taurenmd

Release 0.11.2

Joao MC Teixeira

Jul 14, 2022

CONTENTS

A command-line interface for analysis routines of Molecular Dynamics data.

Taurenmd provides an easy, flexible and extensible, **command-line** interface for the most common *(and not so common)* routines of analysis and representation of Molecular Dynamics (MD) data.

It bridges the gap between the highly complex (and powerful) Python libraries available for analysis of MD data and the *non-developer* users that lack the programming skills to perform a thorough and proficient use those libraries. *But not only*, **taurenmd** also facilitates high throughput operations, even to those proficient *devs*, because complex executions are reduced to single argument-rich command-lines that can be concatenated or aliased.

Taurenmd wraps around feature-rich and powerful MD analysis libraries such as MDAnalysis and MDTraj *(but not only)*, combining them to extract the best of *those worlds*. We use these libraries to access and extract MD data and calculate observables, and we have also added our own routines of analysis when needed. When using this software, you **should** cite **taurenmd** together with the dependencies used, please read our Citing page for a detailed explanation.

Though designed to perform as a command-line user-directed interface, all **taurenmd**'s core functions are openly distributed and documented. Currently, there are already several command-line interfaces available, some that perform only single tasks, while others allow complex setups, all are *one-liners*.

With this said, **taurenmd** aims to be a flexible and extensible peace of software, built as simple and modular as we can think of, to *agile* the incorporation of new functionalities as needed.

You can read now through the **contents** bellow.

CHAPTER

ONE

CONTENTS

1.1 Installation

taurenmd is written in, and depends on projects written in, Python; therefore, its installation process is based on the Python installation routines and related community-available tools. Find *taurenmd*:

1. package at PyPI
2. GitHub source repository

1.1.1 Supported Platforms

taurenmd is designed to run natively under any platform compatible with Python (paths are not hard coded `;-)`). However, **the libraries taurenmd depends on may or may not be compatible with all OS platforms**, and we are **not** responsible for providing compatibility or support for such libraries. To be able to exploit all its features, you should choose a platform compatible with all the required Molecular Dynamics analysis libraries used by *taurenmd*. *At the bottom of this page we have a section that describes taurenmd's dependencies*.

We can **guarantee** *taurenmd* works fully with all its dependencies using Anaconda on Ubuntu 18.04 LTS, and we are *positive* (not sure) it will be the same for any system supporting Anaconda.

1.1.2 Installation steps

From a previous defined environment

If you use Molecular Dynamics for your research, odds are you have already installed the *required dependencies*; if this is the case, you can just install *taurenmd* on top of them, run: `pip install taurenmd` in your MD analysis Python environment.

From scratch

To install *taurenmd* from scratch:

With Anaconda

If you use Anaconda as your Python package manager just do the following on your `terminal`:

1. Download the *taurenmd* Anaconda environment file from our repository:

```
curl https://raw.githubusercontent.com/joaomcteixeira/taurenmd/master/requirements.
↪yml -o taurenmdenv.yml
```

If for some reason the above does not work, just open the link on your WebBrowser and save the text to a file (or save the file).

2. Create a new Anaconda Python environment to host *taurenmd*. Choose the python version that best fits your needs; if you are not sure chose 3.7.:

```
conda create -n taurenmd python=3.8
conda env update -n taurenmd --file taurenmddev.yml
```

Where `taurenmdenv.yml` is the file downloaded in the previous step.

3. Activate the newly created environment:

```
conda activate taurenmd
```

4. You are ready, type:

```
taurenmd
```

to start using `taurenmd`.

With PyPI

If you do not use Anaconda and you actually rely on PyPI as your package manager, that is also (almost) perfectly fine.

1. Create a new Python environment if you wish following the official instructions for your running Python version. We do not provide specific commands for these operations because these change with certain frequency, so it is best to refer to the official sources.

2. Install *taurenmd*:

```
python -m pip install --upgrade pip wheel
pip3 install taurenmd[all]
```

3. You should be good to go

Note. What is the problem with the pure PyPI installation?

taurenmd relies on OpenMM to read `.cif` topology files when using routines based on `MDTraj`, and OpenMM is not deployed on PyPI and requires installation through its conda channel. Therefore, unless you need to load `.cif` files you can use *taurenmd* from a pure PyPI installation. Otherwise, you should follow the *With Anaconda* instructions. *May be you want to help us out solving this problem :-)*.

Other Platforms

We do not provide support for other distribution platforms such as HomeBrew or Chocolatey, but may be you can emulate the steps described above for these systems. Feel welcomed to *improve this documentation with your insights*!

User installation suggestions for particular systems:

1. pyenv in Arch Linux
2. on zsh

From GitHub

If you are a proficient Pythonista you might want to install **taurenmd** directly from the GitHub repository. If that is the case you might not need to read this section because you already know well what to do; nonetheless, let's go through it:

Note: `taurenmd` follows *Semantic Version 2.0*, meaning that every single new addition to the master branch gets released on PyPI with a new version number. Therefore, installing from the `master` GitHub branch actually adds no benefit to installing with `pip`.

---

1. Clone our repository: `git clone https://github.com/joaomcteixeira/taurenmd`
2. Place yourself in the new `taurenmd` folder, in Linux-like systems: `cd taurenmd`.
3. Install the dependencies using Anaconda. Choose your preferred python version.:

```
conda create -n taurenmd python=3.8
conda env update -n taurenmd --file requirements-dev.yml
```

4. Install **taurenmd** with the following command: `python setup.py develop --no-deps`
5. In the future, to keep your installation up to the latest:

1. pull repository updates from the upstream repository: `git pull` (from within `taurenmd` git folder)
2. because taurenmd developments are mostly reflected on new interfaces you need to update those as well: `python setup.py develop --no-deps`
3. beaware, if the version increment denotes API breaks you might need to reinstall `taurenmd` from scratch.

1.1.3 Running taurenmd

After installation you can run *taurenmd* with the following command `:-)`:

```
taurenmd
```

Please read our *Usage* page for, *whatelse*, usage instructions and examples.

1.1.4 Upgrade

To upgrade *taurenmd* and all its dependencies to the latest version:

1. If you installed from PyPI:

```
pip3 install -U --force-reinstall taurenmd
```

2. If you installed from Anaconda:

```
pip3 install -U --force-reinstall taurenmd --no-deps
```

1.1.5 Something failed

In case something is failing during installation, execution or upgrade, please write us an Issue explaining your situation.

1.1.6 How taurenmd manages its dependencies

By default, installing `taurenmd` does **not** install **all** its dependencies. **Why?** Because *taurenmd* relies on large and complex libraries required to manage the Molecular Dynamics (MD) data, such as MDAnalysis and MDTraj, and installing them automatically might not be the optimal solution for every case, for example:

1. Many MD researchers may actually work on:
   - cutting edge *development* versions,
   - forked versions,
   - source-compiled versions
2. There may be platform compatibility issues (read further),
3. Lastly and minor, not all dependencies are required for every *taurenmd command*,

So installing those libraries by default together with *taurenmd* might be counter productive[1].

Nonetheless, *taurenmd* does provide an easy way to install this dependencies whenever possible and needed. These details are explained in the *Installation steps* section above.

The dependencies that are kept separate from the default installation process are listed bellow; here, links point to their respective official installation instructions.

1. MDAnalysis Installation instructions
2. MDTraj installation instructions
3. OpenMM installation
4. Numpy, is installed together with the above dependencies, so you should not need to reinstall it again, just stick to the version compatible with the 3 libraries, this should be managed automatically by your Python package manager. Nonetheless, and for your interest, **taurenmd** requires *Numpy* but it is not installed along with the main installation.

[1] Dependency installation could be disabled using the `--no-deps` flag of `pip`, but we decided for the other strategy.

Other dependencies installed automatically

Other dependencies that are indeed automatically installed alongside with *taurenmd* are listed bellow:

1. python-bioplottemplates
2. pyquaterion

1.2 Usage

Taurenmd provides a command-line interface to many routines in Molecular Dynamics data analysis, therefore **taurenmd** runs by executing one-line and argument-rich commands on the `terminal`.

Taurenmd uses other scientific libraries to handle and generate Molecular Dynamics data. You SHOULD by all means cite also the other libraries when using taurenmd. Please refer to our *Citing* section for more detailed instructions.

We have several *command* interfaces already implemented, our *Command-line interfaces* page documents in detail each and every of them.

IMPORTANT

Do not forget to activate the python environment where you installed *taurenmd* in case it is not yet activated. Please read through our *Installation* page.

taurenmd main client interface.

1.2.1 Usage Examples

To access to the complete list of *taurenmd commands* with a summary information for each, execute:

```
>>> taurenmd -h
```

or simply:

```
>>> taurenmd
```

To see the current version number:

```
>>> taurenmd -v
```

Using `trajedit` as an example, lets inspect its functionality:

```
>>> taurenmd trajedit -h
```

With `trajedit` you can edit a trajectory in many different ways. For example, convert a trajectory to another format:

```
>>> taurenmd trajedit topology.pdb trajectory.xtc -d new_trajectory.dcd
```

The above command reads the original `trajectory.xtc` file and outputs the new `new_trajectory.dcd`. You can also use `trajedit` to reduce the trajectory size, say by slicing every 10 frames:

```
>>> taurenmd trajedit topology.pdb trajetory.xtc -d traj_p10.xtc -p 10
```

the `-p` option refers to the slicing step size, in this case `10` - reads every 10 frames. Likewise, you can pass a *start* (`-s`) and an *end* (`-e`) arguments:

```
>>> taurenmd trajedit topology.pdb trajectory.xtc -d traj_s50_e500_p10.xtc -s 50 -e 500 -
↪p 10
```

Also, you can extract an Atom Selection from a trajectory to a new trajectory file. The example bellow creates a new trajectory from the input one containing only atoms belonging to chain A. In cases like this it is useful to extract the atom selection as an independent topology file.

```
>>> taurenmd trajedit top.pdb traj.xtc -l 'segid A' -d chainA.xtc -o chainA_topology.pdb
```

You can also use `trajedit` to extract a specific frame from a trajectory:

```
>>> taurenmd trajedit topology.pdb trajectory.xtc -d frame40.pdb -s 40 -e 41
```

but, for this example, you could instead use the `fext` interface:

```
>>> taurenmd fext topology.pdb trajectory.xtc -f 40 -x .pdb -f frame_
```

Each an every `taurenmd` sub command is available directly as a main routine by prefixing a `tmd` to its name, for example:

```
>>> taurenmd trajedit
>>> # equals to
>>> tmdtrajedit
```

1.2.2 Logging

taurenmd logs all its running activity as follows:

1. `.taurenmd.cmd`, keeps an historic register of the taurenmd commands run on a given folder together with a list of the research projects that must be cited for that particular run; these are the libraries taurenmd used to access and process the MD data. It also servers as a record for your research project.

2. `.taurenmd.log`, a user readable logging information, the very same that is printed in the `terminal` during runtime. Overwrites previous runs.

3. `.taurenmd.debug`, a full verbose log with all runtime information for the LAST run. Overwrites previous runs.

1.3 Command-line interfaces

This page documents all command-line client interfaces available in `taurenmd`, referred also as *taurenmd subroutines*. You may wish to read before the *Usage* page for general examples on how to use `taurenmd` for different purposes.

On the `terminal`, you can access a list of all *taurenmd subroutines* by running:

```
taurenmd
```

You can then access the individual help for each subroutine as follows, for example:

```
taurenmd dist -h
```

where, `-h` is optional.

The same help messages can be found in this documentation in the links provided bellow.

1.3.1 taurenmd subroutines

This TOC lists all *taurenmd subroutines*, click on each one to read on specific usage examples, and technical documentation.

Client Distances

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Calculates distances between centers of geometry of two selections.

Distance is given in 3D XYZ coordinate space units.

Algorithm

Distance between centers of geometry is calculated by::

```
np.linalg.norm(np.subtract(coord1, coord2))
```

Where, `coord*` are the centers of geometry of each atom selection `-l1` and `-l2`, respectively. Read further on np.linalg.norm and np.subtract.

Examples

Calculate the distances between two carbon alphas:

```
taurenmd dist top.pdb traj.dcd -l1 'resnum 10 and name CA' -l2 'resnum 20 and name CA'
```

Calculate the distances between two chains:

```
taurenmd dist top.pdb traj.dcd -l1 'segid A' -l2 'segid B'
```

`-x` exports the data to a CSV file. You can also plot the data with the `-v` option:

```
[...] -x distances.csv -v title=my-plot-title xlabel=frames ylabel=degrees ...
```

where `[...]` is the previous command example.

`dist` can be run directly as main command instead of subroutine:

```
tmddist
```

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmddist [-h] [-v] [-i] [-l1 SEL1] [-l2 [SEL2 [SEL2 ...]]] [-s START]
               [-e STOP] [-p STEP] [-x [EXPORT]] [--plot [PLOT [PLOT ...]]]
               topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version	show program's version number and exit
-i, --insort	Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`. Default: False
-l1, --sel1	First selection (a single selection). Default: "all"
-l2, --sel2	Second selections. As many as desired. For example: –sel2 "segid A" "segid B and name CA". Default: "all"
-s, --start	The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.
-e, --stop	The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.
-p, --step	The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.
-x, --export	Export calculated values to a CSV file. Defaults to 'results.csv', alternatively, you can give a specific file name. Default: False

--plot Plot results. Additional arguments can be given to configure the plot style. Example: –plot xlabel=frames ylabel=RMSD color=red.Accepted plot arguments are defined by the function used to plot the result. The main description of this client which plotting function is used. Defaults to `None`, no plot is produced.

Default: False

Client Frame Extract

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Extract trajectory frames to individual files.

Normally used to extract frames to PDB topology files so those can be inspected independently.

Note

```
Frame number is 0-indexed.
```

Examples:

Extract frames 11 to 49 (inclusive), remember frames index start at 0:

```
taurenmd fext topology.pdb trajectory.dcd -s 10 -e 50
```

Extract the first frame:

```
taurenmd fext topology.pdb trajectory.dcd -flist 0
```

Extract a selection of frames:

```
taurenmd fext topology.pdb trajectory.dcd -flist 0,10,23,345
```

Frame file types can be specified:

```
taurenmd fext topology.pdb trajectory.dcd -p 10 -x .dcd
```

Atom selection can be specified as well, the following extracts only the 'segid A' atom region of the first frame. Selection rules are as decribed for MDAnalysis selection.

```
taurenmd fext topology.pdb trajectory.xtc -flist 0 -l 'segid A'
```

Multiple trajectories can be given, they will be contatenated:

```
taurenmd fext top.pdb traj1.xtc traj2.xtc traj3.xtc -p 10
```

Can also be used as main command:

```
tmdfext topology.pdb ...
```

References:

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmdfext [-h] [-v] [-i] [-l SELECTION] [-t FLIST [FLIST ...]] [-s START]
               [-e STOP] [-p STEP] [-f PREFIX] [-x EXT] [--odir ODIR]
               topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

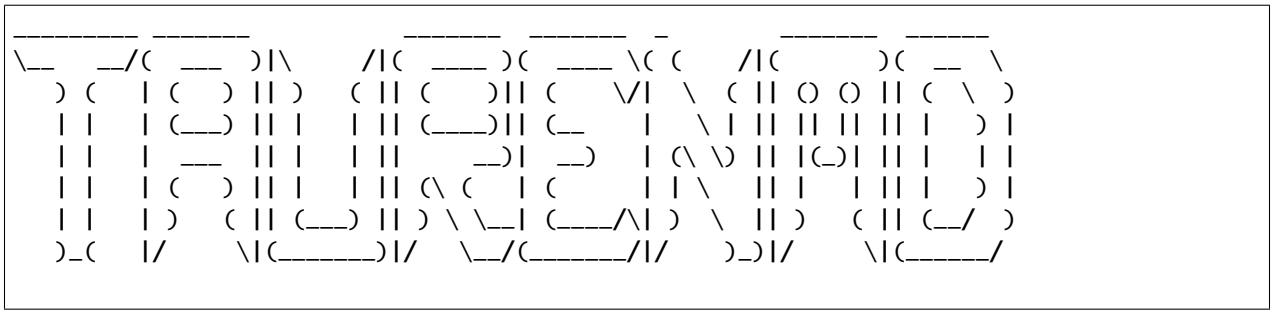
Named Arguments

-v, --version	show program's version number and exit
-i, --insort	Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`. Default: False
-l, --selection	Atom selection for the output trajectory. Selection rules are as defined by the MD analysis library used by the client interface. For instructions read the main command-line client description. Defaults to 'all'. Default: "all"
-t, --flist	List of frames (time steps) to consider.If applicable, this can used instead of the start, stop and step slicing arguments. Default: False
-s, --start	The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.

-e, --stop	The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.
-p, --step	The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.
-f, --prefix	String prefix for each file. Defaults to `frame_`. Default: "frame_"
-x, --ext	Extension of frame files. Defaulst to .pdb Default: ".pdb"
--odir	Save output to directory. Creates directory if doesn't exist. Defaults to the current working directory. Default: /home/docs/checkouts/readthedocs.org/user_builds/taurenmd/checkouts/latest/docs

Client Image Molecule with MDTraj

Welcome to



A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Make molecules whole.

Attempts to "Recenter and apply periodic boundary conditions to the molecules in each frame of the trajectory." (MDTraj documentation)

Algorithm

Uses MDTraj.Trajectory.image_molecule and MDTraj.Topology.find_molecules.

Protocol 1

Performs `mdtraj.top.find_molecules` and `mdtraj.traj.image_molecules` in the trajectory as a whole. `anchor_molecules` parameters get `mdtraj.top.find_molecules[:1]`, and `other_molecules` parameter receives `mdtraj.top.find_molecules[1:]`.

Protocol 2

The same as protocol 1 but executes those steps for each frame separately. Frames are concatenated back to a whole trajectory at the end.

Examples

Basic usage, `-o` saves the first frame in a separate topology file:

```
taurenmd imagemol top.pdb traj.dcd -d imaged.dcd -o
```

For trajectories with *non-standard* molecules you can use a TPR file.

```
taurenmd imagemol top.tpr traj.xtc -d imaged.xtc
```

Using protocol 2

```
taurenmd imagemol top.tpr traj.xtc -d imaged.xtc -i 2
```

References

- MD data accessed and/or processed using MDTraj

```
usage: tmdimagemol [-h] [-v] [-i] [-d TRAJ_OUTPUT] [-o [TOP_OUTPUT]]
                   [--protocol PROTOCOL]
                   topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version show program's version number and exit

-i, --insort Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`.

Default: False

-d, --traj-output Modified trajectory output file name. File type will be defined by file name extension. Defaults to traj_out.dcd.

Default: "traj_out.dcd"

-o, --top-output Export edited trajectory first frame as topololy file. You can specify the exact file name, otherwise, defaults to input trajectory path + *'frame0.pdb'. Also, if name starts with '*', it is used as file suffix, if name ends with '_', it is used as prefix, instead.

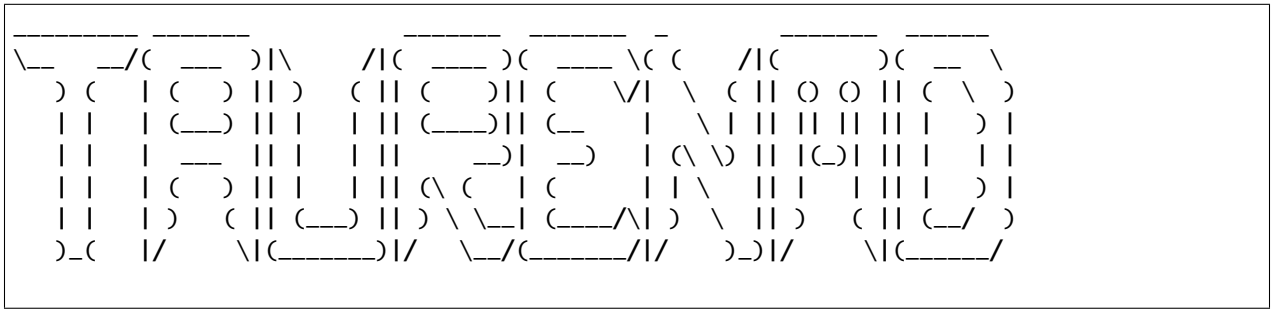
Default: False

--protocol The protocol with which reimage. Read main command description for details.

Default: 1

Client No Solvent

Welcome to



A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Remove solvent from trajectory.

Algorithm

Removes solvent from trajectory using MDTraj.remove_solvent.

Examples

Remove all solvent:

```
taurenmd nosol top.pdb traj.dcd -d traj_nosol.dcd -o
```

Remove all solvent except for NA atoms:

```
taurenmd nosol top.pdb traj.dcd -d traj_nosolNA.dcd -e Na -o
```

`tmdnosol` can be used as main command:

```
tmdnosol [...]
```

References

- MD data accessed and/or processed using MDTraj

```
usage: tmdnosol [-h] [-v] [-i] [-d TRAJ_OUTPUT] [-o [TOP_OUTPUT]]
                [-m MAINTAIN [MAINTAIN ...]]
                topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version	show program's version number and exit
-i, --insort	Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`. Default: False
-d, --traj-output	Modified trajectory output file name. File type will be defined by file name extension. Defaults to traj_out.dcd. Default: "traj_out.dcd"
-o, --top-output	Export edited trajectory first frame as topololy file. You can specify the exact file name, otherwise, defaults to input trajectory path + *'frame0.pdb'. Also, if name starts with* '', it is used as file suffix, if name ends with '_', it is used as prefix, instead. Default: False
-m, --maintain	List of solvent residue names to maintain in trajectory. Feeds MDTraj.Trajectory.remove_solvent.exclude parameter.

Client Plane Angular Oscillations

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Calculate angular oscillation of a plane along the trajectory.

A plane is defined by the centers of geometry of three atom selection groups. The angle between that plane in each frame and itself in the reference frame is computed. Angle can be reported in degrees (default) or radians.

Algorithm

Plane equation is computed by libcalc.calc_plane_eq. Angle between planes is computed by libcalc.calc_planes_angle. Refer to our documentation page for more details.

Examples

Given a protein of 3 subunits (chains or segids) calculate the angle variation of a plane that crosses the protein longitudinally:

```
taurenmd pangle top.pdb traj.xtc -z 'segid A' 'segid B' 'segid C' -x
```

`-x` exports the data to a CSV file. You can also plot the data with the `-v` option:

```
[...] -v title=my-plot-title xlabel=frames ylabel=degrees ...
```

where `[...]` is the previous command example.

`pangle` can be run directly as main command instead of subroutine:

```
tmdpangle
```

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmdpangle [-h] [-v] [-i] -z PLANE_SELECTION PLANE_SELECTION
                 PLANE_SELECTION [-a {degrees,radians}] [-r REF_FRAME]
                 [-s START] [-e STOP] [-p STEP] [-x [EXPORT]]
                 [--plot [PLOT [PLOT ...]]]
                 topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version	show program's version number and exit
-i, --insort	Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`. Default: False
-z, --plane-selection	Three selection strings representing three atom regions. The plane is defined by the three centres of geometry of the three selections. For example: -z 'segid A' 'segid B' 'segid C'.
-a, --aunit	Possible choices: degrees, radians Angular unit, either degrees or radians. Default: "degrees"
-r, --ref-frame	The frame in the trajectory that serves as reference to compute against.Defaults to 0. Default: 0
-s, --start	The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.
-e, --stop	The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.
-p, --step	The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.
-x, --export	Export calculated values to a CSV file. Defaults to 'results.csv', alternatively, you can give a specific file name. Default: False

--plot Plot results. Additional arguments can be given to configure the plot style. Example: –plot xlabel=frames ylabel=RMSD color=red.Accepted plot arguments are defined by the function used to plot the result. The main description of this client which plotting function is used. Defaults to `None`, no plot is produced.

Default: False

Client Report

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Report on trajectory characteristics.

Example

```
taurenmd report topology.pdb trajectory.dcd
```

References

- MD data accessed using MDAnalysis.

```
usage: tmdreport [-h] [-v] [-i] topology trajectories [trajectories ...]
```

Positional Arguments

topology Path to the topology file.

trajectories Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version show program's version number and exit

-i, --insort Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`.

Default: False

Client RMSD

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Calculates RMSDs for a selection.

Algorithm

Calculates the RMSD values along a trajectory slice for different selections. If multiple selections are given creates a series data for that selection.

RMSD is calculated using libcalc.mda_rmsd.

Examples

Calculate RMSD of the whole system:

```
taurenmd rmsd top.pdb traj.dcd -e rmsd.csv
```

Calculates RMSDs for different selections:

```
taurenmd rmsd top.pdb traj.dcd -g 'segid A' 'segid B' -e
```

`-x` exports the data to a CSV file. You can also plot the data with the `-v` option:

```
[...] -x rmsd.csv -v title=my-plot-title xlabel=frames ylabel=RMSDs ...
```

where `[...]` is the previous command example.

You can also use `tmdrmsd` instead of `taurenmd rmsd`.

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmdrmsd [-h] [-v] [-i] [-g SELECTIONS [SELECTIONS ...]] [-r REF_FRAME]
               [-s START] [-e STOP] [-p STEP] [-x [EXPORT]]
               [--plot [PLOT [PLOT ...]]]
               topology trajectories [trajectories ...]
```

Positional Arguments

topology Path to the topology file.

trajectories Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version show program's version number and exit

-i, --insort Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`.

Default: False

-g, --selections List of atom selections to operate with. Selection rules are as defined by the MD analysis library used by the client interface. For instructions read the main command-line client description. Defaults to None, uses a single selection considering all atoms. Example: -g 'segid A' 'segid B' 'name CA'

-r, --ref-frame The frame in the trajectory that serves as reference to compute against.Defaults to 0.

Default: 0

-s, --start The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.

-e, --stop The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.

-p, --step The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.

-x, --export Export calculated values to a CSV file. Defaults to 'results.csv', alternatively, you can give a specific file name.

Default: False

--plot Plot results. Additional arguments can be given to configure the plot style. Example: –plot xlabel=frames ylabel=RMSD color=red.Accepted plot arguments are defined by the function used to plot the result. The main description of this client which plotting function is used. Defaults to `None`, no plot is produced.

Default: False

Client RMSF

Welcome to

```
_________ _______           _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Calculate RMSFS of a selection along the trajectory slice.

Algorithm

Calculates the RMSF values along a trajectory slice for different selections. If multiple selections are given creates a series data for that selection.

RMSF is calculated using libcalc.mda_rmsf.

If multiple selections are given, separate calculations are performed in sequence. Result files (data tables and plots) are exported separately for each selection. Selections can't be overlayed easily in a single plot because they do not share the same labels.

Examples

Calculate RMSF of the whole system:

```
taurenmd rmsf top.pdb traj.dcd -e rmsf.csv
```

Calculates RMSFs for different selections:

```
taurenmd rmsf top.pdb traj.dcd -g 'segid A' 'segid B' -e
```

`-x` exports the data to a CSV file. You can also plot the data with the `-v` option:

```
[...] -x rmsf.csv -v title=my-plot-title xlabel=frames ylabel=RMSFs ...
```

where `[...]` is the previous command example.

you can also use `tmdrmsf` instead of `taurenmd rmsf`.

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmdrmsf [-h] [-v] [-i] [-g SELECTIONS [SELECTIONS ...]]
               [--inverted-selections INVERTED_SELECTIONS [INVERTED_SELECTIONS ...]]
               [-s START] [-e STOP] [-p STEP] [-x [EXPORT]]
               [--plot [PLOT [PLOT ...]]]
               topology trajectories [trajectories ...]
```

Positional Arguments

topology Path to the topology file.

trajectories Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version show program's version number and exit

-i, --insort Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`.

Default: False

-g, --selections List of atom selections to operate with. Selection rules are as defined by the MD analysis library used by the client interface. For instructions read the main command-line client description. Defaults to None, uses a single selection considering all atoms. Example: -g 'segid A' 'segid B' 'name CA'

--inverted-selections Invert order of selections.

-s, --start The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.

-e, --stop The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.

-p, --step The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.

-x, --export Export calculated values to a CSV file. Defaults to 'results.csv', alternatively, you can give a specific file name.

Default: False

--plot Plot results. Additional arguments can be given to configure the plot style. Example: –plot xlabel=frames ylabel=RMSD color=red.Accepted plot arguments are defined by the function used to plot the result. The main description of this client which plotting function is used. Defaults to `None`, no plot is produced.

Default: False

Client Rotations

The image bellow complements the textual explanation of the *roll*, *pitch*, and *yaw* angles, and how we calculate them.

Welcome to

```
_________ _______          _______  _______  _        _______  ______
\__   __/(  ___  )|\     /|(  ____ )(  ____ \( (    /|(       )(  __  \
   ) (   | (   ) || )   ( || (    )|| (    \/|  \  ( || () () || (  \  )
   | |   | (___) || |   | || (____)|| (__    |   \ | || || || || |   ) |
   | |   |  ___  || |   | ||     __)|  __)   | (\ \) || |(_)| || |   | |
   | |   | (   ) || |   | || (\ (   | (      | | \   || |   | || |   ) |
   | |   | )   ( || (___) || ) \ \__| (____/\| )  \  || )   ( || (__/  )
   )_(   |/     \|(_______)|/   \__/(_______/|/    )_)|/     \|(______/
```

A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Decompose Eurler angle rotations for a selection.

Calculate the Roll, Pitch and Yaw angles of a plane along the trajectory.

Read further on roll, pitch and yaw angles (Euler Angles) - wikipedia.

Here we decompose these movements around the three different axis centered at an origin.

Calculation algorithm

Given a selection of three regions, `selection A`, `selection B` and `selection C`:

1. Center the system to the three selections center of geometry for every frame, this is called the *origin*;
2. Calculate a plane given by the center of geometries of the three selections, plane ABC;
3. Define the vector OA that goes from the origin to the center of geometry of `selection A`, this represents the Yaw axis;
4. Define the normal vector to the plane ABC (ABCn), this represents the Roll axis;
5. Define the cross product betwee vectors OA and ABCn (AONn), this is the Pitch axis;
6. Repeat this process for every frame.

Calculating the angles:

Angles represent the right hand rotation around an axis of the sistem in a i-frame compared to the reference frame.

Roll

The roll angle is given by the torsion angle between OA, origin, ABCn, and OAi (in frame), displaced along ABCn.

Pitch

The pitch angle is by the torsion angle between ABCn, origin, AONn, and ABCni (in frame), displaced along AONn.

Yaw

The yaw angle is given by the torsion angle between the AONn, origin, OA, and AONni (in frame), displaced along OA.

Examples

In the case of an homotrimer, define the axis and the origin on the trimers:

```
taurenmd rotations -z 'segid A' 'segid B' 'segid C' -x rotations.csv
```

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.

```
usage: tmdrotations [-h] [-v] [-i] -z PLANE_SELECTION PLANE_SELECTION
                    PLANE_SELECTION [-a {degrees,radians}] [-r REF_FRAME]
                    [-s START] [-e STOP] [-p STEP] [-x [EXPORT]]
                    [--plot [PLOT [PLOT ...]]]
                    topology trajectories [trajectories ...]
```

Positional Arguments

topology	Path to the topology file.
trajectories	Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version	show program's version number and exit
-i, --insort	Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`. Default: False
-z, --plane-selection	Three selection strings representing three atom regions. The plane is defined by the three centres of geometry of the three selections. For example: -z 'segid A' 'segid B' 'segid C'.

-a, --aunit Possible choices: degrees, radians

Angular unit, either degrees or radians.

Default: “degrees”

-r, --ref-frame The frame in the trajectory that serves as reference to compute against.Defaults to 0.

Default: 0

-s, --start The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.

-e, --stop The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.

-p, --step The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.

-x, --export Export calculated values to a CSV file. Defaults to ‘results.csv’, alternatively, you can give a specific file name.

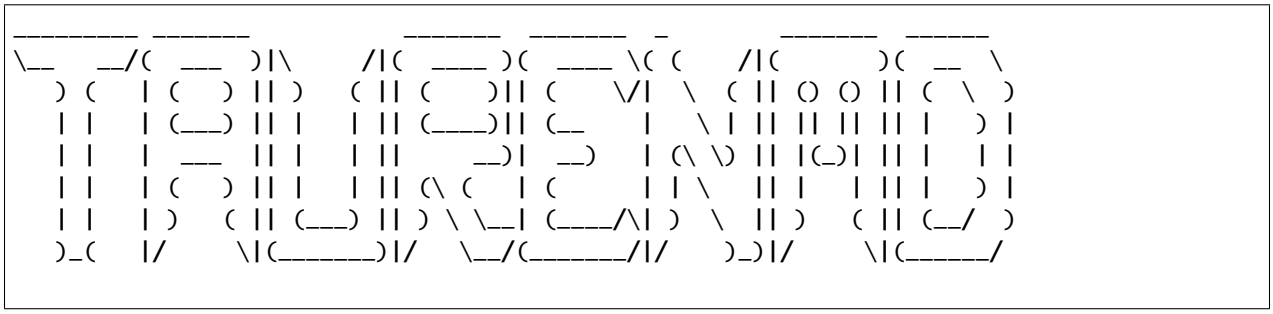
Default: False

--plot Plot results. Additional arguments can be given to configure the plot style. Example: –plot xlabel=frames ylabel=RMSD color=red.Accepted plot arguments are defined by the function used to plot the result. The main description of this client which plotting function is used. Defaults to `None`, no plot is produced.

Default: False

Client Traj Edit

Welcome to



A command-line interface for Molecular Dynamics Analysis routines.

version: 0.11.2

Edit a trajectory.

In short, takes a trajectory and apply a modification:

1. frame slicing
2. atom selection
3. unwrap
4. align
5. file format change

Saves the result to a new trajectory.

All operations are performed with `MDAnalysis <https://www.mdanalysis.org>_`.

Examples

Changes trajectory file format

```
taurenmd trajedit top.pdb traj.xtd -d traj.dcd
```

Extracts a part of the trajectory atoms, in this example `segid A`, the option `-o` saves the first frame of the new trajectory to a topology file:

```
taurenmd trajedit top.pdb traj.xtc -d tsegidA.dcd -o -l "segid A"
```

You can slice the trajectory by appending the following `-s`, `-e` or `-p` options, this saves only every 100 frames:

```
[...] -p 100
```

You can align the trajectory to a part of the system, for example, align the whole system to one of its subunits:

```
taurenmd trajedit top.pdb traj.dcd -d alignedA.dcd -a "segid A and name CA"
```

further restrain the output to a specific subselection with `-l`:

```
[...] -l "segid A or segid B"
```

`trajedit` also implements the `unwrap` method from which is an alternative approach to the `imagemol` client, that implements from `MDTraj`. See references section.

```
taurenmd trajedit top.pdb traj.dcd -d unwrapped.dcd -w -o unwrapped_frame0.pdb
```

References

- MD data accessed using MDAnalysis.
- selection commands follow MDAnalysis selection nomenclature.
- unwrap performed by MDAnalysis unwrap.
- align performed by MDAnalysis unwrap.

```
usage: tmdtrajedit [-h] [-v] [-i] [-l SELECTION] [-s START] [-e STOP]
                   [-p STEP] [-d TRAJ_OUTPUT] [-o [TOP_OUTPUT]] [-a [ALIGN]]
                   [-w] [--unwrap-reference UNWRAP_REFERENCE]
                   [--unwrap-compound UNWRAP_COMPOUND]
                   topology trajectories [trajectories ...]
```

Positional Arguments

topology Path to the topology file.

trajectories Path to the trajectory files. If multiple files are given, trajectories will be concatenated by input order.

Named Arguments

-v, --version show program's version number and exit

-i, --insort Sorts input trajectories paths according to their tail numbers, if paths are formatted as follows: my_trajectory_#.dcd, where # is a number. Defaults to `False`.

Default: False

-l, --selection Atom selection for the output trajectory. Selection rules are as defined by the MD analysis library used by the client interface. For instructions read the main command-line client description. Defaults to 'all'.

Default: "all"

-s, --start The starting index for the frame slicing. Frames are 0-indexed, so the first frame is -s 0. The starting index is inclusive. Defaults to None, considers from the beginning.

-e, --stop The ending index for the frame slicing. Frames are 0-indexed, so the last frame of a 500 frame trajectory is index 499, but because the ending index is exclusive, -e 500 is required. Defaults to None, considers to the end.

-p, --step The periodicity step value for the frame slicing, -p 10 means every 10 frames. Defaults to None, considers every 1 frame.

-d, --traj-output Modified trajectory output file name. File type will be defined by file name extension. Defaults to traj_out.dcd.

Default: "traj_out.dcd"

-o, --top-output Export edited trajectory first frame as topololy file. You can specify the exact file name, otherwise, defaults to input trajectory path + '*frame0.pdb'. Also, if name starts with* '', it is used as file suffix, if name ends with '_', it is used as prefix, instead.

Default: False

-a, --align Align system to a atom group. Alignmed RMSD is compared to the Topology coordinates. Uses MDAnalysis.analysis.align.alignto. If given without argument defaults to 'all'. Defaults to `False`.

Default: False

-w, --unwrap Unwraps selection according to: https://www.mdanalysis.org/docs/documentation_pages/core/groups.html

Default: False

--unwrap-reference The unwrap method reference parameter. Has effect only if '-w' is given. Defaults to `None`.

--unwrap-compound The unwrap method compound parameter. Has effect only if '-w' is given. Defaults to `fragments`.

Default: "fragments"

1.4 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute from the scope of an user or as a core Python developer.

1.4.1 Reporting and Requesting

Bug reports

When reporting a bug please use one of the provided issue templates if applicable, otherwise just start a blank issue and describe your situation.

Documentation improvements

taurenmd could always use more documentation, whether as part of the official docs, in docstrings, or even on the web in blog posts, articles, and such. Write us a documentation issue describing what you would like to see improved in here, and, if you can do it, just Pull Request your proposed updates `:-)`.

Feature requests and feedback

The best way to send feedback is to file an issue using the feature template.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that code contributions are welcome :)

1.4.2 Code Development

General instructions

Though *taurenmd* can be used without relying on Anaconda to manage its dependencies, some dependencies are only available in the Anaconda ecosystem. Therefore, although *taurenmd* can be used almost entirely without relying on Anaconda, its development is bound, currently, to Anaconda. You can read further on this on our installation page.

Also, by using Conda, our CI integration is much faster because the heavy dependencies are handled just faster in Anaconda than in pure PyPI. However, it is possible to do some gymnastics and avoid using Conda and still contribute to the development *taurenmd*; if this is your case, please write us an issue, and we will help you out.

From this moment on, we will assume you are using Anaconda as your Python package manager. Also, we are assuming you have Git installed, and you are running a Linux machine – we are not aware of the possible divergences of these instructions in other OSes.

1. Create a new environment with the *taurenmd* dependencies, **only its dependencies**, running the following 3 commands:

```
curl https://raw.githubusercontent.com/joaomcteixeira/taurenmd/master/requirements-
↪dev.yml -o taurenmddev.yml
conda env create -f taurenmdenv.yml
# Activate the conda taurenmddev environment
conda activate taurenmddev
```

2. Fork taurenmd (look for the "Fork" button on the top right corner of our repository).
3. Clone your forked repository to your local machine:

```
git clone https://github.com/YOUR-USER-NAME/taurenmd.git <destination folder>
```

4. Navigate to the fork folder and create a branch for local development:

```
git checkout -b name-of-your-bugfix-or-feature
```

5. Install a development version of your development branch, remember to active the `taurenmddev` environment beforehand:

```
python setup.py develop
```

Now you can make your changes locally.

6. When you're done making changes run all the checks and docs builder with **tox** one command:

```
tox
```

And correct the errors reported. You can run individual test environment with *tox*, for example, to test syntax and code style and building related tasks:

```
tox -e check
```

to test documentation:

```
tox -e docs
```

to perform coverage-reported tests:

```
tox -e py37
# or if you are implementing on python3.6
tox -e py36
```

altogether, for example:

```
tox -e check,docs
```

Note: The `tox` command will run testing environments for both `py36` and `py37`, it will fail if you don't have both interpreters installed. If that is the case, just ignore the errors for that env.

7. Commit your changes and push your branch to your *taurenmd fork* on GitHub:

```
git add .
git commit -m "Your detailed description of your changes."
git push origin name-of-your-bugfix-or-feature
```

8. Submit a pull request through the GitHub website.

A New Command-line client

One of the most natural and straightforward ways to contribute to *taurenmd* is to develop a new command-line interface that implements a new analysis routine; in this way this routine becomes available in the *taurenmd* catalog. We provide a command-line client template file from which you can start developing your own new command-line client, just copy the template file to a new file named `cli_NAME.py` and follow the instructions provided as comments in that same file, the instructions will guide you throughout all required steps. Found the template file under `src/taurenmd/` folder.

Pull Request Guidelines

If you need some code review or feedback while you're developing the code just make a pull request.

For merging, you should:

1. Make sure your PR passes all `tox` tests[1].
2. Update documentation when there's new API, functionality etc.
3. Add a note to `CHANGELOG.rst` about the changes.
4. Add yourself to `AUTHORS.rst`.

1.4.3 Continuous Integration

This project follows Continuous Integration (CI) good practices (let us know if something can be improved). As referred to in the previous section, CI environment is provided by tox in combination with tox-conda. All *tox* testing environments run on Travis-CI; there, we check for code style integrity, documentation, tests, and test coverage. CI configuration is defined in the tox.ini and in the .travis.yml files.

Currently, we do not provide testing for Windows and MacOSX platforms. *taurenmd* depends on several research libraries, and we cannot, and should not, attempt to guarantee proper installation of those libraries on all platforms. Therefore we decided to provide full test coverage just for Linux systems where we know those libraries operate entirely, **however**, `taurenmd` code is written using only Python interfaces, which should make it cross-platform compatible. You may wish to read our Installation page for more comments on this matter.

[1] If you don't have all the necessary python versions available locally you can rely on Travis - it will run the tests for each change you add in the pull request.
It will be slower though ...

1.5 Citing

If you use **taurenmd**, please cite it in your publications and research/work environment, referring always to the main publication:

- Teixeira, J. M., (2020). taurenmd: A command-line interface for analysis of Molecular Dynamics simulations. Journal of Open Source Software, 5(50), 2175. https://doi.org/10.21105/joss.02175

BibText:

```
@article{Teixeira2020,
  doi = {10.21105/joss.02175},
  url = {https://doi.org/10.21105/joss.02175},
  year = {2020},
  publisher = {The Open Journal},
  volume = {5},
  number = {50},
  pages = {2175},
  author = {João M.C. Teixeira},
  title = {taurenmd: A command-line interface for analysis of Molecular Dynamics␣
↪simulations.},
  journal = {Journal of Open Source Software}
}
```

This project is also indexed at Zenodo and, if needed, you can complement the above citation in two by:

1. Referring to the whole project:

   ```
   João M.C. Teixeira. joaomcteixeira/taurenmd: Zenodo. http://doi.org/10.5281/zenodo.
   ↪3551990
   ```

2. **or** referring the exact version used, for example:

   ```
   João M.C. Teixeira. (2019, December 26). joaomcteixeira/taurenmd: v0.7.2 (Version␣
   ↪v0.7.2). Zenodo. http://doi.org/10.5281/zenodo.3593004
   ```

Citing is one of the best ways to support this project.

1.5.1 Citing Dependencies

When using and citing `taurenmd`, you **SHOULD by all means** cite the Molecular Dynamics (MD) analysis libraries, and others, with which `taurenmd` operates to perform the executions you have used. These MD libraries are the *tauremd software dependencies*. Taurenmd uses different libraries for the different *client interfaces*, each command-line interface documentation has a `References` section that indicates third party libraries used that you should cite.

After each command execution, the command used for that execution is appended to the `.taurenmd.cmd` file (see *logging*). Also a reference to each third pary project used during that execution is appended after the command registry, follow the citing instruction to properly cite the related projects.

Bellow, links to citing instructions of other research projects **taurenmd** uses as dependencies, alphabetical order:

1. MDAnalysis citing
2. MDTraj citing

3. matplotlotlib citing
4. Numpy citing
5. OpenMM citing
6. PyQuaternion citing
7. python-bioplottemplates citing

1.6 Authors

- Joao MC Teixeira - https://bit.ly/joaomcteixeira

1.7 Acknowledgements

The initial concept of this project was largely inspired in the pdb-tools *one script one action* idea; here we then pushed this concept further.

The authors deeply thanks to JoaoRodrigues for all the mentoring on MD and to Susana Barrera-Vilarmau for her intensive usage of the program during the very first development versions, discussion, feedback and suggestions on building a user-friendly interface.

Repository layout and Continuous Integration setup was initially provided by cookiecutter-pylibrary, with final personalized modifications by the authors.

1.8 License

1.9 Versioning

This project follows strictly Semantic Versioning 2.0 for version control.

Upon release of version 1.0.0, all additions to the `master` branch are done by PR followed by its respective version increment and release on PyPI.

Upon version 0.8, and before version 1, SV2 major version increments are reflected in the *minor* version number, and minor and patch increments are reflected together in the *patch* version number. Everything else follows SV2 rules, in this way users can track backwards incompatibilities if they happen.

1.10 Changelog

1.10.1 v0.11.2 (2022-07-14)

- Add output dir option to *fext* #66

1.10.2 v0.11.1 (2022-07-14)

- Update badges after 0.11 #65

1.10.3 v0.11.0 (2022-07-13)

- Dropped *python-bioplottemplates*
- updated export statements
- Added plots to *taurenmd rotations*
- Added plots to *taurenmd dist*
- Added plots to *taurenmd rmsf*
- Added plots to *taurenmd rmsd*
- *taurenmd dist* now can take multiple *–sel2*, and all are plotted
- *ParamsToDict* now uses *ast.literal_eval*
- trajectory slice now accepts timesteps (10ns, for example)

1.10.4 v0.10.4 (2022-01-14)

- update logo link in README #63

1.10.5 v0.10.3 (2021-12-11)

- Improved documentation for *cli_rotations*
- Added illustration explaining the roll, pitch, and yaw angles

1.10.6 v0.10.2 (2021-12-03)

- Add frame timestep and whole trajectory duration to *report* (#61)

1.10.7 v0.10.1 (2021-11-24)

- Add support for CIF topologies in libmda using openmm.app.pdbxfile
- removes the install*.yml gitactions

1.10.8 v0.10.0 (2021-11-24)

- Updates MDAnalysis to version 2.0.0
- Defines versions for all other dependencies
- Updates CI

1.10.9 v0.9.9 (2021-11-22)

- Corrects taking/passing of *insort* argument in some clients.

1.10.10 v0.9.8 (2021-11-22)

- Upgraded roll, pitch, and yaw angle calculations with torsion angle strategy
- Removed usage of PyQuaternion

1.10.11 v0.9.7 (2021-11-20)

- improved trajectory report (#54)

1.10.12 v0.9.6 (2021-02-22)

- Remove README badge for build: not needed.
- Added some comments in the workflow files.
- no changes in code, only CI.

1.10.13 v0.9.5 (2021-02-17)

- Upgraded CI to Github Actions according to: https://github.com/joaomcteixeira/python-project-skeleton
- Updated README badges

1.10.14 0.9.4 (2020-06-02)

- Updates documentation with JOSS citation (PR #49)

1.10.15 0.9.3 (2020-05-25)

- Improves CONTRIBUTION.rst guidelines (PR #46)

1.10.16 0.9.2 (2020-05-17)

- Client progression is now represented by a progress bar (PR #44)

1.10.17 0.9.1 (2020-05-15)

- Improves log in *.taurenmd.cmd* (PR #43)

1.10.18 0.9.0 (2020-05-15)

- Adds *-i* to every *CLI* interface (PR #42)
- *major* version change because *cli_imagemol* lost backwards compatibility

1.10.19 0.8.14 (2020-05-12)

- Updates *tox.ini* file for Continuous Integration (PR #40)

1.10.20 0.8.13 (2020-05-12)

- Added support for sequence of trajectories in CLIs that use *MDTraj* (PR #39)

1.10.21 0.8.12 (2020-05-04)

- PR: #37
- Installs taurenmd directly with Conda env

1.10.22 0.8.11 (2020-04-03)

- PR: #33
- Corrected command representation in .taurenmd.cwd adding quotes when needed

1.10.23 0.8.10 (2020-04-02)

- PR: #32
- Corrects incorrect usage of MDAnalysis.analysis.alignto function in `trajedit`.

1.10.24 0.8.9 (2020-03-03)

- Changed logos, PR #28

1.10.25 0.8.8 (2020-02-03)

- PRs: #25 #26 #27
- Added taurenmd logo for readthedocs
- Added tauranmd logo in README
- Added taurenmd repository banner
- Improved details in the documentation
- Removed `.ci` folder, unnecessary

1.10.26 0.8.7 (2020-02-02)

- PR #24
- Added PyPI downloads badge
- Improved installation instructions
- Improved and clarified contributing instructions

1.10.27 0.8.6 (2020-01-20)

- Restructured pip deps: install_requires only takes bioplottemplates and pyquaternion
- two extras_require: *sup* and *md* and *all* which consider both

1.10.28 0.8.5 (2020-01-20)

- PR #22
- organized dependencies for PyPI
- PyPI only dependencies are referred as install_requires
- MDAnalysis and MDTraj referred in extras_require
- OpenMM left out from pip, only available in Anaconda

1.10.29 0.8.4 (2020-01-19)

- PR #15
- Added simtk lib import check for controlled failure
- added error message output for user

1.10.30 0.8.3 (2020-01-19)

- PR #16 and #19
- corrected argparse autodoc in ReadTheDocs (mock strategy)
- improved tox configuration with better env separation
- #19 reports a communication error between TravisCI and coverage servers

1.10.31 0.8.2 (2020-01-17)

- Improved CI workflow * Dropped COVERALLS * Dropped Codacy * Setup test-coverage in CodeClimate * created *.codeclimate.yml* with explicit configuration
- updated badges

1.10.32 0.8.1 (2020-01-15)

- PR #14
- Corrected version display in documentation

1.10.33 0.8.0 (2020-01-15)

- PR #13
- Code architecture improvements
- Complete project main documentation
- Complete library documentation
- command line documented
- Code clean

1.10.34 0.7.2 (2019-12-25)

- bridged from 0.7.1
- Dropped Appveyor and EXPLICIT Windows support because of #1.
- restructured project GitHub layout. Deprecated develop branch.
- Readthedocs documentation improvements in structure and content.

1.10.35 0.7.0 (2019-12-23)

- **implemented `cli_rotations`, calculates roll, pitch and yaw**
  rotation angles of selection.

1.10.36 0.6.0 (2019-12-15)

- implemented `cli_rmsf` to calculate RMSFs.

1.10.37 0.5.1 (skipped to 0.6.0)

- added sort numbered trajs to `cli_trajedit`
- added sort numbered trajectory paths in lib
- improved `cli_imagemol` readability
- added selection in `cli_noSol`

1.10.38 0.5.0 (2019-11-24)

- created `cli_angle`. Calculates angles between a plane along the trajectory. Plane is given by the three centre_of_geometries of three selections.
- args to plot passed as list are transformed to tuple
- added distance calc and plot interface `cli_distances`
- `trajedit` now saves topology unwrapped

1.10.39 0.4.1 (2019-11-21)

- renumbered version to 0.4.1. from 0.3.1
- RMSD Cli now calculates for several selections
- Parse plot vars now registers floats
- corrected fext cli entry point
- added align option to trajedit
- topology model writen from first frame of time slicing
- added unwrap() molecule method from MDAnalysis in `trajedit` with respective options
- topology output now defaults to traj name + `frame0.pdb`
- added .myparents() to Path in `__init__`

1.10.40 0.3.0 (2019-11-06)

- Created *develop* branch
- Created client for frame extraction: `cli_fext`
- Added option to disable export of frame0 topology in trajedit

1.10.41 0.2.1 (2019-10-26)

- dropped py35
- separated lib MDAnalysis from MDTraj
- `libio` concerns only general functions
- improved `imagemol` I/O

1.10.42 0.2.0 (2019-10-26)

- added `cli_report`

1.10.43 0.1.1 (2019-10-26)

- corrected libio
- trajectory loads based on MDAnalysis now read and concatenate multiple trajectories.

1.10.44 0.1.0 (2019-10-26)

- added interfaces: * `trajedit` * `noSol` * `imagemol` * `rmsd` * `cli template`

1.10.45 0.0.0 (2019-10-15)

- First release on PyPI.

1.11 Library Documentation

Thorough documentation on the *command-line* interfaces exists in *this page*. Here the **taurenmd** internals are documented, you may use them independently for a more advanced developer use.

Library package.

Taurenmd libraries, all prefixed as `lib`, contain the functions used by the client interfaces. Though taurenmd is designed to be used as a command-line interface, we provide detailed documentation for its internal functions.

Lib packages are organized by their scope, some have `libs` have general scopes while other are tightly bound to the Molecular Dynamics library with which they operate. For example:

1. `libmda` for MDAnalysis
2. `libmdt` for MDTraj

These modules store functions that relate solely to the scope of MDAnalysis and MDTraj packages; normally these are I/O related operations.

On the other hand, we decided to organize other libraries based on the scope of their functions regardless of the dependencies they use. For example, `libcalc` contains functions calculate MD parameters, and combine the usage of different libraries when needed.

Further instructions are provided within each module documentation.

1.11.1 core

Library wide core utils.

class taurenmd.core.**Path**(**args*, ***kwargs*)

Extends Python's Path object interface.

myparents()

List of the path parent folders.

Alias to `pathlib.Path.resolve().parents[0]`.

Returns
list – Parent paths. Name file or folder are excluded.

str()

Represent path as string.

Alias to `os.fspath(self)`.

Returns
str – `os.fspath(self)`.

taurenmd.core.**add_reference**(*ref*)

Add reference decorator.

Example

```
>>> @add_reference(str)
>>> def myfunct():
>>>     ...
```

taurenmd.core.**ref_mda = '* MD data accessed using [MDAnalysis](https://www.mdanalysis.org).\n'**

Command-line docstring to reference MDAnalysis package.

taurenmd.core.**ref_mda_alignto = '* align performed by MDAnalysis [unwrap](https://www.mdanalysis.org/docs/documentation_pages/analysis/align.html?highlight=alignto#MDAnalysis.analysis.align.alignto).\n'**

Command-line docstring to reference MDAnalysis alignto function.

taurenmd.core.**ref_mda_selection = '* selection commands follow MDAnalysis [selection nomenclature](https://www.mdanalysis.org/docs/documentation_pages/selections.html#).\n'**

Command-line docstring to reference MDAnalysis selection commands.

taurenmd.core.**ref_mda_unwrap = '* unwrap performed by MDAnalysis [unwrap](https://www.mdanalysis.org/docs/documentation_pages/core/groups.html?highlight=unwrap#MDAnalysis.core.groups.AtomGroup.unwrap).\n'**

Command-line docstring to reference MDAnalysis selection.unwrap method.

taurenmd.core.**ref_mdt = '* MD data accessed and/or processed using [MDTraj](https://mdtraj.org/)\n'**

Command-line docstring to reference MDTraj package.

taurenmd.core.**ref_numpy = '* Matrix operations performed with [Numpy](https://www.scipy.org/citing.html).'**

Command-line docstring to reference numpy lib.

taurenmd.core.**ref_openmm = '* Data loaded with [OpenMM](http://openmm.org/)'**

Command-line docstring to reference OpenMM package.

taurenmd.core.**ref_taurenmd = '* Cite taurenmd according to: https://taurenmd.readthedocs.io/en/latest/citing.html\n'**

How to cite taurenmd project.

1.11.2 libcalc

Calculate parameters from Molecular Dynamics data.

This module contains functions to calculate MD parameters such as:

1. RMSDs
2. RMSFs
3. plane angle variation
4. axes rotation decomposition

It contains also other functions that help on the calculation of the desirables. Those functions are also avaible for independent use.

This library contains functions that operate on different Molecular Dynamics data types. When special data types (MD analysis libraries) are used, a prefix to the function name is used, and its docstring explicitly refers to it.

When using these functions, you should always cite taurenmd together with the other library(ies) used. Read our citing reference page.

taurenmd.libs.libcalc.**calc_plane_eq**(*p1*, *p2*, *p3*)

Calculate equation that defines the (p1, p2, p3) plane.

Further reading.

Parameters

p1, p2, p3 (*numpy.ndarray of shape (3,)*) – The three 3D coordinate points that define the plane.

Returns

tuple of length 4 – The four parameters (a, b, c, d) that defined the plane equation:

$$ax + by + cz = d$$

taurenmd.libs.libcalc.**calc_plane_normal**(*p1*, *p2*, *p3*)

Calculate the normal vector for the (p1, p2, p3) plane.

Given 3 XYZ space coordinate points, calculates the normal vector of the plane defined by those points.

Parameters
p1, p2, p3 (*numpy.ndarray of shape (3,)*) – The three 3D coordinate points that define the plane.

Returns
Numpy array of shape (3,) – The normal vector to the (p1, p2, p3) plane. This vector is **NOT** an unitary vector.

taurenmd.libs.libcalc.**calc_planes_angle**(*a1*, *b1*, *c1*, *a2*, *b2*, *c2*, *aunit='radians'*)

Calculate the angle between two planes.

Plane 1 is defined by a1, b1, c1 plane parameters, plane 2 is defined by a2, b2, c2, where:

$$a1 * x + b1 * y + c1 * z + d = 0$$
$$a2 * x + b2 * y + c2 * z + d = 0$$

Read further.

Parameters

- **a1, b1, c1, a2, b2, c2** (*float*) – Plane parameters
- **angle** (*str, optional*) – `degrees` returns angle quantity in degrees, else returns in radians.

Returns
float – The angle between plane 1 and plane 2.

taurenmd.libs.libcalc.**calc_torsion_angles**(*coords*)

Calculate torsion angles from sequential coordinates.

Uses `NumPy` to compute angles in a vectorized fashion. Sign of the torsion angle is also calculated.

Uses Prof. Azevedo implementation: https://azevedolab.net/resources/dihedral_angle.pdf

Example

Given the sequential coords that represent a dummy molecule of four atoms:

```
>>> xyz = numpy.array([
>>>     [0.06360, -0.79573, 1.21644],
>>>     [-0.47370, -0.10913, 0.77737],
>>>     [-1.75288, -0.51877, 1.33236],
>>>     [-2.29018, 0.16783, 0.89329],
>>>     ])
```

A1—A2
A3—A4

Calculates the torsion angle in A2-A3 that would place A4 in respect to the plane (A1, A2, A3).

Likewise, for a chain of N atoms A1, …, An, calculates the torsion angles in (A2, A3) to (An-2, An-1). (A1, A2) and (An-1, An) do not have torsion angles.

If coords represent a protein backbone consisting of N, CA, and C atoms and starting at the N-terminal, the torsion angles are given by the following slices to the resulting array:

- phi (N-CA), [2::3]
- psi (CA-N), [::3]
- omega (N-C), [1::3]

Parameters

coords (*numpy.ndarray of shape (N>=4, 3)*) – Where *N* is the number of atoms, must be equal or above 4.

Returns

numpy.ndarray of shape (N - 3,) – The torsion angles in radians. If you want to convert those to degrees just apply `np.degrees` to the returned result.

`taurenmd.libs.libcalc.`**`mda_rmsd`**(*universe*, *frame_slice=None*, *selection='all'*, *ref_frame=0*)

Calculate RMSDs observed for a selection.

Uses MDAnalysis RMSD.

Example

- Calculate RMSDs observed for the whole system along the whole trajectory.

```
>>> mda_rmsd(universe)
```

- Calculate the RMSDs observed for selection *segid A* for every 10 frames.

```
>>> mda_rmsd(universe, slice(0, None, 10), selection='segid A')
```

Parameters

- **MDAnalysis Universe** – The MDAnalysis universe.
- **frame_slice** (*any, optional*) – The frames in the trajectory to consider. If `None` considers all frames. Accepts any argument that *`taurenmd.libs.libio.evaluate_to_slice()`* can receive. Defaults to `None`.
- **selection** (*str, optional*) – The selection upon which calculate the RMSDs. Defaults to `'all'`.
- **ref_frames** (*int, optional*) – The reference frame against which calculate the RMSDs. Defaults to `0`.

Returns

Numpy Array – The array containing the calculated RMSDs of shape (N,), where N is the number of frames.

Raises

- **`ValueError`** – If `frame_slice` is not `None` or `slice` object.
- **`MDAnalysis Exceptions`** – Any exceptions that could come from MDAnalysis RMSF computation.

`taurenmd.libs.libcalc.`**`mda_rmsf`**(*atom_group*, *frame_slice=None*)

Calculate RMSFs.

Uses MDAnalysis RMSF.

Parameters

- **atom_group** (*MDAnalysis Atom Group.*) – MDAnalysis Atom group.
- **frame_slice** (*any, optional*) – Any argument that `taurenmd.libs.libio.evaluate_to_slice()` can receive. Defaults to `None`, considers all frames.

Returns

Numpy Array – With the calculated RMSFs values, of shape (N,) where N are the frames sliced from `frame_slice`.

Raises

`MDAnalysis Exceptions` – Any exceptions that could come from MDAnalysis RMSF computation.

`taurenmd.libs.libcalc.`**`torsion_set`**(*p1*, *p2*, *p3*, *p4_vecs*)

Calculate torsion angles of set torwards three initial vectors.

Same implementation as *calc_torsion_angles* but A1, A2, and A3 are fixed and A4 is an array of vectors. The torsion angles are calculated for A4s with respect to the fixed A1, A2, and A3.

Example

```
>>> torsion_set(
    [x1, y1, z1],
    [x2, y2, z2],
    [x3, y3, z3],
    [
        [x4_1, y4_1, z4_1],
        [x4_2, y4_2, z4_2],
        [x4_3, y4_3, z4_3],
        [x4_4, y4_4, z4_4],
        ...
        ],
    )
```

Returns

numpy.ndarray of shape (N,) – The anges in radians. Where N is the length of A4.

1.11.3 libcli

Shared operations for client interfaces.

This module contains functions and classes that are shared amongst the client interfaces. It contains also others used to enhance the user experience.

`class` `taurenmd.libs.libcli.`**`CustomParser`**(*prog=None*, *usage=None*, *description=None*, *epilog=None*, *parents=[]*, *formatter_class=<class 'argparse.HelpFormatter'>*, *prefix_chars='-'*, *fromfile_prefix_chars=None*, *argument_default=None*, *conflict_handler='error'*, *add_help=True*, *allow_abbrev=True*)

Custom Parser class.

`error`(*message*)

Present error message.

class taurenmd.libs.libcli.**ParamsToDict**(*option_strings*, *dest*, *nargs=None*, *const=None*, *default=None*, *type=None*, *choices=None*, *required=False*, *help=None*, *metavar=None*)

Convert command-line parameters in an argument to a dictionary.

Example

Where `-x` is an optional argument of the command-line client interface.

```
>>> par1=1 par2='my name' par3=[1,2,3]
>>> {'par1': 1, 'par2': 'my name', 'par3': [1, 2, 3]}
```

class taurenmd.libs.libcli.**ProgressBar**(*total*, *prefix=''*, *suffix=''*, *decimals=1*, *bar_length=None*)

Contextualizes a Progress Bar.

Parameters

- **total** (*int convertable*) – The total number o iteractions expected.
- **prefix** (*str*) – Some prefix to enhance readability.
- **suffix** (*str*) – Some suffix to enhance readability.
- **decimals** (*int-convertable*) – The demicals to show in percentage. Defaults to *1*.
- **bar_length** (*int, float, -convertable*) – The length of the bar. If not provided (`None`), uses half of the terminal window.
- **Thanks to for the initial template function**
- **https** (*//dev.to/natamacm/progressbar-in-python-a3n*)

Examples

```
>>> with ProgressBar(5000, suffix='frames') as PB:
>>>     for i in trajectory:
>>>         # do something
>>>         PB.increment()
```

increment()

Print next progress bar increment.

taurenmd.libs.libcli.**add_angle_unit_arg**(*parser*)

Add angle unit selectiona argument to parser.

Is defined by `-a` and `--aunit`.

Wether angles are to be calculated in degrees or radians.

Parameters

parser (argparse.ArgumentParser) – The argument parser to add the topology positionl argument.

taurenmd.libs.libcli.**add_atom_selection_arg**(*parser*)

Add selection optional argument.

Selection argument is a string that defines the atom selection, this is defined by `-l` and `--selection`, and defaults to `all`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the selection argument.

taurenmd.libs.libcli.**add_atom_selections_arg**(*parser*)

Add selections optional argument.

Selections argument is a string that defines a list of atom selections, this is defined by `-g` and `--selections`, and defaults to `all`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the selections argument.

taurenmd.libs.libcli.**add_data_export_arg**(*parser*)

Add export argument.

Export argument configures data export to a text file in table format.

Is defined by `-x` and `--export`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the export argument.

taurenmd.libs.libcli.**add_frame_list_arg**(*parser*)

Add frame list argument.

Registers a list of frame numbers, is defined by `-t` and `--flist`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the flist argument.

taurenmd.libs.libcli.**add_insort_arg**(*parser*)

Sort input by trail int.

Applies *taurenmd.libs.libio.sort_numbered_input()*.

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the insort argument.

taurenmd.libs.libcli.**add_inverted_array**(*parser*)

Add inverted selections optional argument.

Saves an array of 0s and 1s to invert the order of selections.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the selections argument.

taurenmd.libs.libcli.**add_output_dir_arg**(*parser*)

Add output dir to client.

Client gains *–odir* option.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the export argument.

taurenmd.libs.libcli.**add_plane_selection_arg**(*parser*)

Add plane selection argument.

Plane selection is a selection of three regions separated by ‘or’ operator.

Is defined by `-z` and `--plane-selection`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the plane-selection argument.

taurenmd.libs.libcli.**add_plot_arg**(*parser*)

Add plot parameters.

Plot kwargs that will be passed to the plotting function.

Defined by `--plot`.

If given, plot results. Additional arguments can be given to specify the plot parameters.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the plot argument.

taurenmd.libs.libcli.**add_reference_frame_arg**(*parser*)

Add a reference frame argument.

Reference frame is the frame to compute the parameter against.

Depending on the client logic the reference frame might have different meanings.

Is defined by `-r` and `--ref-frame`.

Defaults to `0`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the refence-frame argument.

taurenmd.libs.libcli.**add_slice_arg**(*parser*)

Add start, stop and step slicing arguments.

Slicing arguments are according to Python Slice object

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the trajectory positionl argument.

taurenmd.libs.libcli.**add_subparser**(*parser*, *module*)

Add a subcommand to a parser.

Parameters

- **parser** (argparse.add_suparsers object) – The parser to add the subcommand to.
- **module** – A python module containing the characteristics of a taurenmd client interface. Client interface modules require the following attributes: `__doc__` which feeds the description argument of add_parser, `_help` which feeds help, `ap` which is an ArgumentParser, and a `main` function, which executes the main logic of the interface.

taurenmd.libs.libcli.**add_top_output_arg**(*parser*)

Add argument to export first frame as topology PDB file.

Defined by `-o` and `--top-output`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the topology output argument.

taurenmd.libs.libcli.**add_topology_arg**(*parser*)

Add topology positional argument to parser.

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the topology positionl argument.

taurenmd.libs.libcli.**add_traj_output_arg**(*parser*)

Add argument to export trajectory after client modifications.

Defined by `-d` and `--traj-output`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to which add the trajectory output argument.

taurenmd.libs.libcli.**add_trajectories_arg**(*parser*)

Add trajectory positional argument to parser.

Accepts multiple trajectory files.

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the trajectory positionl argument.

taurenmd.libs.libcli.**add_trajectory_arg**(*parser*)

Add trajectory positional argument to parser.

Accepts a single trajectory file.

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the trajectory positionl argument.

taurenmd.libs.libcli.**add_version_arg**(*parser*)

Add version `-v` option to parser.

Displays a message informing the current version. Also accessible via `--version`.

Parameters
parser (argparse.ArgumentParser) – The argument parser to add the version argument.

taurenmd.libs.libcli.**load_args**(*ap*)

Load user arguments.

taurenmd.libs.libcli.**maincli**(*ap*, *main*)

Client main function.

Operates when client is called directly outside the `taurenmd` client interface.

- Reads input parameters
- saves inpu command to log file
- runs client `main` function
- saves references to log file

Returns
The result value from client `main` function.

taurenmd.libs.libcli.**represent_argument**(*arg*)

Represent argument in a string.

If argument has spaces represents string with quotation marks “.

taurenmd.libs.libcli.**save_command**(*fname*, **args*)

Append the execution command to a log file.

Parameters

- **fname** (*string or Path*) – The file name of the log file where to append the command.
- ***args** (*strings*) – String parts that compose the command.

taurenmd.libs.libcli.**save_references**()

Save used references to log file.

1.11.4 libio

Handle input and output general operations.

taurenmd.libs.libio.**add_prefix_to_path**(*ipath*, *prefix*)

Add prefix to file path.

Example

```
>>> mk_frame_path('traj_output.xtc', prefix='my_prefix')
>>> my_prefixtraj_output.xtc
```

Mind the _ is not placed automatically.

```
>>> mk_frame_path('traj_output.xtc', prefix='my_prefix_')
>>> my_prefix_traj_output.xtc
```

Parameters

- **ipath** (*str or Path*) – The file path to alter.
- **prefix** (*str*) – The complete prefix for the file name.

Returns

taurenmd.core.Path() – The new file path.

taurenmd.libs.libio.**add_suffix_to_path**(*ipath*, *suffix*)

Add suffix to file path.

If suffix has extention, updates the path extension, otherwise keeps the original extension.

Examples

```
>>> mk_frame_path('traj_output.xtc', suffix='my_suffix')
>>> traj_outputmy_suffix.xtc
```

Mind the underscore is not placed automatically:

```
>>> mk_frame_path('traj_output.xtc', suffix='_my_suffix')
>>> traj_output_my_suffix.xtc
```

Updating extensions:

```
>>> mk_frame_path('traj_output.xtc', suffix='_my_suffix.pdb')
>>> traj_output_my_suffix.pdb
```

Parameters

- **ipath** (*str or Path*) – The file path to alter.
- **suffix** (*str*) – The complete suffix for the file name, extension should be included in the suffix, extension of the `ipath` is ignored.

Returns

taurenmd.core.Path() – The new file path.

taurenmd.libs.libio.**evaluate_to_slice**(*, *value=None*, *start=None*, *stop=None*, *step=None*)

Evaluate to slice.

If any of `start`, `stop` or `step` is given returns `slice(start, stop, step)`. Otherwise tries to evaluate `value` to its representative slice form.

Examples

```
>>> evaluate_to_slice(value='1,100,2')
>>> slice(1, 100, 2)
```

```
>>> evaluate_to_slice(start=10)
>>> slice(10, None, None)
```

```
>>> evaluate_to_slice(value=(0, 50, 3))
>>> slice(0, 50, 3)
```

```
>>> evaluate_to_slice(value=(None, 100, None))
>>> slice(None, 100, None)
```

```
>>> #ATTENTION
>>> evaluate_to_slice(value='10')
>>> slice(10, None, None)
>>> # this is different from slice(10)
>>> #though
>>> evaluate_to_slice(value=10)
>>> slice(None, 10, None)
```

Parameters

- **value** (*list, tuple, str, None or int*) – A human readable value that can be parsed to a slice object intuitively. Defaults to `None`.
- **start** (*None or int*) – The starting index of the slice (inclusive). Defaults to `None`.
- **stop** (*None or int*) – The final index of the slice (exclusive). Defaults to `None`.
- **step** (*None or int*) – Slice periodicity. Defaults to `None`.

Returns

slice – Python slice object.

Raises

ValueError – If slice can not be computed.

taurenmd.libs.libio.**export_data_to_file**(*xdata*, **ydata*, *fname='results.csv'*, *header=''*, *fmt='{:.3f}'*, *delimiter=','*)

Save data to file.

Following the format:

```
>>> # header
>>> x1,ya1,yb1,yc1
>>> x2,ya2,yb2,yc2
>>> x3,ya3,yb3,yc3
```

Where `x` are the elements in `xdata`, and `y*` are the elements in the different `ydata` series.

Parameters

- **xdata** (*list-like*) – Contains the *x axis* data.
- **ydata** (*list of list-like*) – Contains the *y axis* data series.
- **fname** (*str*) – The output file path name.
- **header** (*str*) – The commented header of the file. Comment character, like `#` is not placed, it should be already provided if desired.
- **fmt** (*str*) – The float format. Defaults to `{:.3f}`.
- **delimiter** (*str*) – The string delimiter between columns. Defaults to `,`.

taurenmd.libs.libio.**frame_list**(*len_traj*, *start=None*, *stop=None*, *step=None*, *flist=None*)

Create frame integer list from a length and slice parameters.

Examples

```
>>> frame_list(10)
>>> list(range(10)) # returns
```

```
>>> frame_list(10, start=2, stop=5)
>>> list(range(2, 5)) # returns
```

```
>>> frame_list(10, 2, 50)
>>> list(range(2, 10)) # returns
```

```
>>> frame_list(10, None, None, 2)
>>> list(range(0, 10, 2)) # returns
```

```
>>> frame_list(10, flist='1,2,45,65')
>>> [1, 2, 45, 65]
```

```
>>> frame_list(10, flist=[1, 2, 45, 65])
>>> [1, 2, 45, 65]
```

```
>>> frame_list(10, flist=['1', '2', '45', '65'])
>>> [1, 2, 45, 65]
```

```
>>> frame_list(None, flist=['1', '2', '45', '65'])
>>> [1, 2, 45, 65]
```

Parameters

- **len_traj** (*int*) – The length to evaluate. Normally this is the length of the trajectory.
- **start** (*int or None, optional*) – The start index for the frame list after length evaluation. Defaults to None.
- **stop** (*int or None, optional*) – The stop index for the frame list after length evaluation. Defaults to None.
- **step** (*int or None, optional*) – the step index for the frame list after length evaluation. Defaults to None.
- **flist** (*list-like, or comma-separated string, optional*) – The list of specific frames. Defaults to None.

Returns

list – The resulting frame list.

taurenmd.libs.libio.**frame_slice**(*start=None*, *stop=None*, *step=None*)

Create a slice object from parameters.

Logs the created slice object.

Returns

Slice Object – The slice object created from slice(start, stop, step)

taurenmd.libs.libio.**get_number**(*path*)

Extract tail number from path.

Examples

```
>>> get_number('traj_1.dcd')
>>> 1
```

```
>>> get_number('traj_3.dcd')
>>> 3
```

```
>>> get_number('traj_1231.dcd')
>>> 1231
```

```
>>> get_number('traj_0011.dcd')
>>> 11
```

```
>>> get_number('traj_1_.dcd')
>>> 1
```

```
>>> get_number('traj_20200101_1.dcd')
>>> 1
```

Parameters
path (*str or Path obj*) – The path to evaluate.

Returns
int – The tail integer of the path.

taurenmd.libs.libio.**mk_frame_path**(*ipath*, *frame=0*, *ext='.pdb'*, *leading=0*, *suffix=None*)

Create the path name for a frame.

Given an input_path `ipath` (usually the name of the trajectory), create the corresponding frame file name.

Example

```
>>> mk_frame_path('traj_output.xtc')
>>> traj_output_frame0.pdb
```

```
>>> mk_frame_path('traj_output.xtc', frame=4, leading=4)
>>> traj_output_frame0004.pdb
```

Parameters

- **ipath** (*str or Path*) – The file path. Normally, trajectory file path.
- **frame** (*int, optional*) – The frame to label the new path. Defaults to `0`.
- **ext** (*str, optional*) – The returned file desired extension. Defaults to `.pdb`.
- **leading** (*int*) – The leading zeros to left append to the frame number. Defaults to `0`.
- **suffix** (*str*) – Complete specifications of the desired suffix. If `suffix` is given, `frame` and `ext` and `leading` are ignored and *add_suffix_to_path()* is used directly.

Returns
taurenmd.core.Path() – The new file path.

taurenmd.libs.libio.**parse_top_output**(*top_output*, *traj_output=None*)

Parse different output definitions for topology output file name.

If `top_output` startswith `_` uses :py:func:add_suffix_to_path. If `top_output` endswith `_` uses :py:func:add_prefix_to_path. Else: Return Path object of `top_output`.

Parameters

- **top_output** (*str or Path*) – The string to evaluate.
- **traj_output** (*str or Path*) – The trajectory output file name. Return value depends on this parameters.

Returns
taurenmd.core.Path() – The new topology file path.

taurenmd.libs.libio.**report_input**(*topology*, *trajectories*)

Report on topology and trajectory file paths used as input.

taurenmd.libs.libio.**sort_numbered_input**(**inputs*)

Sort input paths to tail number.

If possible, sort criteria is provided by *get_number()*. If paths do not have a numbered tag, sort paths alphabetically.

Parameters
***inputs** (*str of Paths*) – Paths to files.

Returns
list – The sorted pathlist.

1.11.5 libmda

Functions that wrap around MDAnalysis library.

Functions contained in this module operate with MDAnalysis (MDA) functionalities, either by using MDA to access Molecular Dynamics data or by receiving MDA data structures and parsing them in some way.

When using functions contained in this library you should cite both taurenmd and MDAnalysis.

taurenmd.libs.libmda.**convert_time_or_frame_to_frame**(*s*, *u*)

Convert a time or frame string to frame.

Parameters

- **s** (*str or int or float*) – A string defining the time ('12ns') or frame.
- **u** (*mda.Universe*)

Returns
int – see *convert_time_to_frame()*

taurenmd.libs.libmda.**convert_time_to_frame**(*x*, *dt*, *base_unit='ps'*)

Convert a string *x* into a frame number based on given *dt*.

If *x* does not contain any units its assumed to be a frame number already.

Original function taken from *MDAnalysis.mdacli* project, from commit: https://github.com/MDAnalysis/mdacli/blob/15f6981df7b14ef5d52d64b56953d276291068ab/src/mdacli/utils.py#L25-L66 The original function was modified internally without modifying its API.

See also: https://github.com/MDAnalysis/mdacli/pull/81

Note that in case of decimal values, for example, 10.3ps, the integer division between the value and *dt* remains.

Parameters

- **x** (*str, int or float*) – the input string
- **dt** (*float*) – the time step in ps

Returns
int – frame number

Raises
ValueError – The input does not contain any units but is not an integer.

taurenmd.libs.libmda.**create_x_data**(*u*, *xdata_in_time*, *frame_list*)

Create X data for plotting.

Create a frame list given a time or else return the frame_list given.

Parameters

- **u** (*mda.Universe*)
- **xdata_in_time** (*bool*) – Whether to select frames based on a time duration.
- **frame_list** (*list of int*) – A list of integers referring to the number of frames in the trajectory.

taurenmd.libs.libmda.**draw_atom_label_from_atom_group**(*atom_group*)

Translate MDAnalysis Atom Group to list of strings for each atom.

Strings represent each atom by SEGID.RESNUM|RESNAME.NAME, for example carbon alpha of Cys 18 of chain A would:

```
>>> A.18Cys.CA
```

This function is used by taurenmd for data representation purporses.

Parameters
atom_group (*Atom Group obj*) – MDAnalysis Atom group.

Returns
list of strings – Containing the atom string representation for each atom in the Atom Group.

taurenmd.libs.libmda.**get_frame_list_from_slice**(*u*, *frame_slice*)

Create a frame number list from a slice for a Universe.

Parameters

- **u** (*mda.Universe*) – The MDAnalysis universe.
- **frame_slice** (*slice object*)

Returns
list of ints – A list with the number of frames corresponding to that Universe and slice.

taurenmd.libs.libmda.**get_frame_slices**(*u*, *start=None*, *stop=None*, *step=None*)

Make a frame slice for a universe given start, stop, and step.

Parameters
start, stop, step (*None or int or str.*) – If string, accepts `libutil.split_time_unit()`.

Returns
slice object

taurenmd.libs.libmda.**get_timestep**(*u*, *i0=0*, *i1=1*)

Get time step between two trajectory frames in picoseconds.

Parameters
it0, it1 (*int*) – The initial and end frame, respectively.

taurenmd.libs.libmda.**load_universe**(*topology*, **trajectories*, *insort=False*, ***universe_args*)

Load MDAnalysis universe.

Accepts MDAnalysis compatible topology formats and trajectory formats. Read further on the MDAnalysis Universe.

Examples

```
>>> libmda.load_universe('topology.pdb', 'trajectory.dcd')
```

```
>>> libmda.load_universe(
        'topology.pdb',
        'traj_part_1.xtc',
        'traj_part_2.xtc',
        Path('my', 'md', 'folder', 'traj_part_3.xtc'),
        )
```

Parameters

- **topology** (*str or Path object*) – Path to topology file.
- **trajectories*** (*str of Path objects*) – Paths to trajectory file(s). Trajectory files will be used sequentially to create the Universe.
- **insort** (*bool*) – Whether to sort trajectory files by suffix number. See `libio.sort_numbered_input()`.
- **universe_args** (*any*) – Other arguments to be passed to *MDAnalysis Universe*.

Returns

MDAnalysis Universe

`taurenmd.libs.libmda.`**`mdaalignto`**(*universe*, *reference*, *selection='all'*)

Align universe to reference.

Uses MDAnalysis.analysis.align.alignto.

Parameters

universe, reference, selection – Same as in `MDAnalysis.analysis.align.alignto` function.

Raises

`ZeroDivisionError` – If selection gives empty selection.

`taurenmd.libs.libmda.`**`report`**(*universe*)

Report information about the Universe.

Example

```
>>> u = libmda.load_universe('topology.pdb', 'trajectory.xtc')
>>> libmda.report(u)
```

Parameters

universe (*MDAnalysis Universe*) – MDAnalysis universe.

Returns

None

1.11.6 libmdt

Functions that wrap around MDTraj library.

Functions contained in this module operate with MDTraj functionalities, wither by using MDTraj to access Molecular Dynamics data or by receiving MDTraj data structures and parsing them in some way.

Simtk OpenMM is also used in some functions.

Read our citing documentation to understand how to cite multiple libraries.

`taurenmd.libs.libmdt.`**`imagemol_protocol1`**(*traj*)

Attempt to image molecules acting on the whole traj.

`taurenmd.libs.libmdt.`**`imagemol_protocol2`**(*traj*)

Attempt to image molecules frame by frame.

`taurenmd.libs.libmdt.`**`load_traj`**(*topology*, *trajectories*, *insort=False*)

Load trajectory with MDTraj.

Uses mdtraj.load.

Example

```
>>> libmdt.load_traj('bigtopology.cif', 'trajectory.dcd')
```

Parameters

- **topology** (*str or Path or list*) – Path to the topology file. Accepts MDTraj compatible topology files. mmCIF format is loaded using OpenMM.
- **trajectory** (*str or Path*) – Path to the trajectory file. Accepts MDTraj compatible files

Returns

MDTraj trajectory – Trajectory object.

1.11.7 plots

Plot a single parameter.

`taurenmd.plots.plotparams.`**`plot`**(*x_data*, *y_data*, ***, *labels='No label provided'*, *title=None*, *xlabel=None*, *ylabel=None*, *xlabel_fs=8*, *ylabel_fs=8*, *xticks=None*, *yticks=None*, *xticks_labels=None*, *yticks_labels=None*, *colors=('b', 'g', 'r', 'c', 'm', 'y', 'k')*, *alpha=0.7*, *xmax=None*, *xmin=None*, *ymax=None*, *ymin=None*, *hline=None*, *hline_color='black'*, *hline_lw=1*, *xticks_fs=10*, *yticks_fs=10*, *grid=True*, *grid_color='lightgrey'*, *grid_ls='-'*, *grid_lw=1*, *grid_alpha=0.5*, *legend=True*, *legend_fs=8*, *legend_loc=0*, *vert_lines=None*, *figsize=(8, 5)*, *filename='plot_param.pdf'*, *dpi=150*)

Plot a parameter.

Bellow parameters concern data representation and are considered of highest importance because their incorrect use can mislead data analysis and consequent conclusions.

Plot style parameters concernning only plot style, i.e., colors, shapes, fonts, etc… and which do not distort the actual data, are not listed in the paremeter list bellow. We hope these parameter names are self-explanatory and are listed in the function definition.

Parameters

- **x_data** (*interable of numbers*) – Container of the X axis data. Should be accepted by matplotlib.
- **y_data** (*np.ndarray, shape=(M, len(x_data))*) – Container of the Y axis data. Where M is the number of series.
- **labels** (*str, optional*) – The label to represent in plot legend. If a list of series is provided, a list of labels can be provided as well. Defauts to: "no labels provided".
- **filename** (*str, optional*) – The file name with which the plot figure will be saved in disk. Defaults to rmsd_individual_chains_one_subplot.pdf. You can change the file type by specifying its extention in the file name.
- **fig_size** (*tuple of float or int*) – The size ratio of the subplot in the figure.

Plot a single parameter.

taurenmd.plots.labeldots.**plot**(*y_data*, *x_labels=None*, *x_label_rot=90*, *labels=None*, *ymax=None*, *title=None*, *xlabel=None*, *ylabel=None*, *legend=True*, *legend_fs=6*, *legend_loc=4*, *colors=('b', 'g', 'r', 'c', 'm', 'y', 'k')*, *alpha=0.7*, *grid=True*, *grid_color='lightgrey'*, *grid_ls='-'*, *grid_lw=1*, *grid_alpha=0.5*, *figsize=(10, 6)*, *filename='plot_param.pdf'*, *dpi=300*)

Plot label dots template.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

L

M

P

R

S

T