Protons for OpenMM Documentation

Release 0.0.1

Bas Rustenburg, Greg Ross, John Chodera et al

Mar 14, 2019

Contents

Caution: This is module is undergoing heavy development. None of the API calls are final. This software is provided without any guarantees of correctness, you will likely encounter bugs.

If you are interested in this code, please wait for the official release to use it. In the mean time, to stay informed of development progress you are encouraged to:

- Follow this feed for updates on releases.
- Check out the github repository .

CHAPTER 1

Table of contents

1.1 Introduction

Protons is a python module that implements a constant-pH molecular dynamics scheme for sampling protonation states and tautomers of amino acids and small molecules in OpenMM. The codebase is currently under development by the Chodera lab at Memorial Sloan Kettering Cancer center. There is no official release of the code yet.

If you are interested in the development of this code, you can watch the repository on Github. For the planned features of the code, see *Feature Roadmap*.

1.2 Feature Roadmap

The completed code will contain the following features:

- Simulation of proteins protonation states
- Support for small molecule protonation states, and tautomers through Epik calculations.
- Support for implicit and explicit solvent models.
- Support for instantaneous Monte Carlo, and Non-equilibrium (NCMC) state switches.
- Compatibility with the Yank free energy calculation framework.

1.2.1 Development notes

This code is currently under heavy development. At the current time, we are ironing out potential bugs, and extending the test suite. Here is a status update on some of the features we're working on.

Protein simulation

Our code is capable of performing instantaneous state switching for implicit solvent simulations. At the moment, we only support updating the protonation states of the sidechains if the following amino acids:

- Glutamic acid, (pKa=4.4)
- Aspartic acid, (pKa=4.0)
- Histidine, (pKa delta=6.5, pKa epsilon = 7.1)
- Tyrosine, (pKa=9.6)
- Cysteine, (pKa=8.5)
- Lysine, (pKa=10.4)

The pKa values used originate from [Mongan2004].

Small molecule support

Small molecule support is is a much anticipated feature that we plan to implement soon. The code will be extended to use output from Epik calculations ([Shelley2007], [Greenwood2010], [Epik2016]) to provide us with the populations of different protonation states and tautomers in aqueous solvent conditions. For parameter generation, we will rely on the GAFF forcefield [Wang2004], as provided with the antechamber program [Amber2016].

Explicit solvent

Explicit solvent is supported by the code, but it is not feature complete. The current version of the software only features a working implementation of instantaneous Monte Carlo. This will likely lead to low acceptance rates for protonation state switching. We are in the process of developing an approach that uses NCMC [Nilmeier2011], [Chen2015] but the implementation is not finished.

1.3 Installing Protons

Follow these instructions to install `protons`, and its dependencies. If you run into any issues, please raise an issue on our Github page.

1.3.1 Install using conda

The recommended way to install `protons` will be through conda. You can obtain conda by installing the Anaconda python distribution. It will provide a wide suite of scientific python packages, including some of the dependencies in `protons`. Additionally it provides the very powerful `conda` python environment and package manager. For instructions on how to use conda, please see this page to get you started.

As part of the first release, we will start building conda packages for `protons`. You will be able to install `protons` using the following conda command.

```
conda install -c omnia protons
```

This will install the latest version of `protons` from the Omnia channel.

Note: Currently, no conda packages are being built. You can install dependencies through conda, but the package itself needs to be installed using `setup.py` . See below for instructions.

1.3.2 Install using setup.py

Currently, the only way to install protons is by using the `setup.py` script that is provided with the package.

Download a copy of the repository, and cd into it.

```
git clone https://github.com/choderalab/constph-openmm.git
cd constph-openmm
```

Them, use the command

```
python setup.py install
```

to install the package. The `setup.py` installation does not automatically check for requirements. Make sure to install the dependencies separately using `pip`, or `conda`. See below for instructions.

1.3.3 Requirements

The `protons` package has the following requirements:

- python 2.7, 3.4 or 3.5
- openmm >=7.0.1
- numpy >=1.10
- scipy >=0.17.0
- openmmtools >=0.7.5
- pymbar
- openmoltools >=0.7.0
- ambermini
- joblib
- lxml

These should be available from the *omnia* conda channel. The following example commands create a new conda environment called "protons", with all dependencies installed.

```
# Point conda to the omnia channel as package source
conda config --add channels omnia
# Create a new environment called "protons", with all dependencies included
conda create --name protons openmm=7.0.1 numpy=1.10 scipy=0.17.0 openmmtools=0.7.5␣
↪pymbar openmoltools ambermini joblib lxml
# Switch to the new environment
source activate protons
```

1.3.4 Testing your installation

Our library comes with an extensive test suite, designed to detect potential problems with your installation. After you've installed `protons`, it is recommended to verify that the code is working as expected.

To test the installation, run

```
py.test --pyargs protons
```

This requires that you have installed py.test in your python environment. The output of this command will tell you if any parts of the library are not working correctly. Note that it may take some time to complete the tests.

If a test fails, please try and verify whether your installation was successful. You may want to try reinstalling the library in a clean python environment and then testing it. If your tests still fail, please raise an issue on our Github page.

1.4 Preparing systems for constant-pH simulation

Several steps are necessary to prepare a simulation system for simulation using `protons`. Most of `protons` relies on the OpenMM API for Amber files. In order to run constant-pH simulations, the `protons` package requires input files generated by several programs from the Ambertools suite.

These instructions assume that you have a `.pdb` file of your system that has all necessary components to start an MD simulation. If you need help adding missing residues, atoms, or other features to your system, consider reading the OpenMM Modeller documentation, or the Amber manual.

Using the examples shown below, we will demonstrate how to generate input for an implicit solvent constant-pH simulation, including

- An amber topology file (`.prmtop`), which contains the parameters and topology of the system.
- A coordinate file (`.inpcrd`), which contains the 3-dimensional coordinates of your system.
- A protonation state file (`.cpin`), which enumerate protonation states of amino acids.

1.4.1 Obtaining and using Ambertools

Ambertools provides a series of command-line utilities that aid in setting up simulation systems. We will be using these for the manipulation of `.pdb` and `.mol2` files, and for generating parameters for small molecules using the general Amber force field (GAFF). The instructions below have been tested with Ambertools16.

Ambertools16 can be obtained from the Amber website.

For any instructions relating to Ambertools programs, we refer to the Amber manual. This includes instructions on how to install Ambertools.

After following the installation procedure, please make sure you can access the following programs on your command-line:

- `tleap`,
- `cpinutil.py`,

and if you're planning to use small molecules:

- `antechamber`,
- `parmchk`.

We will be using these to prepare systems for constant-pH simulation in OpenMM.

1.4.2 Fixing residue names and atoms

Several modifications will need to be made to your input file, in order to rename residues, and prevent issues with preexisting hydrogen atoms.

Renaming residues

One of the first steps of preparing an input file for constant-pH simulation is to rename all residues in the pdb file to their most protonated form.

The shortlist for PDB residues that our code supports is as follows:

- Histidine (HIP)
- Aspartic acid (AS4)
- Glutamic acid (GL4)
- Cysteine (CYS)
- Lysine (LYS)
- Arginine (ARG)

Note: You may also need to rename any cysteine residues involved in disulfide bonds from `CYS` → `CYX`, for further processing steps.

Treating hydrogens

To properly add all necessary hydrogens to the system, it is easiest to first delete all current hydrogens in the system. Otherwise, improperly named hydrogens may cause issues when defining the topology of the system based on the `.pdb` file. Missing hydrogens will be added back in the next preparation step, so it is safe to delete them at this stage.

An example bash script for processing `.pdb` files

Calling the following shell script on your pdb file should help you prepare your `.pdb` file for simulation. It replaces the names of residues in-place in the file, and it removes hydrogens from your file.

```
pdbfile="input.pdb"

# Rename to constph residues
sed -i 's/HIE/HIP/g' ${pdbfile}
sed -i 's/HID/HIP/g' ${pdbfile}
sed -i 's/HIS/HIP/g' ${pdbfile}
sed -i 's/ASP/AS4/g' ${pdbfile}
sed -i 's/GLU/GL4/g' ${pdbfile}

# Remove hydrogens (print lines with atom names not starting with H)
awk '$3 !~ /^ *H/' ${pdbfile} > tmp && mv tmp ${pdbfile}
```

Tools like `sed` and `awk` can be very useful for manipulating text files quickly, though I recommend that you verify the output manually afterwards.

1.4.3 Writing out coordinate and parameter files using Leap

The next step in system preparation is generating the Amber coordinate (`.inpcrd`), and the topology and parameters files (`.prmtop`). This can be done using the command-line utility `tleap`, which is available as part of ambertools.

These instructions will suffice if your system does not include any small molecules. If your system contains a ligand, please see the *next section*.

You can use `tleap` interactively on the command line. Just type

```
tleap
```

And you will see output similar to this

```
-bash-4.1$ tleap
-I: Adding /home/user/bin/../dat/leap/prep to search path.
-I: Adding /home/user/bin/../dat/leap/lib to search path.
-I: Adding /home/user/bin/../dat/leap/parm to search path.
-I: Adding/home/user/bin/../dat/leap/cmd to search path.

Welcome to LEaP!
(no leaprc in search path)
>
```

You can start typing your commands line by line. Alternatively, you can store commands in a text file, and then use

```
tleap -f tleap.txt
```

and tleap will run the specified commands automatically. Tleap output can be rather verbose. It is recommended to write the output to file, so you can verify that all steps were executed correctly.

Here is a bash example:

```
tleap -f tleap.in >> tleap.out 2>&1
```

You can rename the `.out` file to anything of your choosing.

Tleap commands

The following sequence of commands should do for a simple pdb file containing one protein structure.

```
# Load constant ph parameters
source leaprc.constph

# Load the PDB file, rename it to your input file
protein = loadPDB input.pdb

# Validate the input
check protein

# Calculate the total charge, for logging purposes
charge protein

# Write parameters.
saveAmberParm protein complex.prmtop complex.inpcrd

# Write PDB files, optional
```

(continues on next page)

(continued from previous page)

```
savepdb protein complex.pdb

# Exit, make sure not to forget this part
quit
```

If you need to perform other steps to prepare your system for simulation, please read the Amber manual.

Validating tleap results

If you run interactively, tleap should provide error messages on screen. The output can be rather verbose, so make sure that your terminal is configured to scroll back far.

Alternatively, if you run using an input file, make sure that `tleap` ran successfully.

I often write output to a log file, and check the log file for errors. Here is a short bash snippet that does the trick.

```
tleap -f tleap.in >> tleap.out 2>&1

# There might be other error clues. This method isn't fail safe.
tleap_result=$(grep "usage" tleap.out || grep -i "error" tleap.out)

# As long as the grep results are empty
 if [ -z "$tleap_result" ]
 then
   echo -e "\e[32mTleap looks successful. Still, act cautious. She's a slippery one.
↪\e[39m"
 else
   echo -e "\e[31mCaught an error in Tleap. Tough luck, buddy.\e[39m"
   echo $tleap_result
 fi
```

This procedure generates three different files:

- `complex.prmtop`, an Amber topology file which contains the topology and parameters of the protein system.
- `complex.inpcrd`, a file containing the coordinates of all atoms in the system
- `complex.pdb`, this file is optional. You can use a `pdb` file in software such as PyMOL, to verify that the prepared structure doesn't contain mistakes.

You will be needing these to run your OpenMM script.

1.4.4 Including ligands in your system

Warning:

- Ligand support is a work in progress. We've experienced system instability with small molecules in implicit solvent simulations.

If you have a ligand, you will have to prepare your ligand using `antechamber`, and `parmchk`. This is used to generate two files

- `ligand.gaff.mol2`, a mol2 file with GAFF atom types.
- `ligand.gaff.frcmod`, an frcmod file with GAFF parameters for the ligand.

Here is an example of how to run `antechamber` and `parmchk`.

```
antechamber -i ligand.mol2 -fi mol2 -o ligand.gaff.mol2 -fo mol2
parmchk -i ligand.gaff.mol2 -o ligand.gaff.frcmod  -f mol2
```

You may wish to explore the advanced options of `antechamber` if you need to generate charges for your ligands. If you want to generate charges in another program, using a `.mol2` file should allow you to maintain those charges. Now that you've generated parameters for your ligand, these files then need to be added to your leap setup.

Here is an example leap script.

```
# Load constant ph parameters
source leaprc.constph

# Gaff params
source leaprc.gaff

# Load ligand parameters
ligand = loadMol2 ligand.gaff.mol2
loadAmberParams ligand.gaff.frcmod

# Load the PDBs
protein = loadPDB protein.pdb

# Combine into one complex
complex = combine { protein ligand }

# Validate the input
check complex

# Calculate the total charge, for logging purposes
charge complex

# Write parameters.
saveAmberParm  complex  complex.prmtop complex.inpcrd

# Write PDB files
savepdb protein complex.pdb

# Exit, make sure not to forget this part
quit
```

Todo:

- In the current version of the code, ligands can not be treated using constant-pH methodologies.

1.4.5 Generating parameters for amino acid protonation states

The last step to generate input for the constant-pH simulation is to generate a `.cpin` file for your protein. This file contains the parameters for the different protonation states of the amino acids in the system.

A `.cpin` file can be generated by `cpinutil.py`, which is also distributed as part of Ambertools.

```
cpinutil.py -resnames HIP GL4 AS4 TYR LYS CYS -p complex.prmtop -o complex.cpin
```

1.5 Running a constant-pH MD simulation

The `protons` package extends OpenMM with the capability to alter the configuration of protons in simulation. This allows for sampling over multiple protonation states and prototropic tautomers of amino acid residues and small molcules. These effects have been shown as important in multiple biological systems ([Czodrowski2007a], [Czodrowski2007b], [Steuber2007], [Neeb2014]), and now you can include them in OpenMM simulations.

1.5.1 Setting up the `AmberProtonDrive` class

In order to run a simulation containing multiple protonation states and tautomers, you can use `AmberProtonDrive`. This object is responsible for keeping track of the current system state, and updates the OpenMM context with the correct parameters. It uses an instantaneous Monte Carlo sampling method [Mongan2004] to update implicit solvent systems. Explicit solvent systems can be updated using NCMC [Stern2007], [Nilmeier2011], [Chen2015]. The driver maintains a dictionary of all possible protonation states and tautomers of each residue in the simulation system, and their parameters in the AMBER99 constant-pH force field [Mongan2004].

To instantiate the `AmberProtonDrive`, you need a prmtop file (loaded using `simtk.openmm.app.AmberPrmtopFile`), and an OpenMM system (`simtk.openmm.openmm.System`).

Additionally, a `.cpin` file (generated by `cpinutil.py`, which is part of Ambertools) is needed to provide information on amino acid parameters. Please see the *Preparing systems for constant-pH simulation* section for instructions how to generate input files.

In order to perform NCMC, you need to provide the integrator that you will use for the regular MD part as well. The implementation of NCMC uses a Velocity Verlet integrator, so make sure that your integrator is not timewise incompatible like for instance Leapfrog integrators.

```
from protons import AmberProtonDrive
from simtk import unit, openmm
from simtk.openmm import app
from openmmtools.integrators import VelocityVerletIntegrator

# Load a protein system
prmtop = app.AmberPrmtopFile('protein.prmtop')
cpin_filename = 'protein.cpin'

# System settings
pH = 7.4
temperature = 300.0 * unit.kelvin

# Define an integrator for the system
integrator = VelocityVerletIntegrator(1.0 * unit.femtoseconds)

# Create an implicit solvent system from the AMBER prmtop file
system = prmtop.createSystem(implicitSolvent=app.OBC2, nonbondedMethod=app.NoCutoff,
↪constraints=app.HBonds)
# Create the driver that will update the protons on the system and track its state
driver = AmberProtonDrive(system, temperature, pH, prmtop, cpin_filename, integrator,
↪pressure=None, ncmc_steps_per_trial=0, implicit=True)

# Create a simulation object using the correct integrator
simulation = app.Simulation(prmtop.topology, system, driver.compound_integrator,
↪platform)
```

Next, you will need to provide reference free energies for each residue in solvent.

1.5.2 Solvent reference state calibrations

In order to simulate a protein or small molecule with multiple protonation states and/or tautomers, it is necessary to calculate the free energy difference between a reference state. This free energy can depend on the temperature, solvent model, pressure, pH and other factors. It is therefore imperative that this is run whenever you've changed simulation settings.

Individual states are denoted with a subscript k. We use self-adjusted mixture sampling (SAMS) to calculate g_k, a reference free energy. The g_k s correct for (electrostatic) force field contribution to the free energy difference between the reference states, so that the populations produced in simulation match what is expected from the pH dependence, or tautomeric populations.

Residues

The package supports automatic g_k calculations the following residues by default, denoted by the residue name with the max number of protons added. The reference state is taken to be the state of a single capped amino acids in water.

- Glutamic acid, `GL4` (pKa=4.4)
- Aspartic acid, `AS4` (pKa=4.0)
- Histidine, `HIP` (pKa delta=6.5, pKa epsilon = 7.1)
- Tyrosine, `TYR` (pKa=9.6)
- Cysteine, `CYS` (pKa=8.5)
- Lysine, `LYS` (pKa=10.4)

To automatically calibrate all amino acids available in a system, one can use the `AmberProtonDrive.calibrate()` method.

The `AmberProtonDrive.calibrate()` method

The `AmberProtonDrive.calibrate()` method will set this up automatically for the settings you have provided.

```
calibration_results = driver.calibrate()
```

It will automatically perform a free energy calculation using self-adjusted mixture sampling (SAMS) to calculate the reference free energy for each state g_k. While this is conveniently carried out automatically, this may take quite some time (minutes to 2-hours on a GTX-Titan per unique residue type). We are experimenting with a setup that can perform calibration in parallel so that you can run calibration more efficiently. If you store these results, you can reload them in a subsequent run.

```
# Pre-calculated values
# temperature = 300.0 * unit.kelvin
# pressure = None
# timestep = 1.0 * unit.femtoseconds
# pH = 7.4
# Amber 99 constant ph residues

calibration_results = {'as4': np.array([3.98027947e-04,  -3.61785292e+01,  -3.
↪98046143e+01,
                                        -3.61467735e+01,  -3.97845096e+01]),
                       'cys': np.array([7.64357397e-02,   1.30386793e+02]),
                       'gl4': np.array([9.99500333e-04,  -5.88268681e+00,  -8.
↪98650420e+00,
```

(continues on next page)

(continued from previous page)

```
                                        -5.87149375e+00,  -8.94086390e+00]),
                    'hip': np.array([2.39229276,   5.38886021,  13.12895206]),
                    'lys': np.array([9.99500333e-04,  -1.70930870e+01]),
                    'tyr': np.array([6.28975142e-03,   1.12467299e+02])}

driver.import_gk_values(calibration_results)
```

Warning: When reusing calibrated values, you must make sure that you are using the exact same force field, pH and other properties of the system. If you are not sure, we recommend that you rerun the calibration.

For more in depth explanation of the calibration procedure, please see *Advanced calibration options*.

Now that g_k values have been calibrated, you are ready to run a simulation.

1.5.3 Running the simulation

After calibration, you can start running a simulation. Decide on the number of timesteps, and the frequency of updating the residue states. To propagate in regular dynamics, just use `simulation.step`. The residue states are updated using the `AmberProtonDrive.update()` method. This method selects new states using a Monte Carlo procedure, and modifies the parameters in your simulation context to reflect the selected states.

```
nupdates, mc_frequency = 10000, 6000

for iteration in range(1, nupdates):
    simulation.step(mc_frequency) # MD
    driver.update(simulation.context)  # protonation
```

In this example, every 6000 steps of molecular dynamics, the residue states are driven once. This gets repeated for a total of 10000 iteration.

1.5.4 Tracking the simulation

This section and the API still need to be written.

1.5.5 Basic example

Below is a basic example of how to run a simulation using the AmberProtonDrive without using the calibration API.

```
  from simtk import unit, openmm
  from simtk.openmm import app
  from protons import AmberProtonDrive
  import numpy as np
  from openmmtools.integrators import VelocityVerletIntegrator
  from sys import stdout


  # Import one of the standard systems.
  temperature = 300.0 * unit.kelvin
  timestep = 1.0 * unit.femtoseconds
  pH = 7.4
```

(continues on next page)

(continued from previous page)

```
platform = openmm.Platform.getPlatformByName('CUDA')

prmtop = app.AmberPrmtopFile('complex.prmtop')
inpcrd = app.AmberInpcrdFile('complex.inpcrd')
positions = inpcrd.getPositions()
topology = prmtop.topology
cpin_filename = 'complex.cpin'
integrator = VelocityVerletIntegrator(timestep)

# Create a system from the AMBER prmtop file
system = prmtop.createSystem(implicitSolvent=app.OBC2, nonbondedMethod=app.NoCutoff,
↪ constraints=app.HBonds)
# Create the driver that will track the state of the simulation and provides the␣
↪updating API
driver = AmberProtonDrive(system, temperature, pH, prmtop, cpin_filename,␣
↪integrator, pressure=None, ncmc_steps_per_trial=0, implicit=True)

# Create an OpenMM simulation object as one normally would.
simulation = app.Simulation(topology, system, driver.compound_integrator, platform)
simulation.context.setPositions(positions)
simulation.context.setVelocitiesToTemperature(temperature)

# pre-equilibrated values.
# temperature = 300.0 * unit.kelvin
# pressure = None
# timestep = 1.0 * unit.femtoseconds
# pH = 7.4
# Amber 99 constant ph residues, converged to threshold of 1.e-7

calibration_results = {'as4': np.array([3.98027947e-04,  -3.61785292e+01,  -3.
↪98046143e+01,
                                        -3.61467735e+01,  -3.97845096e+01]),
                       'cys': np.array([7.64357397e-02,   1.30386793e+02]),
                       'gl4': np.array([9.99500333e-04,  -5.88268681e+00,  -8.
↪98650420e+00,
                                        -5.87149375e+00,  -8.94086390e+00]),
                       'hip': np.array([2.39229276,   5.38886021,  13.12895206]),
                       'lys': np.array([9.99500333e-04,  -1.70930870e+01]),
                       'tyr': np.array([6.28975142e-03,   1.12467299e+02])}

driver.import_gk_values(calibration_results)

# 60 ns, 10000 state updates
niter, mc_frequency = 10000, 6000
simulation.reporters.append(app.DCDReporter('trajectory.dcd', mc_frequency))

for iteration in range(1, niter):
    simulation.step(mc_frequency) # MD
    driver.update(simulation.context)  # protonation
```

1.6 Advanced calibration options

Keyword arguments supplied to the `protons.AmberProtonDrive.calibrate()` method can be used to tweak the calibration settings. This feature is recommended for experts only. The calibration settings are passed

to this function:

1.7 The ligutils submodule

This submodule will contain the features that enable parametrizing ligands for constant-ph simulation.

1.7.1 Treating small molecules

Todo:

- Small molecules are currently work in progress.
- Will need to add support to determine populations and molecular data from other sources than Epik.
- Adding openeye am1 bcc charge support as an optional feature for those that have openeye available

---

Several steps are needed in order to prepare a small molecule for simulation using the constant-pH code.

1. Protonation states and tautomers of the ligand need to be enumerated. This relies on Epik at the moment.
2. Molecule states need to be parametrized using antechamber.
3. All states need to be combined into a single residue template as an ffxml file.
4. Calibrate reference free energies of the molecule in solvent to return the expected state populations in the reference state.

This code provides an automated pipeline to sequentially perform all of these steps using this handy function!

1.8 API documentation

1.8.1 The protons module

AmberProtonDrive class

1.9 References

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[Czodrowski2007a] Czodrowski, Paul; Sotriffer, Christoph A.; Klebe, Gerhard (2007a): Atypical protonation states in the active site of HIV-1 protease: a computational study. In Journal of chemical information and modeling 47 (4), pp. 1590–1598. DOI: 10.1021/ci600522c.

[Czodrowski2007b] Czodrowski, Paul; Sotriffer, Christoph A.; Klebe, Gerhard (2007b): Protonation changes upon ligand binding to trypsin and thrombin: structural interpretation based on pK(a) calculations and ITC experiments. In Journal of molecular biology 367 (5), pp. 1347–1356. DOI: 10.1016/j.jmb.2007.01.022.

[Neeb2014] Neeb, Manuel; Czodrowski, Paul; Heine, Andreas; Barandun, Luzi Jakob; Hohn, Christoph; Diederich, Francois; Klebe, Gerhard (2014): Chasing protons: how isothermal titration calorimetry, mutagenesis, and pKa calculations trace the locus of charge in ligand binding to a tRNA-binding enzyme. In Journal of medicinal chemistry 57 (13), pp. 5554–5565. DOI: 10.1021/jm500401x.

[Steuber2007] Steuber, Holger; Czodrowski, Paul; Sotriffer, Christoph A.; Klebe, Gerhard (2007): Tracing changes in protonation: a prerequisite to factorize thermodynamic data of inhibitor binding to aldose reductase. In Journal of molecular biology 373 (5), pp. 1305–1320. DOI: 10.1016/j.jmb.2007.08.063.

[Mongan2004] Mongan J, Case DA, and McCammon JA. Constant pH molecular dynamics in generalized Born implicit solvent. J Comput Chem 25:2038, 2004. http://dx.doi.org/10.1002/jcc.20139

[Stern2007] Stern HA. Molecular simulation with variable protonation states at constant pH. JCP 126:164112, 2007. http://link.aip.org/link/doi/10.1063/1.2731781

[Nilmeier2011] Nonequilibrium candidate Monte Carlo is an efficient tool for equilibrium simulation. PNAS 108:E1009, 2011. http://dx.doi.org/10.1073/pnas.1106094108

[Greenwood2010] Greenwood, J. R.; Calkins, D.; Sullivan, A. P.; Shelley, J. C., "Towards the comprehensive, rapid, and accurate prediction of the favorable tautomeric states of drug-like molecules in aqueous solution," J. Comput. Aided Mol. Des., 2010, 24, 591-604

[Shelley2007] Shelley, J.C.; Cholleti, A.; Frye, L; Greenwood, J.R.; Timlin, M.R.; Uchimaya, M., "Epik: a software program for pKa prediction and protonation state generation for drug-like molecules," J. Comp.-Aided Mol. Design, 2007, 21, 681-691

[Epik2016] Schrödinger Release 2016-3: Epik, Schrödinger, LLC, New York, NY, 2016.

[Amber2016] D.A. Case, R.M. Betz, W. Botello-Smith, D.S. Cerutti, T.E. Cheatham, III, et al. (2016), AMBER 2016, University of California, San Francisco. http://ambermd.org/

[Wang2004] Wang, Junmei, et al. "Development and testing of a general amber force field." Journal of computational chemistry 25.9 (2004): 1157-1174.

[Chen2015] Chen, Yunjie, and Benoît Roux. "Constant-pH hybrid nonequilibrium molecular dynamics–monte carlo simulation method." Journal of chemical theory and computation 11.8 (2015): 3919-3931.