MDpow Documentation

Release 0.3.1

Oliver Beckstein and Bogdan Iorga

June 13, 2010

CONTENTS

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 *water-octanol partition coefficients* (P_{OW}).

The package is built on top of the GromacsWrapper framework (which can be automatically installed).

Contents:

CHAPTER

ONE

QUICK INSTALLATION INSTRUCTIONS FOR *POW*

Use easy_install (setuptools) because it will fetch additional dependencies.

1.1 Standard installation

Example for installation under `~/opt` with python2.6:

```
export PYTHONPATH=$HOME/opt/lib/python2.6:$PYTHONPATH
easy_install --prefix=$HOME/opt POW
```

Check that you can import the module:

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

In case of problems contact Oliver Beckstein <orbeckst@gmail.com>.

1.2 Developer installation

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

```
cd POW
python setup.py develop
```

Use `--prefix`/`--install-lib` as above or use the defaults in `~/.pydistutils.cfg`; I have:

```
# Mac OS X user installation:
# http://peak.telecommunity.com/DevCenter/EasyInstall#mac-os-x-user-installation
# http://peak.telecommunity.com/DevCenter/EasyInstall#downloading-and-installing-a-package

# note python 2.6 uses ~/.local
# http://docs.python.org/whatsnew/2.6.html

[install]
```

```
install_lib = ~/.local/lib/python$py_version_short/site-packages
install_scripts = ~/bin
```

CHAPTER

TWO

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 are performed with Gromacs 4.x

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 4.0.x.

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*
   (a) set up a short equilibrium simulation of the molecule in a *water* box (and run the MD simulation);
   (b) set up a free energy perturbation calculation of the ligand in water , which will yield the hydration free energy;
2. solvent = *octanol*
   (a) set up a short equilibrium simulation of the molecule in a *octanol* box (and run the MD simulation);
   (b) 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()`.

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
```

2.1.3 Example 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 4.0.x tools can be found, e.g. `which grompp` should show you the path to **grompp**.

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/wa
```

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.)

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)

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.

Analyze output and logPow 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

Helmholtz*, *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;

DeltaA0 = (A_solv - A_vac)

In principle, we calculate the Helmholtz free energy (at constant volume V). In order to obtain the Gibbs free energy (at constant pressure) a small correction Vdp is required. *This correction is currently ignored.*

coulomb contribution of the de-charging process to DeltaA

vdw contribution of the de-coupling process to DeltaA

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`.

Error analysis

The data points are the (time) **average <A>** of A = dV/dl over each window. The **error bars** s_A are the error of the mean <A>. 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_A**2 = 2*tc*acf(0)/T
```

where tc is the decay time of the ACF of <(A-<A>)**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_A 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.

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 `--install-scripts`). See *The mdpow-* scripts* for details.

CHAPTER

THREE

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.

class 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 and mdp 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

solvent water or octanol

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.

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.

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)

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.

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.

coordinate_structures
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
Keyword arguments to pre-set some file names; they are keys in `Simulation.files`.

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.

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

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`)

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', **kwargs*)
Generate a topology for compound *molecule*.

Keywords

itp Gromacs itp file; 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 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

solvent water or octanol

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

class 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

solvent water or octanol

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

ITP
itp files are picked up via include dirs

BOX
solvent boxes

DIST
minimum distance between solute and box surface (in nm)

CHAPTER

FOUR

MDPOW.FEP – CALCULATE FREE ENERGY OF SOLVATION

Set up and run free energy perturbation (FEP) calculations to calculate the free energy of hydration of a solute in a solvent box. The protocol follows the works of D. Mobley (Free Energy Tutorial) and M. Shirts, and uses Gromacs 4.0.x.

Required Input:

- topology
- equilibrated structure of the solvated molecule

See the docs for `Gsolv` for details on what is calculated.

4.1 Differences to published protocols

Some notes on how the approach encoded here differs from what others (notabley Mobley) did:

- We use Gromacs 4.x and use the new decoupling feature

  ```
  couple-intramol = no
  ```

  which indicates that "intra-molecular interactions remain". It should (as I understand it) allow us to only calculate the free energy changes in solution so that we do not have to do an extra calculation in vacuo. http://www.mail-archive.com/gmx-users@gromacs.org/msg18803.html

  Mobley does an extra discharging calculation in vacuo and calculates

  > DeltaA = DeltaA_coul(vac) - (DeltaA_coul(sol) + DeltaA_vdw(sol))

  (but also annihilates the interactions on the solute, corresponding to `couple-intramol = yes`) whereas we do

  > DeltaA = - (DeltaA_coul(sol) + DeltaA_vdw(sol))

- Pressure and Volume: Mobley scales the box size with an affine transformation to the density corresponding to the average pressure of the equilibration run. We simply take the last frame from the trajectory (and also assume that the pressure is exactly what we set, namely 1 bar).

  Arguably we should

  – scale the box to the average volume obtained from the second half of the simulation. This is potentially more important because the density of the water has an effect on the interactions but we would need to run tests to check how big the effect typically is.

- do a Vdp (?) correction (how?)

4.2 Example

see mdpow

4.3 User reference

Simulation setup and analysis of all FEP simulations is encapsulated by a mdpow.fep.Gsolv object. For the hydration free energy there is a special class Ghyd and for the solvation free energy in octanol there is Goct. See the description of Gsolv for methods common to both.

class Gsolv (*molecule=None, top=None, struct=None, **kwargs*)

Simulations to calculate and analyze the solvation free energy.

The simulations are run at constant volume so this is in fact a Helmholtz solvation free energy, DeltaA(V). To compute the Gibbs solvation free energy (which is the experimental quantity) at the standard state (1 M solution and 1 M in the gas phase, also known as the "Ben-Naim standard state"):

DeltaG* = DeltaA + (p*-p)*Vsim

The pressure term is currently NOT IMPLEMENTED. 1 bar = 0.06 kJ mol^-1 nm^-3 typical values of the term are XXX kJ/mol. Vsim is the constant simulations box volume (taken from the last frame of the equilibrium simulation that is the starting point for the FEP simulations.)

(We neglect the negligible correction DeltaA = DeltaAsim -kT ln Vx/Vsim where Vx is the volume of the system without the solute but the same number of water molecules as in the fully interacting case [see Michael Shirts' Thesis, p82].)

DeltaA is computed from the decharging and the decoupling step. With our choice of lambda=0 being the fully interacting and lambda=1 the non-interacting state, it is computed as

Delta A = -(Delta A_coul + Delta A_vdw)

With this protocol, the concentration in the liquid and in the gas phase is the same. (Under the assumption of ideal solution/ideal gas behaviour this seems to directly relate to the Ben-Naim 1M/1M standard state.)

Typical work flow:

```
G = Gsolv(simulation='drug.simulation')           # continue from :mod:`mdpow.equil`
G.setup(qscript=['my_template.sge', 'local.sh'])  # my_template.sge is user supplied
G.qsub()    # run SGE job arrays as generated from my_template.sge
G.analyze()
G.plot()
```

See gromacs.qsub for notes on how to write templates for queuing system scripts (in particular queuing system templates).

Set up Gsolv from input files or a equilibrium simulation.

Arguments

molecule name of the molecule for which the hydration free energy is to be computed (as in the gromacs topology) [REQUIRED]

top topology [REQUIRED]

struct solvated and equilibrated input structure [REQUIRED]

ndx index file

dirname directory to work under ['FEP/*solvent*']

solvent name of the solvent (only used for path names); will not affect the simulations

lambda_coulomb list of lambdas for discharging: q+vdw –> vdw

lambda_vdw list of lambdas for decoupling: vdw –> none

runtime simulation time per window in ps [5000]

temperature temperature in Kelvin of the simulation [300.0]

qscript template or list of templates for queuing system scripts (see `gromacs.config.templates` for details) [local.sh]

includes include directories

simulation Instead of providing the required arguments, obtain the input files from a `mdpow.equil.Simulation` instance.

filename Instead of providing the required arguments, load from pickle file

basedir Prepend *basedir* to all filenames; `None` disables [.]

permissive Set to `True` if you want to read past corrupt data in output xvg files (see `gromacs.formats.XVG` for details); not that *permissive*=''True'' can lead to **wrong results*. Overrides the value set in a loaded pickle file. [`False`]

kwargs other undocumented arguments (see source for the moment)

`Vdp` (*p=1.0*)

Vdp contribution to Gibbs free energy

DeltaG = DeltaA + V*DeltaP (for V=const)

Delta refers to

aq (fully coupled) –> gaseous (decoupled)

> **Warning:** Not implemented at the moment and simply set to 0 because we cannot use the pressures directly from the simulation. In such a small system they fluctuate too much and completely dominate all thermodynamic calculations.

Returns 0 (although it should be something like Vsim*(p*-p)) [in kJ/mol]

Arguments

p constant pressure in bar [1.0]

`analyze` (*p=1.0, force=False, autosave=True*)

Extract dV/dl from output and calculate dG by TI.

Thermodynamic integration (TI) is performed on the individual component window calculation (typically the Coulomb and the VDW part, DeltaA_coul and DeltaA_vdw). DeltaA_coul is the free energy component of discharging the molecule and DeltaA_vdw of decoupling (switching off LJ interactions with the environment). The free energy components must be interpreted in this way because we defined lambda=0 as interaction switched on and lambda=1 as switched off.

Delta A* = -(Delta A_coul + Delta A_vdw)

Delta_G* = Delta_A* + (p*-p)*V [not implemented]

Data are stored in `Gsolv.results`.

The dV/dlambda graphs are integrated with the composite Simpson's rule (and if the number of datapoints are even, the first interval is evaluated with the trapezoidal rule); see `scipy.integrate.simps()` for details). Note that this implementation of Simpson's rule does not require equidistant spacing on the lambda axis.

For the Coulomb part using Simpson's rule has been shown to produce more accurate results than the trapezoidal rule [Jorge2010].

Errors are estimated from the errors of the individual <dV/dlambda>:

1. The error of the mean <dV/dlambda> is calculated via the decay time of the fluctuation around the mean (see `gromacs.formats.XVG.error`).
2. The error on the integral is calculated analytically via propagation of errors through Simpson's rule (with the approximation that all spacings are assumed to be equal; taken as the average over all spacings as implemented in `numkit.integration.simps_error()`).

Note: For the Coulomb part, which typically only contains about 5 lambdas, it is recommended to have a odd number of lambda values to fully benefit from the higher accuracy of the integration scheme.

Keywords

p pressure in bar [1.0] TODO: should be obtained from equilib sim

force reload raw data even though it is already loaded

autosave save to the pickle file when results have been computed

arraylabel(*component*)
Batch submission script name for a job array.

collect(*autosave=True*)
Collect dV/dl from output

contains_corrupted_xvgs()
Check if any of the source datafiles had reported corrupted lines.

Returns `True` if any of the xvg dgdl files were produced with the permissive=True flag and had skipped lines.

For debugging purposes, the number of corrupted lines are stored in `Gsolv._corrupted` as dicts of dicts with the component as primary and the lambda as secondary key.

correlationtime(*nstep=100*)
Calculate the correlation time from the ACF.

The autocorrelation function is calculated via FFT on every *nstep* of the data. It is assumed to decay exponentially, f(t) = exp(-t/tau) and the decay constant is estimated as the integral of the ACF from the start up to its first root.

frombase(**args*)
Return path with `Gsolv.basedir` prefixed.

has_dVdl()
Check if dV/dl data have already been collected.

Returns `True` if the dV/dl data have bee aquired (`Gsolv.collect()`) for all FEP simulations.

label(*component*)
Simple label for component, e.g. for use in filenames.

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

logger_DeltaA0()
Print the free energy contributions (errors in parentheses).

plot(***kwargs*)
Plot the TI data with error bars.

Run `mdpow.fep.Gsolv.analyze()` first.

All *kwargs* are passed on to `pylab.errorbar()`.

qsub(*script=None*)
Submit a batch script locally.

If *script* == `None` then take the first script (works well if only one template was provided).

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.

setup(***kwargs*)
Prepare the input files for all Gromacs runs.

Keywords

qscript (List of) template(s) for batch submission scripts; if not set then the templates are used that were supplied to the constructor.

kwargs Most kwargs are passed on to `gromacs.setup.MD()` although some are set to values that are required for the FEP functionality. Note: *runtime* is set from the constructor.

summary()
Return a string that summarizes the energetics.

Each energy component is followed by its error.

Format:: . ———- kJ/mol —— molecule solvent total coulomb vdw Vdp

tasklabel(*component, lmbda*)
Batch submission script name for a single task job.

wdir(*component, lmbda*)
Return path to the work directory for *component* and *lmbda*.

write_DeltaA0(*filename, mode='w'*)
Write free energy components to a file.

Arguments

filename name of the text file

mode 'w' for overwrite or 'a' for append ['w']

Format:: . ———- kJ/mol —— molecule solvent total coulomb vdw Vdp

class Ghyd(*molecule=None, top=None, struct=None, **kwargs*)
Sets up and analyses MD to obtain the hydration free energy of a solute.

Set up Gsolv from input files or a equilibrium simulation.

Arguments

molecule name of the molecule for which the hydration free energy is to be computed (as in the gromacs topology) [REQUIRED]

top topology [REQUIRED]

struct solvated and equilibrated input structure [REQUIRED]

ndx index file

dirname directory to work under ['FEP/*solvent*']

solvent name of the solvent (only used for path names); will not affect the simulations

lambda_coulomb list of lambdas for discharging: q+vdw –> vdw

lambda_vdw list of lambdas for decoupling: vdw –> none

runtime simulation time per window in ps [5000]

temperature temperature in Kelvin of the simulation [300.0]

qscript template or list of templates for queuing system scripts (see `gromacs.config.templates` for details) [local.sh]

includes include directories

simulation Instead of providing the required arguments, obtain the input files from a `mdpow.equil.Simulation` instance.

filename Instead of providing the required arguments, load from pickle file

basedir Prepend *basedir* to all filenames; `None` disables [.]

permissive Set to `True` if you want to read past corrupt data in output xvg files (see `gromacs.formats.XVG` for details); not that *permissive*=''True'' can lead to **wrong results**. Overrides the value set in a loaded pickle file. [`False`]

kwargs other undocumented arguments (see source for the moment)

class `Goct` (*molecule=None, top=None, struct=None, **kwargs*)

Sets up and analyses MD to obtain the solvation free energy of a solute in octanol.

NOT ENABLED YET:

The *coulomb* lambda schedule is enhanced compared to water as the initial part of the dV/dl curve is quite sensitive. By adding two additional points we hope to reduce the overall error on the dis-charging free energy.

Set up Gsolv from input files or a equilibrium simulation.

Arguments

molecule name of the molecule for which the hydration free energy is to be computed (as in the gromacs topology) [REQUIRED]

top topology [REQUIRED]

struct solvated and equilibrated input structure [REQUIRED]

ndx index file

dirname directory to work under ['FEP/*solvent*']

solvent name of the solvent (only used for path names); will not affect the simulations

lambda_coulomb list of lambdas for discharging: q+vdw –> vdw

lambda_vdw list of lambdas for decoupling: vdw –> none

runtime simulation time per window in ps [5000]

temperature temperature in Kelvin of the simulation [300.0]

qscript template or list of templates for queuing system scripts (see `gromacs.config.templates` for details) [local.sh]

includes include directories

simulation Instead of providing the required arguments, obtain the input files from a `mdpow.equil.Simulation` instance.

filename Instead of providing the required arguments, load from pickle file

basedir Prepend *basedir* to all filenames; `None` disables [.]

permissive Set to `True` if you want to read past corrupt data in output xvg files (see `gromacs.formats.XVG` for details); not that *permissive*=‘‘True‘‘ can lead to **wrong results*. Overrides the value set in a loaded pickle file. [`False`]

kwargs other undocumented arguments (see source for the moment)

pOW(*G1, G2, force=False*)

Compute water-octanol partition coefficient from two `Gsolv` objects.

transfer free energy from water into octanol DeltaDeltaA0 = DeltaA0_oct - DeltaA0_water

water-octanol partition coefficient log P_oct/wat = log [X]_oct/[X]_wat

Arguments

G1, G2 *G1* and *G2* should be a `Ghyd` and a `Goct` instance, but order does not matter

force force rereading of data files even if some data were already stored [False]

Returns (transfer free energy, log10 of the water-octanol partition coefficient = log Pow)

4.4 Developer notes

A user really only needs to access classes derived from `mdpow.fep.Gsolv`; all other classes and functions are auxiliary and only of interest to developers.

Additional objects that support `mdpow.fep.Gsolv`.

class FEPschedule()

Describe mdp parameter choices as key - value pairs.

mdp_dict

Dict of key-values that can be set in a mdp file.

molar_to_nm3(*c*)

Convert a concentration in Molar to nm^-3.

kcal_to_kJ(*x*)

Convert a energy in kcal to kJ.

kJ_to_kcal(*x*)

Convert a energy in kJ to kcal.

N_AVOGADRO

Avogadro’s constant N_A in mol^-1 (NA NIST value).

kBOLTZ

Boltzmann’s constant k_B in kJ mol^-1 (kB NIST value).

fep_templates
Template mdp files for different stages of the FEP protocol. (add equilibration, too?)

4.4.1 TODO

- run minimization, NVT-equil, NPT-equil prior to production (probably use preprocessed topology from `grompp -pp` for portability)

  See Free Energy Tutorial.

CHAPTER

FIVE

MDPOW.ANALYSIS — COLLECTION OF ANALYSIS AND PLOTTING FUNCTIONS

Simple functions to quickly plot data. Typically, it works best if ran interactively from the top level of the POW directory!

Experimental values are loaded from the targets list (`targets.csv`) and computed values from the table in `pow.txt`. See `plot_exp_vs_comp()` for details.

5.1 Prepare data

First copy the **computed results** , the `pow.txt` and `energies.txt` files that are produced by **mdpow-pow**, into the `data` directories.

Then format them:

```
./lib/scripts/make_tables.sh data/*
```

The **experimental data** are taken from *targets.numbers*. In **numbers** export the table as *UTF-8* in *CSV* format to `experimental/targets.csv`. This is only necessary if the experimental data were changed. We only plot entries for which

- a id number (first column *no*) is defined (should be unique)
- a *logPow* value exists
- a *itp_name* exists, which must correspond to the *molecule* name used in `mdpow.fep.Gsolv`

5.2 Making graphs

Plot results and save to a pdf file with `plot_exp_vs_comp()`:

```
mdpow.analysis.plot_exp_vs_comp(figname="figs/exp_vs_comp.pdf")
```

By default we also include the SAMPL2 results.

Remove the bad runs from `pow.txt` and save as `pow_best.txt`. Then plot again (this time excluding the SAMPL2 results):

```
pylab.clf()
mdpow.analysis.plot_exp_vs_comp(data="data/run01/pow_best.txt", data2=None, figname='figs/run01/exp_v
```

Note that the SAMPL2 results are excluded by setting `data2=None`.

5.3 Functions

plot_exp_vs_comp(***kwargs*)

class ExpComp(***kwargs*)

Keywords

experiments path to `targets.csv`

data path to `pow.txt` of the test set ("run01")

data2 SAMPL2 data `pow.txt`; set *data2* = `False` or `None` to exclude set; unset chooses the default

class ExpData(*filename='experimental/targets.csv', **kwargs*)

Object that represents our experimental data.

Access the raw data via `ExpData.rawdata` and a table enriched with statistics as `ExpData.data` (which is a `recsql.SQLarray`).

Load experimental values table and return `recsql.SQLarray`.

To obtain `targets.csv` export `targets.numbers` in **Numbers** as **UTF-8** (important!) in the **CSV* format (File->Export)

class ComputedData(*filename='data/run01/pow.txt', **kwargs*)

Object that represents computed data.

Access via `ComputedData.data`.

Load computed POW table and return `recsql.SQLarray`.

The data file is typically `pow.txt`. It *must* contain a proper reST table. Use the `_header` and `_footer` files if you only have the raw output from **mdpow-pow**.

Furthermore, the column names are important because we use them here.

load_data(*filename='data/run01/pow.txt', **kwargs*)

Load computed POW table and return `recsql.SQLarray`.

The data file is typically `pow.txt`. It *must* contain a proper reST table. Use the `_header` and `_footer` files if you only have the raw output from **mdpow-pow**.

Furthermore, the column names are important because we use them here.

load_exp(*filename='experimental/targets.csv', **kwargs*)

Load experimental values table and return `recsql.SQLarray`.

To obtain `targets.csv` export `targets.numbers` in **Numbers** as **UTF-8** (important!) in the **CSV* format (File->Export)

CHAPTER

SIX

HELPER MODULES

The code described here is only relevant for developers.

6.1 `mdpow.config` – Configuration for POW

The config module provides configurable options for the whole package; eventually it might grow into a sophisticated configuration system such as matplotlib's rc 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 rc files has been implemented.

6.1.1 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.

templates
: *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.

topfiles
: List of all topology files that are included in the package.

includedir
: The package's include directory for `gromacs.grompp()`.

6.1.2 Functions

The following functions can be used to access configuration data.

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.

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.

6.2 `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)
```

CHAPTER

SEVEN

THE MDPOW-* SCRIPTS

A number of python scripts are installed together with the mdpow package. They simplufy 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".

7.1 Running analysis

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)

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 (–plot=auto). If multiple DIRECTORY arguments are provided then one has to choose the auto option (or None).

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

```
molecule   DeltaA0(kJ/mol)    log P_OW  wat_ok octa_ok  directory
```

molecule molecule name as used in the itp

DeltaA0 transfer free energy water –> octanol, in kJ/mol

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

wat_ok, octa_ok "OK": input data was clean; "BAD": some input data xvg files contained unparseable lines that were skipped; this can mean that the data became corrupted and caution should be used.

directory folder in which the simulations were stored

Options:

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

-p FILE, --plotfile=FILE plot dV/dlambda to FILE; use png or pdf suffix to determine the file type. 'auto' generates a pdf file DIRECTORY/dVdl.pdf and 'None' disables it [auto]

-o FILE, --outfile=FILE append one-line results summary to FILE [pow.txt]

-e FILE, --energies=FILE append solvation free energies to FILE [energies.txt]

--force force rereading all data [False]

--ignore-corrupted skip lines in the md.xvg files that cannot be parsed. 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 [False]

7.2 Checking if the simulation is complete

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

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]

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

It can become necessary to recreate the fep pickle 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: 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.)

Options:

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

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

CHAPTER

EIGHT

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

BIBLIOGRAPHY

[Jorge2010] M. Jorge, N.M. Garrido, A.J. Queimada, I.G. Economou, and E.A. Macedo. Effect of the integration method on the accuracy and computational efficiency of free energy calculations using thermodynamic integration. Journal of Chemical Theory and Computation, 6 (4):1018–1027, 2010. 10.1021/ct900661c.

MODULE INDEX

M

INDEX

Q

S

T

V

W