optional bond file format: somehow avoid the need to use the special optional bond file, where the ids of beads must strictly adhere to FIELD. Possibly require use of a vcf file in conjunction with bond file

JoinRuns: implement wholy `--script` common option

3.5 BondLength utility

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

Usage:

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

`<input.vcf>`

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

`<output file>`

output filename containing distribution of bond lengths

`<width>`

width of each bin for the distribution

`<molecule names>`

names of molecule types to calculate the distribution for

3.6 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. That is, if bead types `A`, `B` and `C` are given, Aggregates utility will use all six bead type pairs, that is `A-A`, `A-B`, `A-C`, `B-B`, `B-C` and `C-C` (provided the beads are in different molecules), but Aggregates-NotSame↩ Beads will not use `A-A`, `B-B` or `C-C` contacts. Therefore at least two bead types must be provided for `<type names>` argument in Aggregates-NotSameBeads.

Usage:

```
Aggregates <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`)

3.7 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

3.8 DistrAgg utility

DistrAgg calculates number and weight average aggregation numbers for each timestep (time evolution). The number average aggregation number, $\langle A_\mathrm{s} \rangle_n$ is defined as:

$$\langle A_\mathrm{s} \rangle_n = \frac{\sum_{i=1}^{N} m_i}{N}, \tag{1}$$

where m_i is weight (aggregation number) of aggregate i and N is total number of aggregates. The weight average aggregation number, $\langle A_\mathrm{s} \rangle_w$ is then defined as:

$$\langle A_\mathrm{s} \rangle_w = \frac{\sum_{i=1}^{N} m_i^2}{\sum_{i=1}^{N} m_i}. \tag{2}$$

It also calculates overall number and weight distribution function. The number distribution function, $F_n(A_\mathrm{s})$ is defined as:

$$F_n(A_\mathrm{s}) = \frac{N_{A_\mathrm{s}}}{\sum_{i=1}^{N} N_i}, \tag{3}$$

where N_i is the number of aggregates with aggregation number $A_\mathrm{s} = i$. The weight distribution function, $F_w(A_\mathrm{s})$ is then defined as:

$$F_w(A_\mathrm{s}) = \frac{m_{A_\mathrm{s}} N_{A_\mathrm{s}}}{\sum_{i=1}^{N} m_i N_i}, \tag{4}$$

where m_{A_s} and m_i are again the weight, that is the aggregation number.

Lastly, the utility calculates volume fractions of all aggregates, where it (for now) assumes that all beads have reduced mass of 1. Volume fraction of an aggregate with aggregation number A_s is defined as:

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

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

It should be noted that weight average aggregation number and weight distribution function do not take into account the actual weight of an associates – it is weighted via the aggregation number itself.

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

Usage:

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

`<input>`

input filename with information about aggregates

`<output distr file>`

output filename with weight and number distribution functions

`<output avg file>`

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

`<options>`

`-n <int>`

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

`--no-unimers`

free chains shouldn't be used to calcalute average aggregation numbers

3.9 DensityAggregates

This utility calculates number bead density for aggregates of specified size from their center 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).

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>`

`-j`

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

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

3.10 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 center 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 center 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 center of mass for `NameA`, then an argument of the form `-c x 1` must be used.

Usage:

```
DensityMolecules <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>`

`-j`

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

`-n <average>`

number of bins to average

`-c <int>`

use specified molecule bead instead of center of mass

3.11 GyrationAggregates utility

This utility calculates a gyration tensor and its eigenvalues (using Jacobi transformations) for aggregates of given size. It then determines various shape descriptors. It saves averages during the simulation (time evolution) to an output file and prints overall averages to standard output.

It calculates radius of gyration, R_{G}:

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

where λ_i^2 is the i-th principle moment of the tensor of gyration ($\lambda_x^2 \leq \lambda_y^2 \leq \lambda_z^2$). 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:

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

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>`

`-j`

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

`-t`

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

Todo GyrationAggregates: understand `jacobi` function

3.12 GyrationMolecules utility

This utility function in the same way as GyrationAggregates, but it calculates radii of gyration for specified molecule names 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>`

`-j`

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

Todo GyrationMolecules: implement using only selected bead types for calculation

Todo GyrationMolecules: understand `jacobi` function

Todo Gyration: move function from GyrationAggregates and GyrationMolecules to a separate header file

3.13 JoinRuns utility

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>`

`-j`
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

3.14 Average utility

Utility calculating average values with standard deviation and autocorrelation time from values contained in a text file. The first line of the file has to contain the number of data lines and no comments are allowed.

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

Todo Average: completely rewrite - especially remove requirement for number of lines on the first line of input file

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

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_t is used:

$$l_\mathrm{P} = \langle b \rangle \sum_{i=0}^{i=N_b} \langle \cos\theta_i \rangle, \tag{10}$$

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.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

5 Todo List

Page Common utilities

Optional bond file format: somehow avoid the need to use the special optional bond file, where the ids of beads must strictly adhere to FIELD. Possibly require use of a vcf file in conjunction with bond file

JoinRuns: implement wholy `--script` common option

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

GyrationAggregates: understand `jacobi` function

GyrationMolecules: implement using only selected bead types for calculation

GyrationMolecules: understand `jacobi` function

Gyration: move function from GyrationAggregates and GyrationMolecules to a separate header file

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

Average: completely rewrite - especially remove requirement for number of lines on the first line of input file

6 Data Structure Index

6.1 Data Structures

Here are the data structures with brief descriptions:

7 File Index

7.1 File List

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

8 Data Structure Documentation

8.1 Aggregate Struct Reference

Information about every aggregate.

`#include <AnalysisTools.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

8.2 Bead Struct Reference

Information about every bead.

`#include <AnalysisTools.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 <AnalysisTools.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 <AnalysisTools.h>`

Data Fields

- int TypesOfBeads
  number of bead types
- int TypesOfMolecules
  number of molecule types
- int Bonded
  total number of beads in all molecules
- int Unbonded
  total number of monomeric beads
- 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 <AnalysisTools.h>`

Data Fields

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

8.6 Molecule Struct Reference

Information about every molecule.

`#include <AnalysisTools.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)

8.7 MoleculeType Struct Reference

Information about molecule types.

`#include <AnalysisTools.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 Use
  should molecule type be used for calculation?

8.8 Vector Struct Reference

3D vector of floats.

`#include <AnalysisTools.h>`

Data Fields

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

9 File Documentation

9.1 AnalysisTools.h File Reference

Structures and functions common to all analysis utilities.

Data Structures

- struct Vector
  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 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.

Functions

- void CommonHelp (bool error)

  Function printing help for common options.
- bool CommonOptions (int argc, char **argv, char **vsf_file, char **bonds_file, bool *verbose, bool *verbose2, bool *silent, bool *script)

  Function to setup 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.
- int SkipCoor (FILE *vcf_file, Counts Counts, char **stuff)

  Function to skip one timestep in coordinates file.
- void 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, 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 CenterOfMass (int n, int *list, Bead *Bead, BeadType *BeadType)

  Function to calculate center of mass for a collection of beads.
- 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 CenterOfMass()

```
Vector CenterOfMass (
            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 center of mass of a given aggregate

Function to calculate center 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 CommonOptions()

```
bool CommonOptions (
            int argc,
            char ** argv,
            char ** vsf_file,
            char ** bonds_file,
            bool * verbose,
            bool * verbose2,
            bool * silent,
            bool * script )
```

Parameters

in	*argc*	number of program's arguments
in	*argv*	program's arguments
out	*vsf_file*	filename with structure information
out	*bonds_file*	filename with bonds
out	*verbose*	bool for `-v` option (verbose output)
out	*verbose2*	bool for `-V` option (detailed verbose output)
out	*silent*	bool for `-s` option (run silently)
out	*script*	bool for `--script` option (run in script)

Returns

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

Function for options common to most of the utilities.

9.1.1.4 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).

9.1.1.5 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)

9.1.1.6 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)

9.1.1.7 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

9.1.1.8 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

9.1.1.9 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

9.1.1.10 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

9.1.1.11 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()

```
void 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

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 (Ordered coordinate file).

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 and a `.vsf` structure file. Name, mass and charge of every bead type is read from `species` lines in `FIELD`. The number of molecule types are read from `molecule` section. For each molecule type its name, the number of molecules, the number of beads and bonds in each molecule and the bonds themselves are read. Input structure file provides information about what bead is of which type. 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()

```
int 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,
            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	*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).