MDpow Documentation

Release 0.7.0+0.g43485dd.dirty

Ian Kenney, Bogdan Iorga, and Oliver Beckstein

Aug 02, 2021

Contents

Release 0.7.0+0.g43485dd.dirty

Date Aug 02, 2021

MDPOW is a python package that automates the calculation of solvation free energies via molecular dynamics (MD) simulations. In particular, it facilitates the computation of partition coeffcients. Currently implemented:

- *water-octanol* partition coefficient (P_{OW})
- *water-cyclohexane* partition coefficient (P_{CW})

Warning: Development is still very much in flux and the documentation is at least partially out of date. If something appears unclear or just wrong, then please ask questions on the MDPOW Issue Tracker.

CHAPTER 1

MD engine

Calculations are performed with the Gromacs molecular dynamics (MD) software package[1]. MDPOW is tested with

- Gromacs 4.6.5
- Gromacs 2018.6
- Gromacs 2020.6
- Gromacs 2021.1

but versions 5.x, 2016.x, and 2019.x should also work. It should be possible to use any of these Gromacs versions without further adjustments, thanks to the underlying GromacsWrapper library[1].

Nevertheless, you should *always* check the topology and runinput (mdp) files for the version of Gromacs that you are using.

[1] The package is built on top of the GromacsWrapper framework (which is automatically installed).

CHAPTER 2

Force fields

Currently

- OPLS-AA
- CHARMM/CGenFF
- AMBER/GAFF

are supported. In principle it is possible to add force fields sets by changing the `GMXLIB` environment variable and providing appropriate template files but this is currently untested.

A number of different *water models* are supporte (see *mdpow.forcefields.GROMACS_WATER_MODELS*).

See also:

mdpow.forcefields

CHAPTER 3

Required input

As *input*, the user only needs to provide a structure file (PDB or GRO) and a Gromacs ITP file containing the parametrization of the small molecule (e.g. from LigandBook or ParamChem).

CHAPTER 4

Version information

MDPOW uses semantic versioning with the release number consisting of a triplet *MAJOR.MINOR.PATCH*. *PATCH* releases are bug fixes or updates to docs or meta data only and do not introduce new features or change the API. Within a *MAJOR* release, the user API is stable except during the development cycles with MAJOR = 0 where the API may also change (rarely) between MINOR releases. *MINOR* releases can introduce new functionality or deprecate old ones.

The version information can be accessed from the attribute *mdpow.__version__*.

mdpow.**__version__ = '0.7.0+0.g43485dd.dirty'**

str(object=”) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

CHAPTER 5

Limitations and known issues

For current issues and open feature requests please look through the MDPOW Issue Tracker. Some of the major open issues are:

- GROMACS versions < 2021 can silently produce incorrect free energy estimates because exclusions are not properly accounted for for solutes larger than the rlist cutoff when the `couple-intramol = no` decoupling is used (as it is in all of MDPOW), see https://gitlab.com/gromacs/gromacs/-/issues/3403. MDPOW does not detect this situation and does not offer a workaround (namely doing separate vacuum simulations and use `couple-intramol = yes`). GROMACS 2021 at least fails when the failure condition occurs (see https://gitlab.com/gromacs/gromacs/-/merge_requests/861).
- Only free energy calculations of neutral solutes are supported; the workflow also does not include addition of ions (see issue #97).
- Mixed solvents (octanol and water) are only supported with the included template topology files for GROMACS >= 2018.
- Adding new solvents requires modifying the MDPOW code; instead it should be configurable.

5.1 Quick installation instructions for *MDPOW*

MDPOW is compatible with Python 2.7 and 3.7+ and tested on Ubuntu and Mac OS. Python 2.7 is rock-stable and frozen but not officially supported anymore by the Python developers.

We recommend that you install MDPOW in a virtual environment.

5.1.1 Installation for users

Most users should follow these instructions.

We use the Anaconda distribution with the `conda` command to manage dependencies (see installing the Anaconda distribution for details on how to set up `conda`).

To run MDPOW you will also need to install a compatible version of GROMACS.

5.1.1.1 Conda environment with pre-requisites

To make a conda environment with the latest packages for Python 2.7 and name it *mdpow*; this installs the larger dependencies that are pre-requisites for MDPOW:

```
conda create -c conda-forge -n mdpow python=2.7 numpy scipy 'matplotlib<3.3'
↪'mdanalysis<2' 'mdanalysistests<2' pyyaml six
conda activate mdpow
pip install gromacswrapper
```

For Python 3.7 and up.

Note: with Pandas version 1.3 there is an error with Alchemlyb (see issue #147) which will be fixed in Alchemlyb 0.5.:

```
conda create -c conda-forge -n mdpow python=3.7 numpy scipy 'matplotlib' 'mdanalysis'
↪'mdanalysistests' pyyaml six
conda activate mdpow
pip install gromacswrapper
```

5.1.1.2 Installation from source

Install from source:

```
conda activate mdpow
git clone https://github.com/Becksteinlab/MDPOW.git
pip install ./MDPOW
```

(Older releases are available but outdated; use the latest source for now.)

5.1.1.3 Checking that the installation worked

Check that you can run the `mdpow-*` commandline tools:

```
mdpow-equilibrium --help
```

should print a whole bunch of messages. If your GROMACS installation cannot be found, make sure you `source GMXRC` or load the appopriate modules or whatever else you have to do so that the `gmx` command is found. See https://gromacswrapper.readthedocs.io for more details.

Check that you can import the module:

```
python
>>> import mdpow
>>> help(mdpow)
```

In case of problems file an issue at https://github.com/Becksteinlab/MDPOW/issues

5.1.2 Developer installation

A development install is useful while hacking away on the code:

```
cd MDPOW
pip install -e  .
```

5.2 mdpow — Computing the octanol/water partitioning coefficient

The mdpow module helps in setting up and analyzing absolute free energy calculations of small molecules by molecular dynamics (MD) simulations. By computing the hydration free energy and the solvation free energy in octanol one can compute the octanol/water partitioning coefficient, an important quantity that is used to characterize drug-like compounds.

The MD simulations can be performed with all recent versions of Gromacs, starting with Gromacs 4.6.x.

5.2.1 How to use the module

Before you can start you will need

- a coordinate file for the small molecule
- a Gromacs OPLS/AA topology (itp) file
- an installation of Gromacs in a *supported version*.

5.2.1.1 Basic work flow

You will typically calculate two solvation free energies (free energy of transfer of the solute from the liquid into the vacuum phase):

1. solvent = *water*
    1. set up a short equilibrium simulation of the molecule in a *water* box (and run the MD simulation);
    2. set up a free energy perturbation calculation of the ligand in water , which will yield the hydration free energy;
2. solvent = *octanol*
    1. set up a short equilibrium simulation of the molecule in a *octanol* box (and run the MD simulation);
    2. set up a free energy perturbation calculation of the ligand in octanol , which will yield the solvation free energy in octanol;
3. run these simulations on a cluster;
4. analyze the output and combine the free energies to arrive at an estimate of the octanol-water partition coefficient;
5. plot results using `mdpow.analysis.plot_exp_vs_comp()`.

5.2.1.2 Customized submission scripts for queuing systems

One can also generate run scripts for various queuing systems; check the documentation for `gromacs.qsub` and in particular the section on writing queuing system templates . You will have to

- add a template script to your private GromacsWrapper template directory (`~/.gromacswrapper/qscripts`); in this example we call it `my_script.sge`;
- add the keyword *qscript* to the *`mdpow.equil.Simulation.MD()`* and `mdpow.fep.Gsolv.setup()` invocations; e.g. as

  qscript = [‘my_script.sge’, ‘local.sh’]
- submit the generated queuing system script to your queuing system, e.g.

```
cd Equilibrium/water
qsub my_script.sh
```

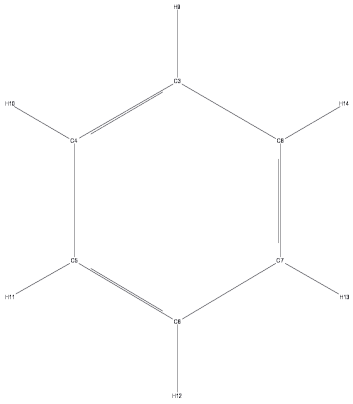
5.2.2 The mdpow scripts

Some tasks are simplified by using scripts, which are installed in a bin directory (or the directory pointed to by `python setup.py --install-scripts`). See *The mdpow-* scripts* for details and *Tutorial: Using the mdpow scripts to compute log POW of benzene* for an example. The scripts essentially execute the steps shown in *Tutorial: Manual session: 1-octanol as a solute* so in order to gain a better understanding of MDpow it is suggested to look at both tutorials.

5.2.3 Tutorial: Using the mdpow scripts to compute log P_{OW} of benzene

The most straightforward use of MDpow is through the Python scripts described in *The mdpow-* scripts*. This tutorial shows how to use them to calculate log P_{OW} for benzene.

The `examples/benzene` directory of the distribution contains a Gromacs OPLS/AA topology for benzene (`benzene.itp`) and a starting structure (`benzene.pdb`) together with a *run input configuration file* for the mdpow scripts (`benzene.yml`) in YAML format.



1. Make a directory `benzene` and collect the files in the directory.
2. Move into the parent directory of `benzene`; in this tutorial all files will be generated under `benzene` but the configuration file has recorded the location of the input files as `benzene/filename`. (For more on to make best use of file names in the configuration file see *Advanced: Separating input from output data*.)
3. Set up Gromacs (e.g. by sourcing `GMXRC`).

   Note: Gromacs must be set up so that all Gromacs commands (such as **`mdrun`** or **`grompp`** can be found on the shell's `PATH`). If this is not the case then the following will fail. Look at the **log file** (named **mdpow.log**) for error messages.)

4. Generate the input files for a equilibrium simulation of benzene in **water** (TIP4P) and run the simulation:

   ```
   mdpow-equilibrium --solvent water benzene/benzene.yml
   ```

   The example file provided will only set up very short simulations and it will also directly call **`mdrun`** to run the simulations. This is configured via the `runlocal = True` parameter in the `MD_relaxed` and `MD_NPT` sections in the `benzene.yml` file.

In fact, the following steps are carried out:

1. generate a system topology (section `Setup`)
2. solvate the compound (section `Setup`)
3. energy minimise the system (section `Setup`)
4. run a short relaxation with a time step of 0.1 fs in order to remove any steric clashes; this step was found to enable much more robust simulations, especially when using octanol as a solvent. Section `MD_relaxed` controls this step.
5. run a equilibrium MD simulation at constant pressure and temperature using a time step of 2 fs. Section `MD_NPT` controls this step.

5. Generate the input files for a equilibrium simulation of benzene in **octanol** (OPLS/AA parameters) and run the simulation:

```
mdpow-equilibrium --solvent octanol benzene/benzene.yml
```

The steps are analogous to the ones described under 4.

6. Use the last frame of the equilibrium simulation as a starting point for the free energy perturbation (FEP) calculations. This step is controlled by the `FEP` section in the run input configuration file.

The example only runs very short windows (`runtime: 25` ps) but it will run *all* 21 individual simulations sequentially (`runlocal: True`). Thus it is recommended to run this example on a fast multi-core workstation and at least Gromacs 4.5.x, which has thread support for **`mdrun`**.

The FEP windows for benzene in **water** are generated and run by

```
mdpow-fep --solvent water benzene/benzene.yml
```

(In order to run the FEP windows on a cluster see *Advanced: Generating queuing system scripts*.)

7. The FEP windows for benzene in **octanol** are generated and run by

```
mdpow-fep --solvent octanol benzene/benzene.yml
```

8. Analyse the simulation output with *mdpow-pow*. It collects the raw data from the FEP simulations and computes the free energies of solvation using thermodynamic integration (TI) together with error estimates. log P_{OW} is calculated from the difference of the octanol and water solvation free energies:

```
mdpow-pow benzene
```

`benzene` is the directory name under which the FEP simulations are stored. By default, results are *appended* to the files `energies.txt` and `pow.txt`.

See also:

The output formats are explained with examples in *Output data file formats*.

5.2.3.1 Advanced: Generating queuing system scripts

(To be written)

See also:

`gromacs.qsub` for the reference on how queuing system script templates are handled and processed

5.2.3.2 Advanced: Separating input from output data

Make another directory under which simulation data will be stored; in this tutorial it will be called `WORK` and we assume that all directories reside in `~/Projects/POW`. We will end up with a directory layout under `~/Projects/POW` like this:

```
benzene/
       benzene.itp
       benzene.pdb
       benzene.yml
WORK/
     benzene/
             water/
             octanol/
```

Edit `benzene.yml` to put in *absolute paths* to the input files: In the `setup` section use absolute paths to the itp and pdb files of benzene:

```
setup:
  name: "benzene"
  molecule:  "BNZ"
  structure: "~/Projects/POW/benzene/benzene.pdb"
  itp: "~/Projects/POW/benzene/benzene.itp"
```

With absolute paths defined, it is easy to generate all other files under `WORK/benzene` (the directory name "benzene" is the *name* entry from the `setup` section of the configuration file):

```
cd WORK
mdpow-equilibrium --solvent water ../benzene/benzene.yml
mdpow-equilibrium --solvent octanol ../benzene/benzene.yml
mdpow-fep --solvent water ../benzene/benzene.yml
mdpow-fep --solvent octanol ../benzene/benzene.yml
```

Finally, calculate log P_{OW}:

```
cd WORK
mdpow-pow benzene
```

5.2.4 Tutorial: Manual session: 1-octanol as a solute

In the following interactive python session we use octanol as an example for a solute; all files are present in the package so one can work through the example immediately.

Before starting **`python`** (preferrably ipython) make sure that the Gromacs 4.6.x tools can be found, e.g. `which grompp` should show you the path to **`grompp`**.

5.2.4.1 Water

Equilibrium simulation

Make a directory `octanol` and copy the `octanol.itp` and `octanol.gro` file into it. Launch **`ipython`** from this directory and type:

```
import mdpow.equil
S = mdpow.equil.WaterSimulation(molecule="OcOH")
S.topology(itp="octanol.itp")
S.solvate(struct="octanol.gro")
S.energy_minimize()
S.MD_relaxed()
# run the simulation in the MD_relaxed/ directory
S.MD(runtime=50, qscript=['my_script.sge', 'local.sh'])  # only run for 50 ps in this␣
↪tutorial
S.save("water.simulation")                               # save setup for later␣
↪(analysis stage)
```

Background (`Ctrl-Z`) or quit (`Ctrl-D`) python and run the simulations in the `MD_relaxed` and `MD_NPT` subdirectory. You can modify the `local.sh` script to your ends or use *qscript* to generate queuing system scripts.

Note: Here we only run 50 ps equilibrium MD for testing. For production this should be substantially longer, maybe even 50 ns if you want to extract thermodynamic data.

Hydration free energy

Reopen the python session and set up a `Ghyd` object:

```
import mdpow.fep
gwat = mdpow.fep.Ghyd(molecule="OcOH", top="Equilibrium/water/top/system.top", struct=
↪"Equilibrium/water/MD_NPT/md.pdb", ndx="Equilibrium/water/solvation/main.ndx",␣
↪runtime=100)
```

Alternatively, one can save some typing if we continue the last session and use the *mdpow.equil.Simulation* object (which we can re-load from its saved state file from disk):

```
import mdpow.equil
S = mdpow.equil.WaterSimulation(filename="water.simulation")  # only needed when quit
gwat = mdpow.fep.Ghyd(simulation=S, runtime=100)
```

This generates all the input files under `FEP/water`.

Note: Here we only run 100 ps per window for testing. For production this should be rather something like 5-10 ns (the default is 5 ns).

Then set up all input files:

```
gwat.setup(qscript=['my_script.sge', 'local.sh'])
```

(The details of the FEP runs can be customized by setting some keywords (such as *lambda_vdw*, *lamda_coulomb*, see `mdpow.fep.Gsolv` for details) or by deriving a new class from the `mdpow.fep.Ghyd` base class but this is not covered in this tutorial.)

5.2.4.2 Octanol

Equilibrium simulation

Almost identical to the water case:

```
O = mdpow.equil.OctanolSimulation(molecule="OcOH")
O.topology(itp="octanol.itp")
O.solvate(struct="octanol.gro")
O.energy_minimize()
O.MD_relaxed()
O.MD(runtime=50, qscript=['my_script.sge', 'local.sh'])     # only run for 50 ps in␣
↪this tutorial
O.save()
```

Note: Here we only run 50 ps equilibrium MD for testing. For production this should be substantially longer, maybe even 50 ns if you want to extract thermodynamic data.

Octanol solvation free energy

Almost identical setup as in the water case:

```
goct = mdpow.fep.Goct(simulation=O, runtime=100)
goct.setup(qscript=['my_script.sge', 'local.sh'])
```

This generates all the input files under `FEP/octanol`.

Note: Here we only run 100 ps per window for testing. For production this should be rather something like 5-10 ns (the default is 5 ns)

5.2.4.3 Running the FEP simulations

The files are under the `FEP/water` and `FEP/octanol` directories in separate sub directories.

Either run job arrays that should have been generated from the `my_script.sge` template

```
qsub Coul_my_script.sge
qsub VDW_my_script.sge
```

Or run each job in its own directory. Note that **`mdrun`** should be called with at least the following options

```
mdrun -deffnm $DEFFNM  -dgdl
```

where DEFFNM is typically "md"; see the run `local.sh` script in each direcory for hints on what needs to be done.

5.2.4.4 Analyze output and log P_{OW} calculation

For the water and octanol FEPs do

```
gwat.collect()
gwat.analyze()

goct.collect()
goct.analyze()
```

The analyze step reports the estimate for the free energy difference.

Calculate the free energy for transferring the solute from water to octanol and *octanol-water partition coefficient* log P_OW

```
mdpow.fep.pOW(gwat, goct)
```

(see `mdpow.fep.pOW()` for details and definitions).

All individual results can also accessed as a dictionary

```
gwat.results.DeltaA
```

Free energy of transfer from water to octanol:

```
goct.results.DeltaA.Gibbs - gwat.results.DeltaA.Gibbs
```

The individual components are

Gibbs total free energy difference of transfer from solvent to vacuum at the Ben-Naim standard state (i.e. 1M solution/1M gas phase) in kJ/mol;

DeltaG0 = (G_solv - G_vac)

We calculate the Gibbs free energy (at constant pressure P) by using the *NPT* ensemble for all MD simulations.

coulomb contribution of the de-charging process to DeltaG

vdw contribution of the de-coupling process to DeltaG

To plot the data (de-charging and de-coupling):

```
import pylab
gwat.plot()
pylab.figure()
goct.plot()
```

For comparison to experimental values see `mdpow.analysis`.

5.2.4.5 Error analysis

The data points are the (time) **average <G>** of G = dV/dl over each window. The **error bars** s_G are the error of the mean <G>. They are computed from the auto-correlation time of the fluctuations and the standard deviation (see Frenkel and Smit, p526 and `numkit.timeseries.tcorrel()`):

```
s_G**2 = 2*tc*acf(0)/T
```

where tc is the decay time of the ACF of <(G-<G>)**2> (assumed to follow f(t) = exp(-t/tc) and hence calculated from the integral of the ACF to its first root); T is the total runtime.

Errors on the energies are calculated via the propagation of the errors s_G through the thermodynamic integration and the subsequent thermodynamic sums (see `numkit.integration.simps_error()` `numkit.observables.QuantityWithError` for details).

- If the graphs do not look smooth or the errors are large then a longer *runtime* is definitely required. It might also be necessary to add additional lambda values in regions where the function changes rapidly.
- The errors on the Coulomb and VDW free energies should be of similar magnitude because there is no point in being very accurate in one if the other is inaccurate.
- For water the "canonical" lambda schedule produces errors <0.5 kJ/mol (or sometimes much better) in the Coulomb and VDW free energy components.
- For octanol the errors on the coulomb dis-charging free energy can become large (up to 4 kJ/mol) and thus completely swamp the final estimate. Additional lambdas 0.125 and 0.375 should improve the precision of the calculations.

5.3 The `mdpow-*` scripts

A number of python scripts are installed together with the `mdpow` package. They simplify some common tasks (especially at the analysis stage) but they make some assumptions about directory layout and filenames. If one uses defaults for all directory and filename options then it should "just work".

In particular, a directory hierarchy such as the following is assumed:

```
moleculename/
     water.simulation
     octanol.simulation
     Equilibrium/
           water/
           octanol/
     FEP/
           water/
                 Ghyd.fep
                 Coulomb/
                 VDW/
           octanol/
                 Goct.fep
                 Coulomb/
                 VDW/
```

moleculename is, for instance, "benzene" or "amantadine"; in the run input file (see *Equilibrium simulations*) is the value of the variable `name` in the `[setup]` section.

5.3.1 Equilibrium simulations

The **`mdpow-equilibrium`** script

- sets up equilibrium MD simulations for the solvents (e.g., *water* or *octanol*)
- runs **energy minimization**, **MD_relaxed**, and **MD_NPT** protocols; the user can choose if she wants to launch **`mdrun`** herself (e.g. on a cluster) or let the script do it locally on the workstation

The script runs essentially the same steps as described in the tutorial *Tutorial: Manual session: 1-octanol as a solute* but it gathers all required parameters from a run input file and it allows one to stop and continue and the protocol transparently.

It requires as at least Gromacs 4.6.5 ready to run (check that all commands can be found). The required **input** is

1. a run configuration file (*runinput.yml*);
2. a structure file (PDB or GRO) for the compound

3. a Gromacs ITP file for the compound (OPLS/AA force field)

A template *RUNFILE* can be generated with **mdpow-get-runinput** `runinput.yml`, which will copy the default run input file bundled with MDPOW and put it in the current directory under the name `runinput.yml`.

For an example of a *RUNFILE* see `benzene.yml`. It is recommended to use absolute paths to file names. The run input file uses YAML syntax (and is parsed by `yaml`).

Note: It is recommended to enclose all strings in the input file in quotes, especially if they can be interpreted as numbers. For example, a name "005" would be interpreted as the number 5 unless explicitly quoted.

The script keeps track of the stages of the simulation protocol (in the state files `water.simulation`, `octanol.simulation` etc) and allows the user to **restart from the last completed stage**. For instance, one can use the script to set up a simulation, then run the simulation on a cluster, transfer back the generated files, and start **mdpow-equilibrium** again with the exact same input to finish the protocol. Since Gromacs 4.5 it is also possible to interrupt a running **mdrun** process (e.g. with `Control-c`) and then resume the simulation at the last saved trajectory checkpoint by running **mdpow-equilibrium** again.

If in doubt, just try running **mdpow-equilibrium** running again and let it figure out the best course of action. Look at the log file to see what has been done. Lines reading "Fast forwarding: *stage*" indicate that results from *stage* are available.

Usage of the command:

> **mdpow-equilibrium** [options] *RUNFILE*
>
> Set up (and possibly run) the equilibration equilibrium simulation for one compound and one solvent. All parameters except the solvent are specified in the *RUNFILE*.
>
> Arguments:
>
> **RUNFILE**
> The runfile `runinput.yml` with all configuration parameters.
>
> Options:
>
> **-h, --help**
> show this help message and exit
>
> **-S** <NAME>, **--solvent**=<NAME>
> solvent <NAME> for compound, can be 'water', 'octanol', 'cyclohexane' [water]
>
> **-d** <DIRECTORY>, **--dirname**=<DIRECTORY>
> generate files and directories in <DIRECTORY>, which is created if it does not already exist. The default is to use the molecule *name* from the run input file.

5.3.2 FEP simulations

The **mdpow-fep** script sets up (and can also run) the free energy perturbation calculations for one compound and one solvent. It uses the results from *mdpow-equilibrium* together with the run input file.

You will require:

1. at least Gromacs 4.0.5 ready to run (check that all commands can be found)
2. the results from a previous complete run of **mdpow-equilibrium**

Usage of the command:

mdpow-fep [options] *RUNFILE*

Arguments:

RUNFILE
The runfile `runinput.yml` with all configuration parameters.

Options:

-h, --help
show this help message and exit

-S <NAME>, **--solvent**=<NAME>
solvent <NAME> for compound, can be 'water' or 'octanol' [water]

-d <DIRECTORY>, **--dirname**=<DIRECTORY>
generate files and directories in <DIRECTORY>, which is created if it does not already exist. The default is to use the molecule *name* from the run input file.

5.3.3 Running analysis

5.3.3.1 Solvation free energy

The **mdpow-solvationenergy** calculatest the solvation free energy. It

- collects data from FEP simulations (converting EDR files to XVG.bz2 when necessary)
- calculates desolvation free energies for *solvent* –> vacuum via thermodynamic integration (TI)
- plots dV/dlambda graphs
- appends results to `energies.txt` (when the default names are chosen), see *Output data file formats*.

Usage of the command:

mdpow-solvationenergy [options] DIRECTORY [DIRECTORY ...]

Run the free energy analysis for a solvent in <DIRECTORY>/FEP and return DeltaG.

DIRECTORY should contain all the files resulting from running `mdpow.fep.Ghyd.setup()` (or the corresponding `Goct.setup()` or `Gcyclohexane.setup()` and the results of the MD FEP simulations. It relies on the canonical naming scheme (basically: just use the defaults as in the tutorial).

The dV/dlambda plots can be produced automatically (`--plotfile auto`). If multiple DIRECTORY arguments are provided then one has to choose the "auto" option (or "none").

The total solvation free energy is calculated as

$$\Delta G^* = -(\Delta G_{\mathrm{coul}} + \Delta G_{\mathrm{vdw}})$$

Note that the standard state refers to the "Ben-Naim" standard state of transferring 1 M of compound in the gas phase to 1 M in the aqueous phase.

Results are *appended* to a data file with **Output file format**:

```
.                  ---------- kJ/mol ---
molecule solvent  DeltaG*  coulomb  vdw
```

All observables are quoted with an error estimate, which is derived from the correlation time and error propagation through Simpson's rule (see `mdpow.fep.Gsolv()`). It ultimately comes from the error on <dV/dlambda>, which is estimated as the error to determine the average.

molecule molecule name as used in the itp

DeltaG* solvation free energy vacuum –> solvent, in kJ/mol

coulomb discharging contribution to the DeltaG*

vdw decoupling contribution to the DeltaG*

directory folder in which the simulations were stored

positional arguments:

DIRECTORY [DIRECTORY ...]
directory or directories which contain all the files resulting from running `mdpow.fep.Ghyd.setup()`

optional arguments:

-h, --help
show this help message and exit

--plotfile FILE
plot dV/dlambda to FILE; use png or pdf suffix to determine the file type. *'auto'* generates a pdf file *DIRECTORY*/dVdl_*molname_solvent*.pdf and *none* disables it The plot function is only available for mdpow estimator,and is disabled when using alchemlyb estimators. (default: none)

--solvent NAME, **-S** NAME
solvent *NAME* for compound, 'water', 'octanol', or 'cyclohexane' (default: water)

-o FILE, **--outfile** FILE
append one-line results summary to *FILE* (default: `gsolv.txt`)

-e FILE, **--energies** FILE
append solvation free energies to FILE (default: `energies.txt`)

--estimator {mdpow,alchemlyb}
choose the estimator to be used, *alchemlyb* or *mdpow* estimators (default: *alchemlyb*)

--method {TI,MBAR,BAR}
choose the method to calculate free energy (default: *MBAR*)

--force
force rereading all data (default: False)

--SI
enable statistical inefficiency (SI) analysis. Statistical inefficiency analysis is performed by default when usingalchemlyb estimators and is always disabled when using mdpow estimator. (default: True)

--no-SI
disable statistical inefficiency analysis. Statistical inefficiency analysis is performed by default when usingalchemlyb estimators and is disabled when using mdpow estimator. (default: False)

-s N, **--stride** N
use every *N*-th datapoint from the original dV/dlambda data. (default: 1)

--start START
start point for the data used from the original dV/dlambda data. (default: 0)

--stop STOP
stop point for the data used from the original dV/dlambda data. (default: None)

--ignore-corrupted
skip lines in the md.xvg files that cannot be parsed. (default: False)

> **Warning:** Other lines in the file might have been corrupted in such a way that they appear correct but are in fact wrong. WRONG RESULTS CAN OCCUR! USE AT YOUR OWN RISK

It is recommended to simply rerun the affected simulation(s).

5.3.3.2 Partition coefficients

The **mdpow-pow** script

- collects data from FEP simulations
- calculates desolvation free energies for octanol –> vacuum and water –> vacuum via thermodynamic integration (TI)
- calculates transfer free energy water –> octanol
- calculates log P_{OW}
- plots dV/dlambda graphs
- appends results to `pow.txt` and `energies.txt` (when the default names are chosen), see *Output data file formats*.

An equivalent script **mdpow-pcw** for the water-cyclohexane partition coefficient is also included.

Usage of the command:

mdpow-pow [options] *DIRECTORY* [*DIRECTORY* ...]

Run the free energy analysis for water and octanol in *DIRECTORY*/FEP and return the octanol-water partition coefficient log P_{OW}.

DIRECTORY should contain all the files resulting from running `mdpow.fep.Goct.setup()` and `mdpow.fep.Goct.setup()` and the results of the MD FEP simulations. It relies on the canonical naming scheme (basically: just use the defaults as in the tutorial).

The dV/dlambda plots can be produced automatically (`--plotfile auto`). If multiple *DIRECTORY* arguments are provided then one has to choose the "auto" option (or "none").

positional arguments:

DIRECTORY [DIRECTORY ...]
: One or more directories that contain the state pickle files (`water.simulation`, `octanol.simulation`) for the solvation free energy calculations in water and octanol. These directory or directories should contain all the files resulting from running `mdpow.fep.Ghyd.setup()` and `mdpow.fep.Goct.setup()` and the results of the MD FEP simulations.

optional arguments:

-h, --help
: show this help message and exit

--plotfile FILE
: plot dV/dlambda to FILE; use png or pdf suffix to determine the file type. *'auto'* generates a pdf file *DIRECTORY*/dVdl_*molname*_pow.pdf and *none* disables it The plot function is only available for mdpow estimator,and is disabled when using alchemlyb estimators. (default: none)

-o FILE, **--outfile** FILE
: append one-line results summary to *FILE* (default: `pow.txt`)

-e FILE, **--energies** FILE
append solvation free energies to FILE (default: `energies.txt`)

--estimator {mdpow,alchemlyb}
choose the estimator to be used, *alchemlyb* or *mdpow* estimators (default: *alchemlyb*)

--method {TI,MBAR,BAR}
choose the method to calculate free energy (default: *MBAR*)

--force
force rereading all data (default: False)

--SI
enable statistical inefficiency (SI) analysis. Statistical inefficiency analysis is performed by default when usingalchemlyb estimators and is always disabled when using mdpow estimator. (default: True)

--no-SI
disable statistical inefficiency analysis. Statistical inefficiency analysis is performed by default when usingalchemlyb estimators and is disabled when using mdpow estimator. (default: False)

-s N, **--stride** N
use every *N*-th datapoint from the original dV/dlambda data. (default: 1)

--start START
start point for the data used from the original dV/dlambda data. (default: 0)

--stop STOP
stop point for the data used from the original dV/dlambda data. (default: None)

--ignore-corrupted
skip lines in the md.xvg files that cannot be parsed. (default: False)

> **Warning:** Other lines in the file might have been corrupted in such a way that they appear correct but are in fact wrong. WRONG RESULTS CAN OCCUR! USE AT YOUR OWN RISK

It is recommended to simply rerun the affected simulation(s).

5.3.3.3 Output data file formats

Results are *appended* to data files.

Note: Energies are always output in **kJ/mol**.

POW summary file

The `pow.txt` output file summarises the results from the water and octanol solvation calculations. Its name set with the option *`mdpow-pow -o`*. It contains fixed column data in the following order and all energies are in **kJ/mol**. See the *Table of computed water-octanol transfer energies and logPow* as an example.

itp_name

molecule identifier from the itp file

DeltaG0

transfer free energy from water to octanol (difference between **DeltaG0** for octanol and water), in kJ/mol; >0: partitions into water, <0: partitions into octanol,

errDG0

error on **DeltaG0**; errors are calculated through propagation of errors from the errors on the means <dV/dlambda>

logPOW

log P_{OW}, base-10 logarithm of the octanol-water partition coefficient; >0: partitions into octanol, <0: partitions preferrably into water

errlogP

error on **logPow**

directory

directory under which all data files are stored; by default this is the *name* of the molecule and hence it can be used to identify the compound.

Table 1: Computed log P_{OW} and water-to-octanol transfer energies (in kJ/mol).

itp_name	DeltaG0	errDG0	logPow	errlogP	directory
BNZ	-12.87	0.43	+2.24	0.07	benzene
OC9	-16.24	1.12	+2.83	0.20	octanol
URE	+4.66	1.13	-0.81	0.20	urea
902	+9.35	1.06	-1.63	0.18	water_TIP4P

Energy file

The `energy.txt` output file collects all computed energy terms together with the results also found in the summary file `pow.txt`. Its name is set with the option `mdpow-pow -e`. It contains fixed column data in the following order and all energies are in **kJ/mol**. As an example see *Table of Solvation Energies* for the same compounds as above.

molecule

molecule identifier from the itp file

solvent

solvent name (water, octanol)

DeltaG0

solvation free energy difference in kJ/mol (Ben-Naim standard state, i.e. 1M gas/1M solution)

errDG0

error on **DeltaG0**, calculated through propagation of errors from the errors on the mean <dV/dlambda>

coulomb

Coulomb (discharging) contribution to the solvation free energy **DeltaG0**

errCoul

error on **coulomb**

VDW

Van der Waals/Lennard-Jones (decoupling) contribution to **DeltaG0**

errVDW

error on **VDW**

Vdp

correction if the FEP are run at constant volume but the desired solvation free energy is the Gibbs energy (**currently neglected** and set to 0)

errVdp

error on **Vdp**

w2oct

transfer free energy from water to octanol (difference between **DeltaG0** for octanol and water)

errw2oct

error on **w2oct**

logPOW

log P_{OW}

errlogP

error on **logPow**

directory

directory under which all data files are stored; by default this is the *name* of the molecule and hence it can be used to identify the compound.

Table 2: Solvation free energies for compounds in water and octanol (in kJ/mol).

molecule	solvent	DeltaG0	errDG0	coulomb	errCoul	VDW	errVDW	Vdp	errVdp	w2oct	errw2oct	logPOW	errlogP	directory
BNZ	water	-2.97	0.21	+7.65	0.07	-4.68	0.20	+0.00	0.00	-12.87	0.43	+2.24	0.07	benzene
BNZ	octanol	-15.84	0.37	+1.40	0.19	+14.44	0.32	+0.00	0.00	-12.87	0.43	+2.24	0.07	benzene
OC9	water	-16.03	0.32	+27.35	0.09	-11.32	0.31	+0.00	0.00	-16.24	1.12	+2.83	0.20	octanol
OC9	octanol	-32.28	1.08	+11.32	0.92	+20.96	0.56	+0.00	0.00	-16.24	1.12	+2.83	0.20	octanol
URE	water	-53.52	0.17	+56.94	0.11	-3.41	0.14	+0.00	0.00	+4.66	1.13	-0.81	0.20	urea
URE	octanol	-48.86	1.12	+35.75	1.09	+13.11	0.25	+0.00	0.00	+4.66	1.13	-0.81	0.20	urea
902	water	-25.46	0.11	+34.93	0.10	-9.48	0.06	+0.00	0.00	+9.35	1.06	-1.63	0.18	water_TIP4P
902	octanol	-16.11	1.05	+21.16	1.05	-5.05	0.09	+0.00	0.00	+9.35	1.06	-1.63	0.18	water_TIP4P

5.3.4 House-keeping scripts

A number of scripts are provided to complete simple tasks; they can be used to check that all required files are present or they can help in fixing small problems without one having to write Python code to do this oneself (e.g. by

manipulating the checlpoint files). They generally make the same assumptions about file system layout as the other mdpow scripts.

5.3.4.1 Checking if the simulation is complete

Run **mdpow-check** in order to check if all necessary files are available.

Usage of the command:

> **mdpow-check** [options] *DIRECTORY* [*DIRECTORY* ...]
>
> Check status of the progress of the project in *DIRECTORY*.
>
> Output is only written to the status file (`status.txt`). A quick way to find problems is to do a

```
cat status.txt | gawk -F '|' '$2 !~ /OK/ {print $0}'
```

> Options:
>
> **-h, --help**
> show this help message and exit
>
> **-o** <FILE>, **--outfile**=<FILE>
> write status results to *FILE* [`status.txt`]

5.3.4.2 Changing paths in `water.simulation` and `octanol.simulation`

It can become necessary to recreate the *solvent*`.simulation` status/checkpoint files in order to change paths, e.g. when one moved the directories or transferred all files to a different file system.

Typically, one would execute the **mdpow-rebuild-simulation** command in the parent directory of *moleculename*.

Usage of the command:

> **mdpow-rebuild-simulation** [options] *DIRECTORY* [*DIRECTORY* ...]
>
> Re-create the `water.simulation` or `octanol.simulation` file with adjusted paths (now rooted at *DIRECTORY*).
>
> Options:
>
> **-h, --help**
> show this help message and exit
>
> **--solvent**=<NAME>
> rebuild file for 'water', 'octanol', or 'all' [all]

5.3.4.3 Re-building `Ghyd.fep` and `Goct.fep`

It can become necessary to recreate the *name*`.fep` status/checkpoint files (e.g. if the files became corrupted due to a software error or in order to change paths).

Typically, one would execute the **mdpow-rebuild-fep** command in the parent directory of *moleculename*.

Usage of the command:

> **mdpow-rebuild-fep** [options] *DIRECTORY* [*DIRECTORY* ...]
>
> Re-create the `Goct.fep` or `Ghyd.fep` file using the appropriate equilibrium simulation file under *DIRECTORY*/FEP.

This should only be necessary when the fep file was destroyed due to a software error or when the files are transferred to a different file system and some of the paths stored in the *name*`.fep` files have to be changed.

Options:

-h, --help
show this help message and exit

--solvent=<NAME>
rebuild fep for 'water', 'octanol', or 'all' [all]

--setup=<LIST>
Re-generate queuing system scripts with appropriate paths: runs `fep.Gsolv.setup()` with argument *qscript=[LIST]* after fixing Gsolv.

`LIST` should contain a comma-separated list of queing system templates. For example: `'icsn_8pd.sge,icsn_2pd.sge,local.sh'`. It is an error if `--setup` is set without a `LIST`.

5.4 `mdpow.equil` — Setting up and running equilibrium MD

The *mdpow.equil* module facilitates the setup of equilibrium molecular dynamics simulations of a compound molecule in a simulation box of water or other solvent such as octanol.

It requires as input

- the itp file for the compound
- a coordinate (structure) file (in pdb or gro format)

By default it uses the *OPLS/AA* forcefield and the *TIP4P* water model.

Warning: Other forcefields than OPLS/AA are currently not officially supported; it is not hard to do but requires tedious changes to a few paths in template scripts.

class mdpow.equil.**Simulation**(*molecule=None*, ***kwargs*)
Simple MD simulation of a single compound molecule in water.

Typical use

```
S = Simulation(molecule='DRUG')
S.topology(itp='drug.itp')
S.solvate(struct='DRUG-H.pdb')
S.energy_minimize()
S.MD_relaxed()
S.MD()
```

Note: The OPLS/AA force field and the TIP4P water molecule is the default; changing this is possible but will require provision of customized itp, mdp and template top files at various stages.

Set up Simulation instance.

The *molecule* of the compound molecule should be supplied. Existing files (which have been generated in previous runs) can also be supplied.

Keywords

molecule Identifier for the compound molecule. This is the same as the entry in the `[ molecule ]` section of the itp file. ["DRUG"]

filename If provided and *molecule* is `None` then load the instance from the pickle file *filename*, which was generated with *save()*.

dirname base directory; all other directories are created under it

forcefield 'OPLS-AA' or 'CHARMM' or 'AMBER'

solvent 'water' or 'octanol' or 'cyclohexane' or 'wetoctanol'

solventmodel `None` chooses the default (e.g, *mdpow.forcefields.DEFAULT_WATER_MODEL* for `solvent == "water"`. Other options are the models defined in *mdpow.forcefields.GROMACS_WATER_MODELS*. At the moment, there are no alternative parameterizations included for other solvents.

mdp dict with keys corresponding to the stages `energy_minimize`, `MD_restrained`, `MD_relaxed`, `MD_NPT` and values *mdp* file names (if no entry then the package defaults are used)

distance minimum distance between solute and closest box face

kwargs advanced keywords for short-circuiting; see *mdpow.equil.Simulation.filekeys*.

MD (***kwargs*)

Short NPT MD simulation.

See documentation of `gromacs.setup.MD()` for details such as *runtime* or specific queuing system options. The following keywords can not be changed: *top*, *mdp*, *ndx*, *mainselection*.

Note: If the system crashes (with LINCS errors), try initial equilibration with timestep *dt* = 0.0001 ps (0.1 fs instead of 2 fs) and *runtime* = 5 ps as done in *MD_relaxed()*

Keywords

struct starting conformation; by default, the *struct* is the last frame from the position restraints run, or, if this file cannot be found (e.g. because *Simulation.MD_restrained()* was not run) it falls back to the relaxed and then the solvated system.

mdp MDP run parameter file for Gromacs

runtime total run time in ps

qscript list of queuing system scripts to prepare; available values are in `gromacs.config.templates` or you can provide your own filename(s) in the current directory (see `gromacs.qsub` for the format of the templates)

qname name of the job as shown in the queuing system

startdir **advanced uses**: path of the directory on a remote system, which will be hard-coded into the queuing system script(s); see `gromacs.setup.MD()` and `gromacs.manager.Manager`

MD_NPT (***kwargs*)

Short NPT MD simulation.

See documentation of `gromacs.setup.MD()` for details such as *runtime* or specific queuing system options. The following keywords can not be changed: *top*, *mdp*, *ndx*, *mainselection*.

Note: If the system crashes (with LINCS errors), try initial equilibration with timestep *dt* = 0.0001 ps (0.1 fs instead of 2 fs) and *runtime* = 5 ps as done in *MD_relaxed()*

Keywords

struct starting conformation; by default, the *struct* is the last frame from the position restraints run, or, if this file cannot be found (e.g. because *Simulation.MD_restrained()* was not run) it falls back to the relaxed and then the solvated system.

mdp MDP run parameter file for Gromacs

runtime total run time in ps

qscript list of queuing system scripts to prepare; available values are in `gromacs.config.templates` or you can provide your own filename(s) in the current directory (see `gromacs.qsub` for the format of the templates)

qname name of the job as shown in the queuing system

startdir **advanced uses**: path of the directory on a remote system, which will be hard-coded into the queuing system script(s); see `gromacs.setup.MD()` and `gromacs.manager.Manager`

MD_relaxed(***kwargs*)

Short MD simulation with *timestep* = 0.1 fs to relax strain.

Energy minimization does not always remove all problems and LINCS constraint errors occur. A very short *runtime* = 5 ps MD with very short integration time step *dt* tends to solve these problems.

Keywords

struct starting coordinates (typically guessed)

mdp MDP run parameter file for Gromacs

qscript list of queuing system submission scripts; probably a good idea to always include the default "local.sh" even if you have your own ["local.sh"]

qname name of the job as shown in the queuing system

startdir **advanced uses**: path of the directory on a remote system, which will be hard-coded into the queuing system script(s); see `gromacs.setup.MD()` and `gromacs.manager.Manager`

MD_restrained(***kwargs*)

Short MD simulation with position restraints on compound.

See documentation of `gromacs.setup.MD_restrained()` for details. The following keywords can not be changed: top, mdp, ndx, mainselection

Note: Position restraints are activated with `-DPOSRES` directives for `gromacs.grompp()`. Hence this will only work if the compound itp file does indeed contain a `[ posres ]` section that is protected by a `#ifdef POSRES` clause.

Keywords

struct starting coordinates (leave empty for inspired guess of file name)

mdp MDP run parameter file for Gromacs

qscript list of queuing system submission scripts; probably a good idea to always include the default "local.sh" even if you have your own ["local.sh"]

qname name of the job as shown in the queuing system

startdir **advanced uses**: path of the directory on a remote system, which will be hard-coded into the queuing system script(s); see `gromacs.setup.MD()` and `gromacs.manager.Manager`

coordinate_structures = ('solvated', 'energy_minimized', 'MD_relaxed', 'MD_restrained'
Coordinate files of the full system in increasing order of advancement of the protocol; the later the better. The values are keys into `Simulation.files`.

energy_minimize(***kwargs*)
Energy minimize the solvated structure on the local machine.

kwargs are passed to `gromacs.setup.energ_minimize()` but if *solvate()* step has been carried out previously all the defaults should just work.

filekeys = ('topology', 'processed_topology', 'structure', 'solvated', 'ndx', 'energy_
Keyword arguments to pre-set some file names; they are keys in `Simulation.files`.

get_last_checkpoint()
Returns the checkpoint of the most advanced step in the protocol. Relies on md.gro being present from previous simulation, assumes that checkpoint is then present.

get_last_structure()
Returns the coordinates of the most advanced step in the protocol.

load(*filename=None*)
Re-instantiate class from pickled file.

make_paths_relative(*prefix='.'*)
Hack to be able to copy directories around: prune basedir from paths.

> **Warning:** This is not guaranteed to work for all paths. In particular, check `mdpow.equil.Simulation.dirs.includes` and adjust manually if necessary.

mdp_defaults = {'MD_NPT': 'NPT_opls.mdp', 'MD_relaxed': 'NPT_opls.mdp', 'MD_restraine
Default Gromacs *MDP* run parameter files for the different stages. (All are part of the package and are found with *mdpow.config.get_template()*.)

processed_topology(***kwargs*)
Create a portable topology file from the topology and the solvated system.

protocols = ('MD_NPT', 'MD_NPT_run', 'MD_relaxed', 'MD_relaxed_run', 'MD_restrained',
Check list of all methods that can be run as an independent protocol; see also `Simulation.get_protocol()` and `restart.Journal`

save(*filename=None*)
Save instance to a pickle file.

The default filename is the name of the file that was last loaded from or saved to.

solvate(*struct=None*, ***kwargs*)
Solvate structure *struct* in a box of solvent.

The solvent is determined with the *solvent* keyword to the constructor.

Keywords

struct pdb or gro coordinate file (if not supplied, the value is used that was supplied to the constructor of *Simulation*)

distance minimum distance between solute and the closes box face; the default depends on the solvent but can be set explicitly here, too.

bt any box type understood by `gromacs.editconf()` (`-bt`):

- "triclinic" is a triclinic box,
- "cubic" is a rectangular box with all sides equal;
- "dodecahedron" represents a rhombic dodecahedron;
- "octahedron" is a truncated octahedron.

The default is "dodecahedron".

kwargs All other arguments are passed on to `gromacs.setup.solvate()`, but set to sensible default values. *top* and *water* are always fixed.

topology(*itp='drug.itp'*, *prm=None*, ***kwargs*)
Generate a topology for compound *molecule*.

Keywords

itp Gromacs itp file; will be copied to topology dir and included in topology

prm Gromacs prm file; if given, will be copied to topology dir and included in topology

dirname name of the topology directory ["top"]

kwargs see source for *top_template*, *topol*

class mdpow.equil.**WaterSimulation**(*molecule=None*, ***kwargs*)
Equilibrium MD of a solute in a box of water.

Set up Simulation instance.

The *molecule* of the compound molecule should be supplied. Existing files (which have been generated in previous runs) can also be supplied.

Keywords

molecule Identifier for the compound molecule. This is the same as the entry in the `[ molecule ]` section of the itp file. ["DRUG"]

filename If provided and *molecule* is `None` then load the instance from the pickle file *filename*, which was generated with *save()*.

dirname base directory; all other directories are created under it

forcefield 'OPLS-AA' or 'CHARMM' or 'AMBER'

solvent 'water' or 'octanol' or 'cyclohexane' or 'wetoctanol'

solventmodel `None` chooses the default (e.g, *mdpow.forcefields.DEFAULT_WATER_MODEL* for `solvent == "water"`. Other options are the models defined in *mdpow.forcefields.GROMACS_WATER_MODELS*. At the moment, there are no alternative parameterizations included for other solvents.

mdp dict with keys corresponding to the stages `energy_minimize`, `MD_restrained`, `MD_relaxed`, `MD_NPT` and values *mdp* file names (if no entry then the package defaults are used)

distance minimum distance between solute and closest box face

kwargs advanced keywords for short-circuiting; see *mdpow.equil.Simulation.filekeys*.

class mdpow.equil.**OctanolSimulation**(*molecule=None*, ***kwargs*)

Equilibrium MD of a solute in a box of octanol.

Set up Simulation instance.

The *molecule* of the compound molecule should be supplied. Existing files (which have been generated in previous runs) can also be supplied.

Keywords

molecule Identifier for the compound molecule. This is the same as the entry in the `[ molecule ]` section of the itp file. ["DRUG"]

filename If provided and *molecule* is `None` then load the instance from the pickle file *filename*, which was generated with *save()*.

dirname base directory; all other directories are created under it

forcefield 'OPLS-AA' or 'CHARMM' or 'AMBER'

solvent 'water' or 'octanol' or 'cyclohexane' or 'wetoctanol'

solventmodel `None` chooses the default (e.g, *mdpow.forcefields.DEFAULT_WATER_MODEL* for `solvent == "water"`. Other options are the models defined in *mdpow.forcefields.GROMACS_WATER_MODELS*. At the moment, there are no alternative parameterizations included for other solvents.

mdp dict with keys corresponding to the stages `energy_minimize`, `MD_restrained`, `MD_relaxed`, `MD_NPT` and values *mdp* file names (if no entry then the package defaults are used)

distance minimum distance between solute and closest box face

kwargs advanced keywords for short-circuiting; see *mdpow.equil.Simulation.filekeys*.

mdpow.equil.**DIST = {'cyclohexane': 1.5, 'octanol': 1.5, 'water': 1.0, 'wetoctanol': 1.5**

minimum distance between solute and box surface (in nm)

5.5 Helper modules

The code described here is only relevant for developers.

5.5.1 mdpow.config – Configuration for MDPOW

The config module provides configurable options for the whole package; eventually it might grow into a more sophisticated configuration system but right now it mostly serves to define which gromacs tools and other scripts are offered in the package and where template files are located. If the user wants to change anything they will still have to do it here in source until a better mechanism with a global configuration file has been implemented.

5.5.1.1 Force field

By default, MDPOW uses a collection of OPLS/AA force field files based on the Gromacs 4.5.3 distribution, with the following differences:

- For ions we use the new alkali and halide ion parameters from Table 2 in [Jensen2006] which had shown some small improvements in the paper. They should only be used with the TIP4P water model.
- OPLS/AA parameters for 1-octanol were added. These parameters were validated against experimental data by computing the density (neat), hydration free energy and logP (the latter being a self consistency check).

The force field files are found in the directory pointed to by the environment variable `GMXLIB`. By default, *mdpow.config* sets `GMXLIB` to *includedir* unless `GMXLIB` has already been set. This mechanism allows the user to override the choice of location of force field.

At the moment, only OPLS/AA is tested with MDPOW although in principle it is possible to use other force fields by supplying appropriately customized template files.

References

5.5.1.2 Location of template files

Template variables list files in the package that can be used as templates such as run input files. Because the package can be a zipped egg we actually have to unwrap these files at this stage but this is completely transparent to the user.

mdpow.config.**templates = {'NPT_amber.mdp': '/home/docs/checkouts/readthedocs.org/user_bui**

POW comes with a number of templates for run input files and queuing system scripts. They are provided as a convenience and examples but **WITHOUT ANY GUARANTEE FOR CORRECTNESS OR SUITABILITY FOR ANY PURPOSE**.

All template filenames are stored in `gromacs.config.templates`. Templates have to be extracted from the GromacsWrapper python egg file because they are used by external code: find the actual file locations from this variable.

Gromacs mdp templates

These are supplied as examples and there is *NO GUARANTEE THAT THEY PRODUCE SENSIBLE OUTPUT* — check for yourself! Note that only existing parameter names can be modified with `gromacs.cbook.edit_mdp()` at the moment; if in doubt add the parameter with its gromacs default value (or empty values) and modify later with `edit_mdp()`.

The safest bet is to use one of the `mdout.mdp` files produced by `gromacs.grompp()` as a template as this mdp contains all parameters that are legal in the current version of Gromacs.

mdpow.config.**topfiles = {'1cyclo.gro': '/home/docs/checkouts/readthedocs.org/user_builds/**

List of all topology files that are included in the package. (includes force field files under `top/oplsaa.ff`)

mdpow.config.**includedir = '/home/docs/checkouts/readthedocs.org/user_builds/mdpow/checkout**

The package's include directory for `gromacs.grompp()`; the environment variable `GMXLIB` is set to *includedir* so that the bundled version of the force field is picked up.

mdpow.config.**defaults = {'runinput': '/home/docs/checkouts/readthedocs.org/user_builds/md**

Locations of default run input files and configurations.

5.5.1.3 Functions

The following functions can be used to access configuration data.

mdpow.config.**get_template**(*t*)
Find template file *t* and return its real path.

t can be a single string or a list of strings. A string should be one of

1. a relative or absolute path,
2. a filename in the package template directory (defined in the template dictionary gromacs.config.templates) or
3. a key into templates.

The first match (in this order) is returned. If the argument is a single string then a single string is returned, otherwise a list of strings.

Arguments *t* : template file or key (string or list of strings)

Returns os.path.realpath(*t*) (or a list thereof)

Raises ValueError if no file can be located.

mdpow.config.**get_templates**(*t*)
Find template file(s) *t* and return their real paths.

t can be a single string or a list of strings. A string should be one of

1. a relative or absolute path,
2. a filename in the package template directory (defined in the template dictionary gromacs.config.templates) or
3. a key into templates.

The first match (in this order) is returned for each input argument.

Arguments *t* : template file or key (string or list of strings)

Returns list of os.path.realpath(*t*)

Raises ValueError if no file can be located.

mdpow.config.**get_configuration**(*filename=None*)
Reads and parses a run input config file.

Uses the package-bundled defaults as a basis.

Developer note

Templates have to be extracted from the egg because they are used by external code. All template filenames are stored in config.templates or config.topfiles.

Sub-directories are extracted (see Resource extraction) but the file names themselves will not appear in the template dict. Thus, only store files in subdirectories that don't have to be explicitly found by the package (e.g. the Gromacs force field files are ok).

mdpow.config.**_generate_template_dict**(*dirname*)
Generate a list of included top-level files *and* extract them to a temp space.

Only lists files and directories at the *top level* of the *dirname*; however, all directories are extracted recursively and will be available.

5.5.2 mdpow.log — Configure logging for POW analysis

Import this module if logging is desired in application code and create the logger in `__init__.py`:

```
import log
logger = log.create(logname, logfile)
```

In modules simply use:

```
import logging
logger = logging.getLogger(logname)
```

5.5.3 mdpow.restart — Restarting and checkpointing

The module provides classes and functions to keep track of which stages of a simulation protocol have been completed. It uses a *Journal* class for the book-keeping. Together with saving the current state of a protocol to a checkpoint file (using *Journalled.save()*) it is possible to implement restartable simulation protocols (for example **mdpow-equilibrium**).

exception mdpow.restart.**JournalSequenceError**
Raised when a stage is started without another one having been completed.

class mdpow.restart.**Journal**(*stages*)
Class that keeps track of the stage in a protocol.

Transaction blocks have to be bracketed by calls to *start()* and *completed()*. If a block is started before completion, a *JournalSequenceError* will be raised.

Other methods such as *has_completed()* and *has_not_completed()* can be used to query the status. The attribute *incomplete* flags the state of the current stage (*current*).

All completed stages are recorded in the attribute *history*.

The current (incomplete) stage can be reset to its initial state with *Journal.clear()*.

Example:

```
J = Journal(['pre', 'main', 'post'])
J.start('pre')
...
J.completed('pre')
J.start('main')
...
# main does not finish properly
print(J.incomplete)
# --> 'main'
J.start('post')
# raises JournalSequenceError
```

Initialise the journal that keeps track of stages.

Arguments

stages list of the stage identifiers, in the order that they should per performed. Stage identifiers are checked against this list before they are accepted as arguments to most methods.

clear()
Reset incomplete status and current stage

completed(*stage*)
Record completed stage and reset *Journal.current*

current
Current stage identifier

has_completed(*stage*)
Returns `True` if the *stage* has been started and completed at any time.

has_not_completed(*stage*)
Returns `True` if the *stage* had been started but not completed yet.

history
List of stages completed

incomplete
This last stage was not completed.

start(*stage*)
Record that *stage* is starting.

class mdpow.restart.**Journalled**(**args*, ***kwargs*)
A base class providing methods for journalling and restarts.

It installs an instance of *Journal* in the attribute `Journalled.journal` if it does not exist already.

get_protocol(*protocol*)
Return method for *protocol*.

- If *protocol* is a real method of the class then the method is returned.
- If *protocol* is a registered protocol name but no method of the name exists (i.e. *protocol* is a "dummy protocol") then a wrapper function is returned. The wrapper has the signature

  dummy_protocol(*func*, **args*, ***kwargs*)
  Runs *func* with the arguments and keywords between calls to *Journal.start()* and *Journal.completed()*, with the stage set to *protocol*.

- Raises a `ValueError` if the *protocol* is not registered (i.e. not found in *Journalled.protocols*).

load(*filename=None*)
Re-instantiate class from pickled file.

If no *filename* was supplied then the filename is taken from the attribute `filename`.

protocols = []
Class-attribute that contains the names of computation protocols supported by the class. These are either method names or dummy names, whose logic is provided by an external callback function. The method *get_protocol()* raises a `ValueError` if a protocol is not listed in *protocols*.

save(*filename=None*)
Save instance to a pickle file.

The default filename is the name of the file that was last loaded from or saved to. Also sets the attribute `filename` to the absolute path of the saved file.

mdpow.restart.**checkpoint**(*name*, *sim*, *filename*)
Execute the *Journalled.save()* method and log the event.

5.6 Force field selection

The *mdpow.forcefields* module contains settings for selecting different force fields and the corresponding solvent topologies.

The OPLS-AA, CHARMM/CGENFF and the AMBER/GAFF force field are directly supported. In the principle it is possible to switch to a different force field by supplying alternative template files.

mdpow.forcefields.**DEFAULT_FORCEFIELD = 'OPLS-AA'**
Default force field. At the moment, only OPLS-AA is directly supported.

mdpow.forcefields.**DEFAULT_WATER_MODEL = 'tip4p'**
Use the default water model unless another water model is chosen in the runinput file (`setup.watermodel`).

5.6.1 Solvent models

Different **water models** are already supported

mdpow.forcefields.**GROMACS_WATER_MODELS = {'m24': <M24 water: identifier=m24, ff=OPLS-AA>,**
Dictionary of *GromacsSolventModel* instances, one for each Gromacs water model available under the force field directory. The keys are the water model identifiers. For OPLS-AA the following ones are available.

as well as different general **solvent models**

mdpow.forcefields.**GROMACS_SOLVENT_MODELS = {'AMBER': {'cyclohexane': <CYCLOHEXANE water:**
Solvents available in GROMACS; the keys of the dictionary are the forcefields.

5.6.2 Internal data

mdpow.forcefields.**SPECIAL_WATER_COORDINATE_FILES = {'m24': 'spc216.gro', 'spc': 'spc216.g**
For some water models we cannot derive the filename for the equilibrated box so we supply them explicitly.

mdpow.forcefields.**GROMACS_WATER_MODELS = {'m24': <M24 water: identifier=m24, ff=OPLS-AA>,**
Dictionary of *GromacsSolventModel* instances, one for each Gromacs water model available under the force field directory. The keys are the water model identifiers. For OPLS-AA the following ones are available.

mdpow.forcefields.**GROMACS_SOLVENT_MODELS = {'AMBER': {'cyclohexane': <CYCLOHEXANE water:**
Solvents available in GROMACS; the keys of the dictionary are the forcefields.

5.6.3 Internal classes and functions

class mdpow.forcefields.**GromacsSolventModel**(*identifier*, *name=None*, *itp=None*, *coordinates=None*, *description=None*, *forcefield='OPLS-AA'*)
Data for a solvent model in Gromacs.

guess_filename(*extension*)
Guess the filename for the model and add *extension*

mdpow.forcefields.**get_water_model**(*watermodel='tip4p'*)
Return a *GromacsSolventModel* corresponding to identifier *watermodel*

mdpow.forcefields.**get_solvent_identifier**(*solvent_type*, *model=None*, *forcefield='OPLS-AA'*)
Get the identifier for a solvent model.

The identifier is needed to access a water model (i.e., a *GromacsSolventModel*) through *get_solvent_model()*. Because we have multiple water models but only limited other solvents, the organization of these models is a bit convoluted and it is best to obtain the desired water model in these two steps:

```
identifier = get_solvent_identifier("water", model="tip3p")
model = get_solvent_model(identifier)
```

For `solvent_type` *water*: either provide `None` or "water" for the specific `model` (and the default *DEFAULT_WATER_MODEL* will be selected, or a specific water model such as "tip3p" or "spce" (see *GROMACS_WATER_MODELS*). For other "octanol" or "wetoctanol" of OPLS-AA forcefield, the `model` is used to select a specific model. For other solvents and forcefields, "model" is not required.

Returns Either an identifier or `None`

`mdpow.forcefields.`**`get_solvent_model`**(*identifier*, *forcefield='OPLS-AA'*)

Return a *GromacsSolventModel* corresponding to identifier *identifier*.

If identifier is "water" then the *DEFAULT_WATER_MODEL* is assumed.

Indices and tables

- genindex
- modindex
- search

Bibliography

[Jensen2006] K.P. Jensen and W.L. Jorgensen, *J Comp Theor Comput* **2** (2006), 1499. doi:10.1021/ct600252r

Python Module Index

m

Index

Symbols

C

D

E

F

G

H

I

J

L

M

O

P

R

S

T

W