OMMProtocol Documentation

Release 0.1.13+5.g3105cd3.dirty

Jaime RGP, InsiliChem

Jun 18, 2021

For users

1 Installation & Updates 3

2 Quick usage 5

3 OMMProtocol input files 9

4 OpenMM forcefields 15

5 Software architecture 19

6 Get help 21

7 Citation 23

A command line application to launch molecular dynamics simulations with OpenMM

- No coding required - just a YAML input file!
- **Smart support for different input file formats:**
  - **Topology**: PDB/PDBx, Mol2, Amber's PRMTOP, Charmm's PSF, Gromacs' TOP, Desmond's DMS
  - **Positions**: PDB, COOR, INPCRD, CRD, GRO, DCD
  - **Velocities**: PDB, VEL
  - **Box vectors**: XSC, CSV, PDB, GRO, INPCRD, DCD
  - A fallback method is implemented and will attempt to load everything else that might be supported by ParmEd.
- Choose your preferred **trajectory** format (PDB, PDBx, DCD, HDF5, NETCDF, MDCRD) and **checkpoints** (Amber's, CHARMM's, OpenMM XML states).
- Live report of simulation progress, with estimated ETA and speed.
- Checkpoint every *n* steps. Also, emergency rescue files are created if an error occurs.
- Autochunk the trajectories for easy handling.

CHAPTER 1

Installation & Updates

1.1 How to install OMMProtocol

1.1.1 First method: Standalone installer

If you haven't used Anaconda or Miniconda before (a Python distribution with a cool package manager), your best bet is to simply download the installer for the latest release, which includes everything you need.

1. Go to the OMMProtocol releases page and download the latest installer for your platform.
2. **Run the installer and follow the instructions!**
    a. In Linux, open the terminal and run `bash ~/Downloads/ommprotocol*.sh` or whatever path the file got saved.
    b. In Windows, double click on the downloaded `ommprotocol*.exe`.
3. The installer will create, by default, a new directory called `ommprotocol` in your `$HOME`. Under `ommprotocol/bin`` (Linux) or ``ommprotocol/Scripts` (Windows) you will find the `ommprotocol` executable.

1.1.2 Second method: Conda package

OMMProtocol is also distributed as a separate `conda` package. If you already have Anaconda/Miniconda installed, you won't probably want to download a separate Python distribution. In that case, skip to step 2.

1. Download and install Miniconda, a tiny Python distribution with a cool package manager and installer. Check its webpage for more info.

```
For Linux:

wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
bash Miniconda3*.sh
```

(continues on next page)

(continued from previous page)

```
For Windows, download the EXE installer:
https://repo.continuum.io/miniconda/Miniconda3-latest-Windows-x86_64.exe
```

2. Install it in the default environment…

```
conda install -c omnia -c insilichem ommprotocol
```

3. … or use a new, separate environment (optional):

```
conda create -n ommprotocol -c omnia -c insilichem ommprotocol
conda activate ommprotocol
```

4. If everything is OK, these sould run correctly.

```
ommprotocol -h
ommanalyze -h
```

1.1.3 Third method: From source

If there's no package for your platform, install the dependencies with `conda` and then install `ommprotocol` from pip or source.

```
conda create -n ommprotocol -c omnia openmm ruamel_yaml parmed openmoltools mdtraj␣
↪netcdf4 jinja2 pdbfixer
conda activate ommprotocol
# stable version
pip install ommprotocol
# dev version
pip install https://github.com/insilichem/ommprotocol/archive/master.zip
```

1.2 Updating OMMProtocol

Depending on the installation method, updating OMMProtocol involves different steps.

1.2.1 First method: Standalone installer

Just download the installer for the new version and run it. In Linux/MacOS you will need to append the `-u` flag to the installer. In Windows, just follow the wizard.

1.2.2 Second method: Conda package

Within the activated environment, run `conda update -c insilichem -c omnia ommprotocol`. That's it.

1.2.3 Third method: From source

Simply pass the `-U` flag to pip: `pip install -U ommprotocol` or, for development version, `pip install -U https://github.com/insilichem/ommprotocol/archive/master.zip`.

CHAPTER 2

Quick usage

2.1 OMMProtocol

Once installed (*Installation & Updates*), first thing to do is creating the input file (*OMMProtocol input files*) for your simulation. This task usually involve two different steps:

1. Getting the structural data (topology, coordinates)
2. **Specifying the simulation details:**
    - Forcefield parameters
    - Solvation conditions: explicit, implicit, no solvent?
    - Simulation conditions: temperature, pressure, NPT, NVT?
    - Technical details: how to compute non-bonded interactions, whether to use periodic boundary conditions, whether to constrain some specific types of bonds, the integration method and timestep. . .

These details are probably out of the scope of this documentation, and the reader is encouraged to read specific tutorials about this, such as:

- An Introduction to Molecular Dynamics Simulations using AMBER
- NAMD tutorials
- GROMACS tutorials

With a correctly formed YAML input file named, for example, *simulation.yaml*, the user can now run:

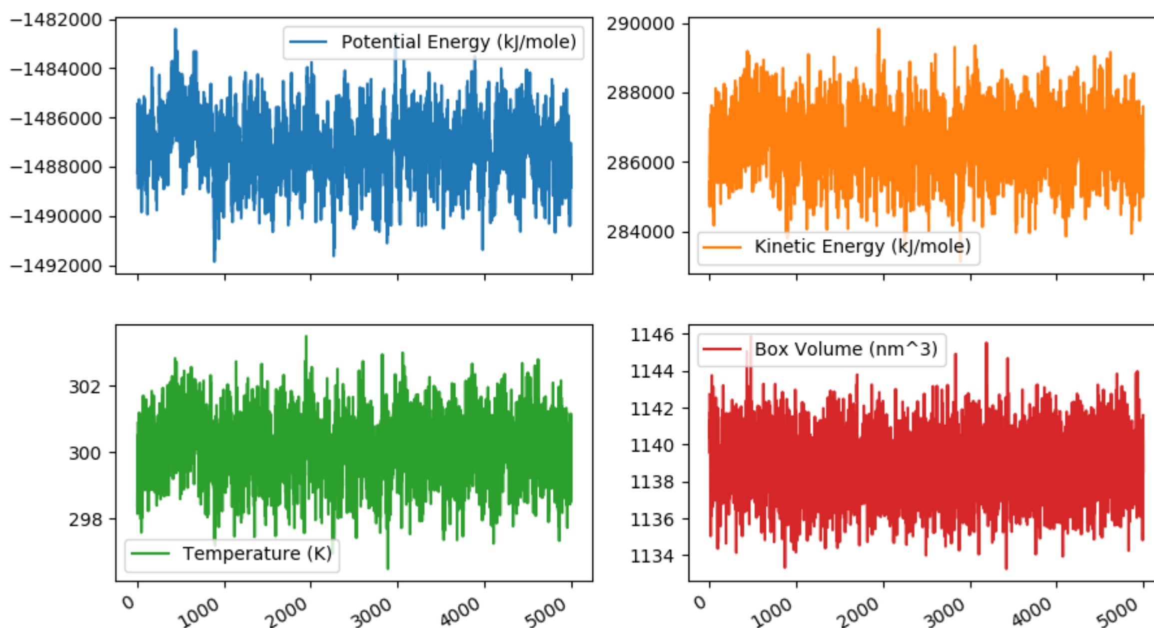
```
ommprotocol simulation.yaml
```

If the structure is correctly formed and the forcefield parameters are well defined, the screen will now display a status like this:

The generated files will be written to the directory specified in the `outputpath` key (or, if omitted, to the same directory *simulation.yaml* is in), with the following name format: `[globalname]_[stagename].[extension]`,

where globalname is the value of the global name key in the input file, and stagename is the value of the stage key in each stage.

Most of this files can be opened during the simulation. That way you can check the progress of the trajectory in viewers like VMD, PyMol or UCSF Chimera. Since .log files are created by default with some metadata about the simulation (temperature, potential energy, volume…), they are a convenient way of checking if everything is working OK. For example, the energies and temperatures should be more or less constant. To do that, a helper utility called ommanalyze is included, which is able to produce interactive plots of such properties:



2.2 OMMAnalyze

ommanalyze is a small collection of analysis utilities for topologies and trajectories. Currently, it offers three subcommands:

- ommanalyze log: Plot *.log reports generated by OMMProtocol (energies, temperature, volume…).
- ommanalyze rmsd: Performs RMSD analysis on trajectories. Produces plots and plain-text file. Since it uses MDTraj iterload, it does not load all the files at once and allows you to analyze long trajectories with small memory footprint.
- ommanalyze top: Quick summary of any topology supported by MDTraj. Designed to debug subset selection queries with --subset flag. Check mdinspect (provided by MDTraj) for more advanced debugging.

2.2.1 Examples

```
> ommanalyze rmsd -t my_topology.prmtop -s 'backbone' my_trajectories*.dcd

 0%|                                                          | 0/26 [00:00<?, ?file/s]
40%|       | 200/500 [00:02<00:03, 77.95frames/s, traj=my_trajectories_01.dcd]
```

(continues on next page)

(continued from previous page)

```
> ommanalyze top my_topology.prmtop -s 'resname UNK'

Topology my_topology.prmtop
***
Contents:
- 1 chains
- 39956 residues
- 132199 atoms
- 132463 bonds
***

Subset `resname UNK` will select 13 atoms.
    name element  resSeq resName  chainID segmentID
14550  H160       H     887     UNK        0
14551  C145       C     887     UNK        0
14552  H159       H     887     UNK        0
14553  C144       C     887     UNK        0
14554  H157       H     887     UNK        0
14555  H158       H     887     UNK        0
14556   N30       N     887     UNK        0
14557  C143       C     887     UNK        0
14558  H156       H     887     UNK        0
14559  C142       C     887     UNK        0
14560  C141       C     887     UNK        0
14561  H154       H     887     UNK        0
14562  H155       H     887     UNK        0
14563   O66       O     887     UNK        0
```

CHAPTER 3

OMMProtocol input files

OMMProtocol is designed with a strong focus on reproducibility. As a result, the input file contains all the necessary details to run a whole simulation. OMMProtocol input files are written with Jinja-enhanced YAML files and look like this:

```
# Protocol for implicit solvent (implicit.yaml)

# input
topology: example.pdb
forcefield: [amber99sbildn.xml, amber99_obc.xml] # only for PDB

# output
project_name: sys
outputpath: output
report: True
report_every: 1000
trajectory: DCD
trajectory_every: 2000
trajectory_new_every: 1e6
restart: rs
restart_every: 1e6
save_state_at_end: True

# hardware
platform: CUDA
platform_properties:
    Precision: mixed

# conditions
integrator: LangevinIntegrator
temperature: 300
friction: 0.1
timestep: 1.0
barostat: False
pressure: 1.01325
```

(continues on next page)

(continued from previous page)

```
barostat_interval: 100
minimization_max_iterations: 1000

# OpenMM system options
nonbondedMethod: CutoffNonPeriodic
nonbondedCutoff: 1.0 # nm
constraints: HBonds
rigidWater: True
extra_system_options:
    implicitSolvent: GBn2

stages:
-   name: implicit
    temperature: 300 # K
    minimization: True
    md_steps: 250e6
    trajectory: DCD
    trajectory_step: 2000
```

There's two main parts in these files:

- Top-level parameters: listed in next section, they are common for all stages
- `stages`: Contains a list with all the stages to be simulated in the requested order. Each stage can override one or more global parameter(s), if needed.

3.1 Provided examples

OMMProtocol ships with some ready-to-use example protocols, which can be used as a template to create a custom one. Most of the time you will only need to change the `topology` and `positions` keys, detailed in the next section. Available examples:

- standard.yaml: The protocol used in most of our solvated protein simulations, such as in [Lur]. It includes a progressive solvent relaxation step, followed by a simulated annealing from 100 to 300K, ending with a long production stage.
- standard_jinja.yaml: Same as previous one, but the simulated annealing stages are described with a Jinja loop for a cleaner result.
- implicit.yaml: Same parameters as `standard.yaml`, but optimized for implicit solvent conditions in a single stage (no need for solvent relaxation).
- test.yaml: Protocol meant to debug a problematic simulation (those that end in `Particle position is NaN`, for example) by dumping states and trajectories every 10 steps. It runs very slow and consumes lots of disk space!
- simple.yaml: Toy example to show the simplest protocol implementable in OMMProtocol.

3.1.1 Default behaviour

In principle, OMMProtocol input files can be as simple as:

```
topology: output.pdb
stages:
```

(continues on next page)

(continued from previous page)

```
- minimization: True
  steps: 100000
```

This is possible due to the chosen default values for almost every key. Specific details for each key are provided below, but globally this results in the following behaviour:

- OMMProtocol will report the simulation progress to the standard output and create an Amber NetCDF checkpoint file every 1,000,000 steps. If an error occurs during the simulation, it will attempt to save a OpenMM XML file with the current state of the simulation, which, if lucky, can be used to restart the simulation or, at least, to debug the problem that could lead to that error.
- If PDB files are being used as topology sources and no forcefield is provided and, it will default to `amber99sbildn.xml` and `tip3p.xml`.

3.2 Top-level parameters

All the parameters are optional except stated otherwise.

3.2.1 Input options

- `topology`: Main input file. Should contain, at least, the topology, but it can also contain positions, velocities, PBC vectors, forcefields… Required. Supports PDB/PDBx, Mol2, Amber's PRMTOP, Charmm's PSF, Gromacs' TOP, Desmond's DMS.
- `positions`: File with the initial coordinates of the system. Overrides those in topology, if needed. Required if the topology does not provide positions. If the file is a trajectory, a frame must be specified with a list: `[ path_to_trajectory.dcd, 1044]`. Supports PDB, COOR, INPCRD, CRD, GRO, DCD.
- `velocities`: File containing the initial velocities of this stage. If not set, they will be set to the requested temperature. Supports PDB, VEL.
- `box_vectors`: File with replacement periodic box vectors, instead of those in the topology or positions file. If the file is a trajectory, a frame must be specified with a list: `[ path_to_trajectory.dcd, 1044]`. Supports XSC, CSV, PDB, GRO, INPCRD, DCD.
- `checkpoint`: Restart simulation from this file. It can provide one or more of the options above. Supports STATE.XML, RS.
- `forcefield`: Which forcefields should be used, if not provided in topology. Required for PDB topologies. More details on *OpenMM forcefields*.
- `charmm_parameters`: CHARMM forcefield. Required for PSF topologies.

Since several types of files can provide the same type of data (positions, vectors…), there is an established order of precedence. `topology < checkpoint < positions & velocities < box`. The only keys out of this chain are``forcefield`` and `charmm_parameters`, which are only required for the specified types of topology.

```
topology  <---------| forcefield (PDB only)
^                   | charmm_parameters (PSF only)
[checkpoint]
^
positions (required if not provided above), [velocities]
^
[box]
```

3.2.2 Output options

- `project_name`: Name for this simulation. Optional. Defaults to a random 5-character string.
- `outputpath`: Path to output folder. If relative, it'll be relative to input file. Optional. Defaults to `.` (directory where the input file is located).
- `report`: True for live report of progress. Defaults to True.
- `report_every`: Update interval of live progress reports. Defaults to 1000 steps.
- `trajectory`: Output format of trajectory file, if desired. Defaults to None (no trajectory will be written).
- `trajectory_every`: Write trajectory every n steps. Defaults to 2000 steps.
- `trajectory_new_every`: Create a new file for trajectory every n steps. Defaults to 1,000,000 steps.
- `restart`: Output format for restart/checkpoint files, if desired. Defaults to None (no checkpoint will be generated).
- `restart_every`: Write restart format every n steps. Defaults to 1,000,000 steps.
- `save_state_at_end`: Whether to save the state of the simulation at the end of every stage. Defaults to True.
- `attempt_rescue`: Try to dump the simulation state into a file if an exception occurs. Defaults to True.

3.2.3 General conditions of simulation

- `minimization`: If *True*, minimize before simulating a MD stage. Defaults to False.
- `steps`: Number of MD steps to simulate. If 0, no MD will take place. Defaults to 0.
- `timestep`: Integration timestep, in fs. Defaults to 1.0.
- `temperature`: In Kelvin. Defaults to 300.
- `barostat`: *True* for NPT, *False* for NVT. Defaults to False.
- `pressure`: In bar. Only used if barostat is *True*. Defaults to 1.01325.
- `barostat_interval`: Update interval of barostat, in steps. Defaults to 25.
- `restrained_atoms`: Parts of the system that should remain restrained (a `k*((x-x0)^2+(y-y0)^2+(z-z0)^2)` force is applied to minimize movement) during the simulation. Supports `mdtraj`'s DSL queries (like `not protein`) or a list of 0-based atom indices (like `[0, 1, 40, 55, 67]`). Default to None (no freezing).
- `restraint_strength`: If restraints are in use, the strength of the applied force in kJ/mol. Defaults to 5.0.
- `distance_restrained_atoms`: Pairs of atoms whose distance should remain constant. Must be specified with a list 2-tuples, with each item being the atom index or a DSL query that returns a single atom. For example, `[[0, 1], [5,2]],[['resid 58 and name OE1', 'resid 43 and name HE1']]`.
- `distance_restraint_length`: Equilibrium distance for each pair of `distance_restrained_atoms`. A list of target values must be provided, one for each pair. If only one value is provided, the same will be used for all pairs. It accepts a positive float (ie, `0.3`) in nm, or the keyword `initial` (to use the starting distance of that pair).
- `distance_restraint_strength`: Force constant for each restrained pair. A list of target values must be provided, one for each pair. If only one value is provided, the same will be used for all pairs. It accepts a positive float (ie, `0.3`) in kcal per mole per squared angstrom.

- `constrained_atoms`: Parts of the system that should remain constrained (no movement at all) during the simulation. Supports `mdtraj`'s DSL queries (like `not protein`) or a list of 0-based atom indices (like `[0, 1, 40, 55, 67]`). Default to None (no freezing).
- `integrator`: Which integrator should be used. Langevin by default.
- `friction`: Friction coefficient for integrator, if needed. In 1/ps. Defaults to 1.0.
- `minimization_tolerance`: Threshold value minimization should converge to. Defaults to 10 kJ/mole.
- `minimization_max_iterations`: Limit minimization iterations up to this value. If zero, don't limit. Defaults to 10000.

3.2.4 OpenMM system parameters

These parameters directly correspond to those used in OpenMM. Their default values will be inherited as a result. For example, if the topology chose is PDB, the system will be created out of the `forcefield` object, whose default values are stated here. For other topologies, check the loaders here.

Most common parameters are summarized here.

- `nonbondedMethod`: The method to use for nonbonded interactions. Choose between *NoCutoff* (default), *CutoffNonPeriodic*, *CutoffPeriodic*, *Ewald*, *PME*.
- `nonbondedCutoff`: The cutoff distance to use for nonbonded interactions, in nm. Defaults to 1.0.
- `constraints`: Specifies which bonds angles should be implemented with constraints. Choose between *None* (default), *HBonds*, *AllBonds*, *HAngles*.
- `rigidWater`: If True (default), water molecules will be fully rigid regardless of the value passed for the constraints argument
- `removeCMMotion`: Whether to remove center of mass motion during simulation. Defaults to *True*.
- `extra_system_options`: A sub-dict with additional keywords that might be supported by the *.createSystem* method of the topology in use. Check the OpenMM docs to know which ones to use.

3.2.5 Hardware options

- `platform`: Which platform to use: *CPU*, *CUDA*, *OpenCL*. If not set, OpenMM will choose the fastest available.
- `platform_properties`: A sub-dict of keyworkds to configure the chosen platform. Check the OpenMM docs to know the supported values. Please notice all values must be strings, even booleans and ints; as a result, you should quote the values like this `'true'`.

CHAPTER 4

OpenMM forcefields

Since OMMProtocol is compatible with a multiple formats thanks to OpenMM itself and other excellent packages (MDTraj, ParmEd…), you can use the forcefield formats defined in other MD suites. Namely, PRMTOP for Amber, PSF+PAR+STR in CHARMM or TOP in Gromacs. If you are already using some of those packages, you don't need to do anything else: just provide the paths in the topology section (CHARMM parameters must be specified in `charmm_parameters`).

However, OpenMM does provide its own set of forcefields, converted from the original formats (Amber, Charmm and others) to its FFXML format. The following section lists all the built in forcefields in OpenMM as of v7.2. The updated list will be available at the OpenMM repo.

```
amber14/DNA.OL15.xml
amber14/DNA.bsc1.xml
amber14/RNA.OL3.xml
amber14/lipid17.xml
amber14/protein.ff14SB.xml
amber14/protein.ff15ipq.xml
amber14/spce.xml
amber14/tip3p.xml
amber14/tip3pfb.xml
amber14/tip4pew.xml
amber14/tip4pfb.xml
charmm36/spce.xml
charmm36/tip3p-pme-b.xml
charmm36/tip3p-pme-f.xml
charmm36/tip4p2005.xml
charmm36/tip4pew.xml
charmm36/tip5p.xml
charmm36/tip5pew.xml
charmm36/water.xml
absinth.xml
amber03.xml
amber03_obc.xml
amber10.xml
amber10_obc.xml
```

(continues on next page)

(continued from previous page)

```
amber14-all
amber96.xml
amber96_obc.xml
amber99Test.xml
amber99_obc.xml
amber99sb.xml
amber99sbildn.xml
amber99sbnmr.xml
amberfb15.xml
amoeba2009.xml
amoeba2009_gk.xml
amoeba2013.xml
amoeba2013_gk.xml
charmm36.xml
charmm_polar_2013.xml
hydrogens.xml
iamoeba.xml
pdbNames.xml
residues.xml
spce.xml
swm4ndp.xml
tip3p.xml
tip3pfb.xml
tip4pew.xml
tip4pfb.xml
tip5p.xml
```

To use them with a PDB file, just specify them in a list for the `forcefield` key, like:

```
topology: some.pdb
forcefield: [amber99sbildn.xml, tip3p.xml]
```

or, if you prefer this other syntax:

```
topology: some.pdb
forcefield:
- amber99sbildn.xml
- tip3p.xml
```

4.1 More forcefields

The OpenMM team is doing a tremendous effort towards the next release, which will include even more forcefields. You can check the progress here. This will include more builtin forcefields and also a separate package called `openmm-forcefields`, developed here. When this is available, it will be shipped with OMMProtocol.

4.2 Custom forcefields

While the best option to generate custom parameters is to use something like AmberTools to create a PRMTOP topology and use that, there are options to develop custom parameters with OpenMM. Check these links for further information:

- Creating and Customizing Force Fields in OpenMM (YouTube video).

- `openmm-forcefields` also features Python converters for Amber & CHARMM forcefields. As a result, automated tooling for those forcefields can be used and then converted to OpenMM, like `antechamber` or `cgenff`.
- openmoltools (included with OMMProtocol) provides some functions to process and convert forcefields. Specifically, `openmoltools.amber.run_antechamber` for parameterizing small molecules through AmberTools' `antechamber`, and `openmoltools.utils.create_ffxml_file` to convert the result to OpenMM XML forcefield format.

CHAPTER 5

Software architecture

OMMProtocol is a glue application, which means that the main business logic resides within the third-party modules it depends on. Nonetheless, this should not necessarily imply a disorganized architecture. The main codebase is clearly divided in two categories: input and output handling (*io* module) and MD settings (*md* module). A third module, utils, collects miscellaneous functions that do not fall within the previous scopes. Finally, code concerning *ommanalyze* is stored in the analyze module.

5.1 Module *io*

This module hosts the input file handling logic, such as the precedence of format files (function *io.prepare_handler*), and the main container class (*io.SystemHandler*) that gives access to the components needed to create an OpenMM System object. Each of those components (*io.Topology*, *io.Positions*, *io.Velocities*, *io.BoxVectors*, and *io.Restart* objects) inherit from *io.MultiFormatLoader*, which supports the automated load of different formats based on the file extension, and io.InputContainer, a simple class that supports different attributes with light validation of the proper data structure.

The custom reporters provided by OMMProtocol are also contained here: *SegmentedDCDReporter* and *ProgressBarReporter*. The first allows the generation of DCD trajectories in chunked files to prevent huge file sizes, and the second converts OpenMM's *StateDataReporter* in a more interactive console reporter (only one line dynamically updated for each protocol stage).

5.2 Module *md*

The goal of this module is to thread together the different stages of the protocol and run the corresponding simulations one after another. The main actor in this module is the *md.Stage* class, which contains all the needed logic to run a simulation in OpenMM: creation of the *System* object, application of restraints or constraints, preparation of the universe conditions such as temperature or pressure, configuration of the platform properties, construction of the Simulation object, setup of the output reporters… Each of these components is encapsulated in cached properties for maximum performance and ease of use in interactive sessions.

A helper function, *md.run_protocol*, takes the options for each stage specified in the input file and builds the needed *Stage* objects to execute them one after the other, passing the final state of each stage as the initial state of the next one. Since each stage must be named uniquely in the input file, the generated output files are meaningfully titled, leading to easy identification during the analysis.

5.3 Module *analyze*

The *ommanalyze* executable provides commands to perform routinary plots in trajectory analysis, like RMSD or potential energy plots. Currently, it only provides two subcommands: `ommanalyze rmsd`, which requires the topology and one or more trajectory files, and outputs an interactive plot with matplotlib and `ommanalyze log`, which simply plots the contents of the .log files generated during the trajectory. This module is only a stub that, if successful, could be further extended with more common analysis procedures thanks to the MDTraj library.

CHAPTER 6

Get help

If you have any questions, please feel free to submit an issue in our Github repository.

CHAPTER 7

Citation

OMMProtocol is scientific software, funded by public research grants (Spanish MINECO's project CTQ2014-54071-P, Generalitat de Catalunya's project 2014SGR989 and research grant 2017FI_B2_00168, COST Action CM1306). If you make use of Ommprotocol in scientific publications, please cite it. It will help measure the impact of our research and future funding! A manuscript is in progress and available as a pre-print in ChemRxiv.

```
@article{ommprotocol,
author    = {Rodríguez-Guerra Pedregal, Jaime and
             Alonso-Cotchico, Lur and
             Velasco-Carneros, Lorea and
             Marechal, Jean-Didier}
title     = {OMMProtocol: A Command Line Application to Launch Molecular Dynamics␣
↪Simulations with OpenMM},
url       = {https://chemrxiv.org/articles/OMMProtocol_A_Command_Line_Application_to_
↪Launch_Molecular_Dynamics_Simulations_with_OpenMM/7059263/1},
DOI       = {10.26434/chemrxiv.7059263.v1}
publisher = {ChemRxiv},
year      = {2018},
month     = {Sep}
}
```