# MDpow Documentation

***Release 0.0.4***

**Oliver Beckstein and Bogdan Iorga**

March 25, 2010

# CONTENTS


**1 Quick installation instructions for *POW*** **3**
- 1.1 Standard installation . . . 3
- 1.2 Developer installation . . . 3

**2 `mdpow` — Computing the octanol/water partitioning coefficient** **5**
- 2.1 How to use the module . . . 5

**3 `mdpow.equil` — Setting up and running equilibrium MD** **9**

**4 `mdpow.fep` – calculate free energy of solvation** **13**
- 4.1 Example . . . 13
- 4.2 User reference . . . 13
- 4.3 Developer notes . . . 17

**5 Helper modules** **19**
- 5.1 `mdpow.config` – Configuration for POW . . . 19
- 5.2 `mdpow.log` — Configure logging for POW analysis . . . 20

**6 Indices and tables** **21**

**Module Index** **23**

**Index** **25**

*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* ($P_{\mathrm{OW}}$).

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.

### 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
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 (if you quit it you will have to `import mdpow` again) and set up a `Ghyd` object:

```
G = mdpow.fep.Ghyd(molecule="OcOH", top="Equilibrium/water/top/system.top", struct="Equilibrium/wate
```

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):

```
S = mdpow.equil.WaterSimulation(filename="water.simulation")  # only needed when quit
G = 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:

```
G.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:

```
H = mdpow.fep.Goct(simulation=O, runtime=100)
H.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

For the water and octanol FEPs do

```
G.collect()
G.analyze()

H.collect()
H.analyze()
```

The analyze step reports the estimate for the free energy difference. All results can also accessed as a dictionary

```
G.results.DeltaA
```

Free energy of transfer from water to octanol:

```
H.results.DeltaA.total - G.results.DeltaA.total
```

The individual components are

**total** total standard free energy difference in kJ/mol; DeltaA0 = (A_solv - A_vac) + DA_v

**standardstate** correction DA_v for the standard state concentration $c$ (depends on the volume of the simulation cell); in kJ/mol. Different $c$ can be supplied as an argument to analyze().

**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
G.plot()
pylab.figure()
H.plot()
```

The error data points are the average of dV/dl over each windows. The error bars are the standard deviations of the data points from the average.

If the graphs do not look smooth then a longer *runtime* is definitely required. It might also be necessary to add additional lambda values in regions where the function changes rapidly.

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

## 4.1 Example

see mdpow

## 4.2 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 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

***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.

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

***kwargs*** other undocumented arguments (see source for the moment)

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

Simulations to calculate and analyze the solvation free energy.

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

***kwargs*** other undocumented arguments (see source for the moment)

**`DeltaA0`**(*c0=1.0, N=1*)

Standard state correction to the free energy.

The total standard hydration free energy is calculated by taking the change to standard concentration *c0* from the simulation box volume into account:

DeltaA_std = Delta A_0 - Delta A = -kT ln V0/V = -kT ln 1/V*c0*N_A

where V is the volume of the simulation cell. (This assumes that there is only a single molecule $N = 1$ for which the free energy change was computed.)

**Note:** This correction is not exact; see Michael Shirts' thesis for the correct one (but the difference should be small).

**`analyze`**(*c0=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). The dV/dlambda graphs are integrated with Simpson's rule (and averaging of results if the number of datapoints is odd; see `scipy.integrate.simps()` for details).

The total standard hydration free energy is calculated by taking the change to standard concentration from the simulation box volume into account:

DeltaA_std = A_0 - A_sim = -kT ln V0/Vsim = -kT ln 1/Vsim*c0*N_A

(This assumes that there is only a single molecule for which the free energy change was computed.)

Data are stored in `Gsolv.results`.

**Keywords**

***c0*** standard state concentration in mol L^-1 (i.e. M) [1.0]

***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

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

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

**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.

**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*.

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

## 4.3 Developer notes

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.

**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?)

### 4.3.1 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

# HELPER MODULES

The code described here is only relevant for developers.

## 5.1 `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.

### 5.1.1 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()`.

### 5.1.2 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.

## 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)
```

CHAPTER

SIX

# INDICES AND TABLES

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

# MODULE INDEX

## M

mdpow, 4
mdpow.config, 19
mdpow.equil, 8
mdpow.fep, 12
mdpow.log, 20

# INDEX


## A

analyze() (mdpow.fep.Gsolv method), 15
arraylabel() (mdpow.fep.Gsolv method), 16

## B

BOX (in module mdpow.equil), 12

## C

collect() (mdpow.fep.Gsolv method), 16
coordinate_structures (mdpow.equil.Simulation attribute), 10

## D

DeltaA0() (mdpow.fep.Gsolv method), 15
DIST (in module mdpow.equil), 12

## E

energy_minimize() (mdpow.equil.Simulation method), 10

## F

fep_templates (in module mdpow.fep), 17
FEPschedule (class in mdpow.fep), 17
filekeys (mdpow.equil.Simulation attribute), 10

## G

get_last_structure() (mdpow.equil.Simulation method), 10
get_template() (in module mdpow.config), 20
get_templates() (in module mdpow.config), 20
Ghyd (class in mdpow.fep), 13
Goct (class in mdpow.fep), 14
Gsolv (class in mdpow.fep), 14

## I

includedir (in module mdpow.config), 19
ITP (in module mdpow.equil), 12

## K

kBOLTZ (in module mdpow.fep), 17

## L

label() (mdpow.fep.Gsolv method), 16
load() (mdpow.equil.Simulation method), 10
load() (mdpow.fep.Gsolv method), 16

## M

make_paths_relative() (mdpow.equil.Simulation method), 10
MD() (mdpow.equil.Simulation method), 9
MD_relaxed() (mdpow.equil.Simulation method), 10
MD_restrained() (mdpow.equil.Simulation method), 10
mdp_dict (mdpow.fep.FEPschedule attribute), 17
mdpow (module), 4
mdpow.config (module), 19
mdpow.equil (module), 8
mdpow.fep (module), 12
mdpow.log (module), 20
molar_to_nm3() (in module mdpow.fep), 17

## N

N_AVOGADRO (in module mdpow.fep), 17

## O

OctanolSimulation (class in mdpow.equil), 11

## P

plot() (mdpow.fep.Gsolv method), 16
processed_topology() (mdpow.equil.Simulation method), 10

## Q

qsub() (mdpow.fep.Gsolv method), 16

## S

save() (mdpow.equil.Simulation method), 11
save() (mdpow.fep.Gsolv method), 16
setup() (mdpow.fep.Gsolv method), 16
Simulation (class in mdpow.equil), 9
solvate() (mdpow.equil.Simulation method), 11

## T

tasklabel() (mdpow.fep.Gsolv method), 16
templates (in module mdpow.config), 19
topfiles (in module mdpow.config), 19
topology() (mdpow.equil.Simulation method), 11

## W

WaterSimulation (class in mdpow.equil), 11
wdir() (mdpow.fep.Gsolv method), 16