MD-TASK Documentation

Release 1.0.1

Mar 09, 2020

Getting started

CHAPTER 1

MD-TASK

MD-TASK consists of a suite of Python scripts that have been developed to analyze molecular dynamics trajectories. These scripts fall into 3 categories:

1. Residue Interaction Network (RIN) analysis
2. Perturbation Response Scanning (PRS)
3. Dynamic Cross-Correlation

1.1 Contribute

- Issue Tracker: https://github.com/RUBi-ZA/MD-TASK/issues
- Source Code: https://github.com/RUBi-ZA/MD-TASK

1.2 Citing MD-TASK

You can cite us here: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5860072/

1.3 License

The project is licensed under GNU GPL 3.0

CHAPTER 2

Installation

2.1 Platform compatibility

MD-TASK should be compatible with any Linux/Unix-based platform, although installation of system dependencies may differ. It has been successfully tested on the following platforms:

- Ubuntu Linux
- MacOS
- Windows 10 (with bash)

2.2 Install system dependencies

Note: package version numbers may differ depending on the OS version. For example, in Ubuntu 16.04, 'libpng12-dev' must be installed. However, in Ubuntu 17.04, 'libpng-dev' should be installed.

Ubuntu 16.04:

```
sudo apt-get install virtualenvwrapper python-dev libblas-dev liblapack-dev libatlas-
↪base-dev gfortran libpng12-dev libfreetype6-dev python-tk r-base
```

Windows 10:

1. Enable the Windows Subsystem for Linux (WSL) by following these instructions.
2. Install the system dependencies as with Ubuntu above.

MacOS:

1. On MacOS, Python comes installed by default, but the default version my not be ideal. Follow these instructions to install a more up-to-date version of Python.
2. Next, install virtualenv by following these instructions

2.3 Download the project

MD-TASK can be cloned from it's GitHub repository

```
git clone https://github.com/RUBi-ZA/MD-TASK.git
cd MD-TASK
```

2.4 Install Python dependencies

We recommend using a Python virtual environment when using MD-TASK. It can be set up by running the *install.sh* script in the root directory of the MD-TASK repository:

```
sh install.sh
```

You should now see a directory, *venv*. This is your Python virtual environment. This environment should always be activated before using MD-TASK. The virtual environment can be activated with the following command:

```
. venv/bin/activate
```

2.5 Install R dependencies

Install the igraph package for R:

```
R
> install.packages("igraph")
```

CHAPTER 3

General

3.1 Activating the virtual environment

If the installation recommendations on the previous page were followed, you would have set up a virtual Python environment for MD-TASK. If that is so, it is important that the environment be active whenever you use MD-TASK. To activate the environment, run the following command in the root MD-TASK folder (if that is where the environment was installed):

```
. venv/bin/activate
```

You will now have all the installed dependencies available and MD-TASK should work perfectly.

3.2 Add MD-TASK to your PATH

To make tools in the MD-TASK suite available from anywhere on the command line, add the root MD-TASK directory to your PATH environment variable as follows:

```
export PATH=/path/to/MD-TASK:$PATH
```

3.3 Trajectory vs Topology

Most of the MD-TASK tools require both a trajectory file and topology/structure file as input. This is because most trajectory formats only contain the atom co-ordinates and not the topological information such as atom and residue names, chains, and bond information. The topology file can be the PDB file that was used in the molecular dynamics simulation to produce the trajectory. When supplying these files, it is important to note that the trajectory file and topology file must contain the exact same number of atoms i.e. if the trajectory has been reduced to CA and CB atoms only (as described below), the topology file must be reduced to the same.

3.4 Reducing your trajectory

Molecular dynamics trajectories can be extremely large. However, MD-TASK tools only require the alpha and beta carbon atoms to be present. To save space and improve performance, the following VMD script can be used to reduce a trajectory, to the bare essentials:

```
mol load pdb example.pdb
set s1 [atomselect top "name CA or name CB and not solvent"]
animate write pdb example_small.pdb sel $s1
animate read xtc example.xtc waitfor all
animate write dcd example_small.dcd waitfor all sel $s1
quit
```

The above assumes that your topology file is a PDB file named `example.pdb` and that your trajectory is named `example.xtc`. It then writes out the reduced structure and trajectory to `example_small.pdb` and `example_small.dcd` respectively. You should change these names accordingly. You will also note that the above converts the trajectory from XTC to DCD format. This is not necessary, but has been added as an example for those who may want to do it.

For very large trajectories that do not fit in memory, reducing as shown above is necessary. Note that when reducing the trajectory, it is important that the same reduction should be applied to the topology PDB file i.e. the trajectory and topology files should have the exact same number of atoms. Failing to do this will result in an error.

3.5 Test Data

There is test data located in the 'examples' directory. Four files are included here:

File	Description
`wt.dcd`	An example trajectory that has been reduced to alpha and beta carbons only (used in the network analysis section)
`wt.pdb`	A PDB file that corresponds to the above trajectory - to be used for topology information (used in the network analysis section)
`mutant.dcd`	A mutated version of the above trajectory (used in the network analysis section)
`mutant.pdb`	A mutated version of the above topology file (used in the network analysis section)
`example_small.dcd`	An example trajectory that has been reduced to alpha and beta carbons only (used in the PRS section)
`example_small.pdb`	A PDB file that corresponds to the above trajectory - to be used for topology information (used in the PRS section)
`initial.xyz`	An XYZ co-ordinate file representing the initial conformation of a protein (used for PRS)
`final.xyz`	An XYZ co-ordinate file representing the target conformation of a protein (used for PRS)

3.6 Logging

All scripts in the suite have two arguments for logging. By default, logging is switched on and is written to the terminal. This can be changed with the following arguments:

Input	Flag	Description
Log file	`--log-file`	Provide a path to a file that will store the logging output of the command. By default, the output will be written to the terminal.
Silent	`--silent`	Switch off logging

CHAPTER 4

Network Analysis

Residue Interaction Networks (RIN) are analyzed using a branch of Mathematics known as graph theory. In a RIN, each residue in the protein is a node in the network. An edge (or connection) between two nodes exists if there is an interaction between the two residues those nodes represent. MD-TASK considers an interaction between two residues to exist if the beta carbon atoms of the residues are within a user-defined cut-off distance (usually around 6.5 – 7.5 Å) of each other. Once the network has been constructed, there are various network measures that can be used to analyze it. Currently, MD-TASK can be used to analyze the change in betweenness centrality (BC) and average shortest path (L) of residues in a protein over a molecular dynamics simulation. This can be used to determine which residues are important for intra-protein communication and conformational changes. RINs can also be useful in the analysis of SNPs. Comparing changes in BC and L between the simulation of a wild-type and mutant protein can provide interesting insights into differences in intra-protein communication, which can affect the function of the protein.

4.1 Measurements

1. Betweenness Centrality (BC)

Betweenness centrality (BC) is a measure of how important a residue is for communication within a protein. It is equal to the number of shortest paths from all vertices to all others that pass through that node. Residues in a protein that have a high BC reveal locations that tend to be important for controlling inter-domain communication in a protein.

2. Average Shortest Path(L)

The average shortest path (L) to a given residue is calculated by working out all the shortest paths to the given node and dividing by the number of paths. The average shortest path to a residue gives an idea of how accessible the residue is within the protein. This can be used to, for example, analyze SNPs. A mutation may result in a change in L of a number of residues in the protein. This may indicate that the mutation has an important effect on protein function e.g. previous studies have suggested that positions that result in high delta L values may steer conformational changes.

4.2 Calculating BC and L

Command:

```
calc_network.py <options> --topology <pdb file> <trajectory>
```

Inputs:

Input (*required*)	Input type	Flag	Description
Trajectory *	File		A trajectory from a molecular dynamics simulation. Can be in DCD or XTC format.
Topology *	File	`--topology`	A PDB reference file for the trajectory.
Ligands	CSV ligand IDs	`--ligands`	Ligands to include in the network construction. Syntax resname1:atom,resname2:atom
Threshold	Integer	`--threshold`	Distance threshold when constructing network.
Step	Integer	`--step`	Step to use when iterating through trajectory frames.
Generate plots	Boolean	`--generate-plots`	Set to generate figures.
Calculate BC	Boolean	`--calc-BC`	Set to calculate average shortest path matrix for the network
Calculate L	Boolean	`--calc-L`	Set to calculate betweenness centrality matrix for the network
Discard graphs	Boolean	`--discard-graphs`	Set to discard the network once BC and L matrices have been calculated
Lazy load	Boolean	`--lazy-load`	Load trajectory frames in a memory efficient manner - use for large trajectories

Note: for `--calc-L` *to work, all nodes in the network must be accessbile from all other nodes in the network. When this is not the case, an error will occur. Try increasing the distance threshold when this happens.*

Given a trajectory called `wt.dcd` and a topology file called `wt.pdb`, the following command could be used:

```
calc_network.py --topology wt.pdb --threshold 7.0 --step 100 --generate-plots --calc-
↪BC --calc-L --discard-graphs --lazy-load wt.dcd
```

The above command will calculate the network for every 100th frame in the trajectory. Depending on the size of your trajectory, you may want to increase this `--step`. Because `--lazy-load` was used, the trajectory will be iterated through and frames will be loaded one-at-a-time and then discarded once the network for that frame has been calculated. Leaving out the `--lazy-load` argument will result in the entire trajectory being loaded into memory. This can be faster for small trajectories, but should be avoided when analysing large trajectories. Edges in the network will be created between nodes that are within 7 Angstroms of each other. The average shortest path for each residue in each frame and the betweenness centrality of each residue in each frame will be calculated as **both flags have been set** in the above command. In addition, the `--discard-graphs` flag was set. As such, the networks for each frame will be discarded once BC and L have been calculated, saving disk space. By default, the networks for each frame are saved in both `gml` and `graphml` format.

Outputs:

Output	Description
BC Matrices	For each frame analyzed, an Nx1 matrix is produced, where N is the number of residues in the protein and each value represents the BC for the residue at that index
avg_L Matrices	For each frame analyzed, an Nx1 matrix is produced, where N is the number of residues in the protein and each value represents the L to the residue at that index
BC & L Plots	If `--generate-plots` flag is set, PNG figures are produced for the BC and L matrices
Network graphs	If `--discard-graphs` flag is set, do not save the networks produced for each frame

4.3 Calculating ΔL

If the `--calc-L` flag in the previous command is set, a number of Nx1 L matrices will be generated. Given the trajectory `wt.dcd`, the matrices will be named `wt_<frame>_avg_L.dat`, where `<frame>` is the frame index in the trajectory.

Command:

```
calc_delta.py --matrix-type L --reference <frame> --alternatives <frames>
```

This script replaces the, now deprecated, `calc_delta_L.py` script, which will be removed from MD-TASK in version 2.0 and onwards:

```
calc_delta_L.py <options> --reference <frame> --alternatives <frames>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Reference frame *	File	`--reference`	Nx1 matrix to be used as the reference (normally the frame from time 0). Delta L will be worked out by comparing the alternative frames to this one.
Alternative frames *	File/s	`--alternatives`	The remaining Nx1 matrices that should be compared to the reference matrix
Normalize	Boolean	`--normalize`	Set this flag to normalize the values
Normalization mode	Text	`--normalization-mode`	Options are `standard` (ΔL/L), `plusone` (ΔL/(L+1)), or `nonzero` (ΔL/L where L > 0 else ΔL) - default mode is `standard`
Generate plots	Boolean	`--generate-plots`	Set to generate figures

Given a set of average shortest path .dat files `wt_*_avg_L.dat` (generated with `calc_network.py`), the `wt_0_avg_L.dat` file could be used as the reference and the rest could be used as the alternatives. If `wt_0_avg_L.dat` is renamed to `ref_wt_L.dat`, the following command could be used:

```
calc_delta_L.py --normalize --generate-plots --reference ref_wt_L.dat --alternatives␣
↪wt_*_avg_L.dat
```

The above command will generate plots as well as Nx1 matrices representing the difference in L between each alternative and the reference frame. The values will be normalized by dividing by the reference values (ΔL/L).

Outputs:

Output	Description
ΔL Matrices	Nx1 matrices representing the change in L between the reference matrix and each alternative
ΔL Plots	Figures for each alternative frame, plotting the difference between L in the alternative and reference

4.4 Calculating ΔBC

If the `--calc-BC` flag was set when running the `calc_network.py` script, a number of Nx1 BC matrices will be generated. Given the trajectory `wt.dcd`, the matrices will be named `wt_<frame>_bc.dat`, where `<frame>` is

the frame index in the trajectory.

Command:

```
calc_delta.py --matrix-type BC --reference <frame> --alternatives <frames>
```

This script replaces the, now deprecated, `calc_delta_BC.py` script, which will be removed from MD-TASK in version 2.0 and onwards:

```
calc_delta_BC.py <options> --reference <frame> --alternatives <frames>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Reference frame *	File	`--reference`	Nx1 matrix to be used as the reference (normally the frame from time 0). Delta BC will be worked out by comparing the alternative frames to this one.
Alternative frames *	File/s	`--alternatives`	The remaining Nx1 matrices that should be compared to the reference matrix
Normalize	Boolean	`--normalize`	Set this flag to normalize the values
Normalization mode	Text	`--normalization-mode`	Options are `standard` (ΔBC/BC), `plusone` (ΔBC/(BC+1)), or `nonzero` (ΔBC/BC where BC > 0 else ΔBC) - default mode is `plusone`
Generate plots	Boolean	`--generate-plots`	Set to generate figures

Given a set of BC .dat files `wt_*_bc.dat` (generated with `calc_network.py`), the `wt_0_bc.dat` file could be used as the reference and the rest could be used as the alternatives. If the `wt_0_bc.dat` is renamed to `ref_wt_bc.dat`, the following command could be used:

```
calc_delta_BC.py --generate-plots --normalize --reference ref_wt_bc.dat --
↪alternatives wt_*_bc.dat
```

The above command will generate plots as well as Nx1 matrices representing the difference in BC between each alternative and the reference frame.

Outputs:

Output	Description
ΔBC Matrices	Nx1 matrices representing the change in BC between the reference matrix and each alternative
ΔBC Plots	Figures for each alternative frame, plotting the difference between BC in the alternative and reference

4.5 Calculating Average BC and L (and standard deviation)

The `avg_network.py` script can be used to calculate and plot the average BC and L as well as the standard deviation of these measurements over the course of the trajectory.

Command:

```
avg_network.py <options> --data-type <BC/delta-BC/L/delta-L> --data <matrices>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Data *	File/s	`--data`	The .dat files that will be averaged
Data type *	Text	`--data-type`	Type of data - BC/delta-BC/L/delta-L
Prefix	Text	`--prefix`	Prefix used to name outputs
Generate plots	Boolean	`--generate-plots`	Generate figures/plots
X axis label	Text	`--x-label`	Label for x-axis (use $Delta$ for delta sign)
Y axis label	Text	`--y-label`	Label for y-axis (use $Delta$ for delta sign)
Max Y axis value	Integer	`--y-max`	Maximum value on y-axis
Min Y axis value	Integer	`--y-min`	Minimum value on y-axis
Graph title	Text	`--title`	Title of plot (use $Delta$ for delta sign)
X-axis start value	Integer	`--initial-x`	The start index of the X-axis
Split position	Integer	`--split-pos`	Position to split the network at for large networks. Splits the plot at the given position to create two plots. Useful when analysing a dimer.
Graph title 1	Text	`--title-1`	Title of first plot
Graph title 2	Text	`--title-2`	Title of second plot
X-axis start value 1	Integer	`--initial-x1`	The start index of the x-axis for the first plot
X-axis start value 2	Integer	`--initial-x2`	The start index of the x-axis for the second plot

Given a set of .dat files generated by one of the previous commands (e.g. `wt_*_bc_delta_BC.dat`), the following command could be used:

```
avg_network.py --data wt_*_bc_delta_BC.dat --data-type delta-BC --prefix wt --
↪generate-plots --x-label "Residues" --y-label "Avg delta BC" --title "Wild Type"

avg_network.py --data wt_*_bc_plusone_delta_BC.dat --data-type delta-BC --prefix wt --
↪generate-plots --x-label "Residues" --y-label "Avg delta BC" --title "Wild Type"
```

The above command will generate two new .dat files and a PNG plot. The first .dat file, `wt_delta_bc_avg.dat`, contains an Nx1 matrix with the average ΔBC values for each residue over the course of the simulation. The second .dat file, `wt_delta_bc_std_dev.dat`, contains the standard deviation of ΔBC for each residue over the course of the simulation. The graph plots residues on the X axis and ΔBC on the Y axis. The average values are shown as a line and the standard deviation, representing the fluctuation of ΔBC over the course of the trajectory, are shown as error bars over each residue. *Note that in the above example, we have calculated the average and standard deviation of ΔBC, but avg_network.py can be used with any set of Nx1 matrix (BC/ΔBC/L/ΔL).*

Outputs:

Output	Description
Average .dat file	Nx1 matrix representing the average BC/ΔBC/L/ΔL values from the input matrics
Std dev .dat file	Nx1 matrix representing the standard deviation of the BC/ΔBC/L/ΔL values of the input matrics
Plot	The plotted values from the above matrices

4.6 SNP Analysis - wild-type vs mutant trajectories

Two scripts have been added for comparing BC/ΔBC/L/ΔL graphs. Essentially, all these scripts do is plot the values from different trajectories on the same set of axes. The first script plots two trajectories, a 'reference' and 'alternative' against each other using a normal line graph.

Command:

```
compare_networks.py <options> --reference <reference .dat> --alternative <alternative␣
↪.dat>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Reference .dat file *	File	`--reference`	The reference Nx1 matrix
Alternative .dat file *	File	`--alternative`	The alternative Nx1 matrix
Prefix	Text	`--prefix`	Prefix used to name outputs
Label for reference traj	Text	`--reference-label`	The label that will be used on the plot for the reference matrix
Label for alternative traj	Text	`--alternative-label`	The label that will be used on the plot for the alternative matrix
Y axis label	Text	`--y-label`	Label for y-axis (use $Delta$ for delta sign)
Max Y axis value	Integer	`--y-max`	Maximum value on y-axis
Min Y axis value	Integer	`--y-min`	Minimum value on y-axis

For example, if we had two trajectories, `wt.dcd` and `mutant.dcd`, and we analyzed both trajectories as discussed above, we would end up with 4 files:

- wt_delta_bc_avg.dat (and/or wt_delta_L_avg.dat)
- wt_delta_bc_std_dev.dat (and/or wt_delta_L_std_dev.dat)
- mutant_delta_bc_avg.dat (and/or mutant_delta_L_avg.dat)
- mutant_delta_bc_std_dev.dat (and/or mutant_delta_L_std_dev.dat)

We could compare the above files with the following two commands:

```
compare_networks.py --prefix "wt_mutant_avg" --reference-label Wild-type --
↪alternative-label Mutant --y-label "Delta BC" --reference wt_delta_bc_avg.dat --
↪alternative mutant_delta_bc_avg.dat
compare_networks.py --prefix "wt_mutant_std_dev" --reference-label Wild-type --
↪alternative-label Mutant --y-label "Delta BC" --reference wt_delta_bc_std_dev.dat --
↪alternative mutant_delta_bc_std_dev.dat
```

The output of these commands will provide two figures containing the average ΔBC of the mutant and wild type trajectories plotted against each other for comparison purposes.

Outputs:

Output	Description
Comparison plot	Plot comparing Nx1 matrix of reference .dat file with alternative .dat file

4.7 SNP Analysis - wild-type vs mutants heatmap

Where the above script allows the comparison of two matrices, the second comparison script, `delta_networks.py`, allows the comparison of many trajectories via a heatmap in which the rows represent the trajectories and the columns represent residues.

Command:

```
delta_networks.py <options> --reference <reference avg .dat> --reference-std
↪<reference std dev .dat> --alternatives <alternative avg .dats> --alternatives-std
↪<alternative std dev .dats>
```

Input:

Input (*required*)	Input type	Flag	Description
Reference avg .dat file *	File	`--reference`	The .dat files that will be averaged
Reference std_dev .dat file *	Text	`--reference-type`	Type of data - BC/delta-BC/L/delta-L
Alternatives avg .dat file *	File	`--alternatives`	The .dat files that will be averaged
Alternatives std_dev .dat file *	Text	`--alternatives-type`	Type of data - BC/delta-BC/L/delta-L
Use absolute values	Boolean	`--absolute`	Convert all values on the heatmap to absolute values
Prefix	Text	`--prefix`	Prefix used to name outputs
Graph title	Text	`--title`	Title of plot (use $Delta$ for delta sign)
X axis label	Text	`--x-label`	Label for x-axis (use $Delta$ for delta sign)
Y axis label	Text	`--y-label`	Label for y-axis (use $Delta$ for delta sign)
X-axis start value	Integer	`--initial-x`	The start index of the X-axis
Split position	Integer	`--split-pos`	Position to split the hetamap at for large proteins/complexes. Splits the plot at the given position to create two plots. Useful when analysing a dimer.
Graph title 1	Text	`--title-1`	Title of first plot
Graph title 2	Text	`--title-2`	Title of second plot
X-axis start value 1	Integer	`--initial-x1`	The start index of the x-axis for the first plot
X-axis start value 2	Integer	`--initial-x2`	The start index of the x-axis for the second plot

Given a set of analyzed trajectories, they can be compared to a wild type trajectory using the following command:

```
delta_networks.py --reference wt_delta_BC_avg.dat --reference-std wt_delta_BC_std_dev.
↪dat --alternatives mutant_*_delta_BC_avg.dat --alternatives-std mutant_*_delta_BC_
↪std_dev.dat --absolute --prefix my_protein_delta --title "My Protein" --x-label
↪"Residues" --y-label "Proteins"
```

The above command will produce a PNG with 2 heatmaps for comparing the average and standard deviation Nx1 BC matrices of the wild-type protein with those of the mutated proteins.

Outputs:

Output	Description
Comparison plot	2 heatmaps comparing average and standard deviation values of a wild type protein with a number of mutated proteins

4.8 SNP Analysis - residue contact map

A weighted residue contact map allows the user to determine how often, throughout the trajectory, a residue was interacting with surrounding residues. A contact map can be generated at a position containing a SNP and compared to the same position in the wild type protein to determine whether the SNP affect the immediate interactions at that position.

Command:

```
contact_map.py <options> --trajectory <trajectory> --topology <pdb file>
```

Input:

Input (*required*)	Input type	Flag	Description
Trajectory *	File		A trajectory from a molecular dynamics simulation. Can be in DCD or XTC format.
Topology *	File	`--topology`	A PDB reference file for the trajectory.
Residue	Text	`--residue`	The residue in the trajectory to build the contact map around
Threshold	Float	`--threshold`	Distance threshold in Angstroms when constructing network (default: 6.7).
Prefix	Text	`--prefix`	Prefix used to name outputs

Given two trajectories, `wt.dcd` and `mutant.dcd`, where a mutation, `ASP31ASN`, occurs, the following could be used to build contact maps around position 31 in both trajectories:

```
contact_map.py --residue ASP31 --prefix wt --topology wt.pdb wt.dcd
contact_map.py --residue ASN31 --prefix mutant --topology mutant.pdb mutant.dcd
```

For each of the commands above, a contact map in PDF format will be produced, as well as a CSV file containing the calculated values. The contact maps can be compared visually to give an idea of the changes cause by the mutation.

Outputs:

Output	Description
Contact map	Network with weighted edges depicting how often residues are interacting with the selected residue over the course of the simulation
Contact network (CSV)	Network in CSV format

CHAPTER 5

Pertubation Response Scanning

PRS is a computational technique that is useful for determining single residues that play an active role in the manipulation of protein conformational changes. As input it requires two distinct atomic conformations for a protein of interest; initial and target structures respectively. The technique then performs a residue-by-residue scanning of the initial conformation, by exerting multiple factious external forces of both random direction and magnitude on each residue in the protein structure. After external force perturbation, the subset of residues/forces that invoke a conformational change closest to the target structure are recorded. To calculate the predicted displacement of all residues in relation to a perturbation at a single residue, PRS requires the construction of a variance-covariance matrix, which can be obtained from suitable length MD simulation trajectories of the initial protein structure. The quality of the predicted displacements is then assessed by correlating the predicted and experimental displacements, averaged over all affected residues. This results in a correlation coefficient for each residue in the protein, where a value close to 1 implies good agreement with the experimental change. PRS can thus be used to map regions on a protein whose perturbation leads to a conformational change that resembles the expected target structure. These regions are often active site residues on the protein, but also potentially point to locations involved in allostery and allosteric control. PRS has also been used in conjunction with molecular docking to calculate ligand bound conformations from an unbound structure, in a scheme for exploring protein-ligand interaction.

5.1 Performing PRS

Command:

```
prs.py <options> --final <final.xyz> --trajectory <trajectory> --topology <pdb>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory *	File		A trajectory from a molecular dynamics simulation. Can be in DCD or XTC format.
Topology *	File	`--topology`	A PDB reference file for the trajectory.
Initial	File	`--initial`	Co-ordinate file (.xyz) depicting the initial conformation (default: co-ordinate file is generated from the first frame of the trajectory)
Final *	File	`--final`	Co-ordinate file (.xyz) depicting the target conformation
Perturbations	Integer	`--perturbations`	Number of perturbations to apply
No. of frames in trajectory	Integer	`--num-frames`	Optionally specify the number of frames in the trajectory. This will run the script in a memory efficient mode. Usefult for large trajectories that don't fit into memory.
Step	Integer	`--step`	Step to use when iterating through trajectory frames i.e. how many frames will be skipped.
Prefix	Text	`--prefix`	Prefix used to name outputs

Given a trajectory, `example_small.dcd`, with initial and target co-odinate files, `initial.xyz` and `final.xyz`, respectively, and topology file, `example_small.pdb`, the following command could be used:

```
prs.py --initial initial.xyz --final final.xyz --perturbations 100 --step 100 --
↪prefix result --topology example_small.pdb example_small.dcd
```

Outputs:

Output	Description
Correlation CSV file	Correlation coefficient for each residue in the protein, where a value close to 1 implies good agreement with the experimental change

CHAPTER 6

Dynamic Cross-Correlation

Molecular Dynamics (MD) is a computational method that analyses the physical motions of atoms within a protein or protein complex. In a given system, the interactions between the atoms can be simulated in the presence of a force field and, following the application of Newtons' equations of motion, trajectories corresponding to the dynamical motions of the atoms are obtained. The trajectories represent sequential snapshots of the system, by presenting the atomic coordinates at specific time intervals throughout the simulation. This allows for the investigation into the dynamical changes of the system over time. The applications of MD simulations are vast. By analysing the trajectories of the system, it is possible to calculate the dynamic correlation between all atoms within the molecule i.e. the degree to which they move together. This dynamic cross-correlation tool produces an NxN heatmap, where N = the number of (alpha carbon) atoms in the system, and each element corresponds to the dynamic cross-correlation between each i,j atom. The correlation values are calculated between -1 and 1, where 1=complete correlation; -1=complete anti-correlation; 0= no correlation.

6.1 Calculating dynamic cross-correlation

Command:

```
calc_correlation.py <options> --trajectory <trajectory> --topology <pdb>
```

Inputs:

Input (**required*)	Input type	Flag	Description
Trajectory *	File		A trajectory from a molecular dynamics simulation. Can be in DCD or XTC format.
Topology *	File	`--topology`	A PDB reference file for the trajectory.
Step	Integer	`--step`	Step to use when iterating through trajectory frames i.e. how many frames will be skipped.
Prefix	Text	`--prefix`	Prefix used to name outputs.
Lazy load	Boolean	`--lazy-load`	Load trajectory frames in a memory efficient manner - use for large trajectories.

Given a trajectory, `example_small.dcd`, and topology file, `example_small.pdb`, the following command could be used:

```
calc_correlation.py --step 100 --prefix example_corr --trajectory example_small.dcd --
↪topology example_small.pdb --lazy-load
```

Outputs:

Output	Description
Correlation heatmap	PNG heatmap depicting the dynamic correlation between atoms in the trajectory
Correlation text file	Correlation data in text format