BornProfiler Documentation

Release 0.9.2+26.g1869d7e.dirty

Lennard van der Feltz, Kaihsu Tai, Oliver Beckstein

Feb 22, 2022

Contents

BornProfiler is a Python package to set up calculations of the electrostatic free energy of an ion in a membrane protein for the Poisson-Boltzmann solver APBS.

CHAPTER 1

Contents

1.1 Overview

BornProfiler is a collection of scripts to set up Poisson-Boltzmann electrostatic calculations for the APBS package, in particular calculations of the electrostatic solvation free energy of an ion along a pathway in a membrane protein (the so-called *Born profile*).

1.1.1 Features

The BornProfiler package helps setting up Poisson-Boltzmann calculations of the electrostatic potential of mean force of an ion in a pore or channel under the influence of a membrane. The membrane is modelled as a dielectric slab of epsilon=2.

- Provide a path (list of coordinates) and a PQR file of the protein as input.
- A membrane can be defined with arbitrary thickness, z-position, and dielectric. A headgroup region can also be defined with a different dielectric constant.
- Define all input parameters in a compact parameter file so that there is always a record of the exact calculation setup available.
- Born radii for all ions from the Rashin & Honig paper [Rashin1985] are included; just select the ion in the input file.
- Born radii for H3O+, OH- (and H+... for testing) have been derived from the solvation free energies in [Pliego2000] directly via the Born equation. USE AT YOUR OWN RISK!!
- Customize run scripts and queuing system submission scripts by providing your own templates.

1.1.2 History and Contributions

Based on Kaihsu Tai's Python rewrite (Poisson-Boltzmann profile for an ion channel) of the original `placeion.sh` and `analyze.sh` bash scripts by Kaihsu Tai and Oliver Beckstein.

Uses material from the APBS Wiki (PMF of a helix in a membrane) and contains a modified version of Michael Grabe's `draw_membrane2` from APBSmem.

1.2 Building and installing BornProfiler

BornProfiler consists of a Python package `bornprofiler` and a a stand-alone executable `draw_membrane2`. The Python package is needed to set up the calculations and `draw_membrane2` is needed as a helper tool for `apbs`; hence it needs to be installed on the same machine where `apbs` is going to run.

1.2.1 Required pre-requisites

- python >= 2.7 and < 3
- NumPy
- a C compiler such as GNU gcc
- APBS >= 1.3

1.2.2 Installation from Source

Unpack the tar ball:

```
tar zxvf BornProfiler-0.9.2.tar.gz
```

Install the python module and scripts:

```
cd BornProfiler
python setup.py install
```

(see `python setup.py install --help` for guidance on what your options are.)

Compile the customized (and improved) version of `draw_membrane` named `draw_membrane2`:

```
mkdir BUILD && cd BUILD
cmake -D CMAKE_INSTALL_PREFIX=$HOME -D CMAKE_BUILD_TYPE=Release ../src/drawmembrane
make
make install
```

The `make install` step will install the executable `draw_membrane2a` under `CMAKE_INSTALL_PREFIX/bin`; change `CMAKE_INSTALL_PREFIX` if you prefer another location.

(cmake is not really needed; if you don't have it try the following:

```
gcc ./src/drawmembrane/draw_membrane2a.c -o draw_membrane2a -lm -lz
```

and install manually in a place where you or your shell can find it.)

Note: `draw_membrane2a` also needs to be installed on the machine where you want to run your BornProfiler jobs: it will run together with `apbs`. If you are going to run you calculations on a cluster then `draw_membrane2a` (and `apbs`) need to be *both* installed on the cluster.

1.2.3 Configuration

Finalize your installation by running

```
apbs-bornprofile-init.py
```

This should tell you that it set up a configuration file `~/.bornprofiler.cfg` and a number of directories.

The default `~/.bornprofiler.cfg` looks like this:

```
[DEFAULT]
configdir = ~/.bornprofiler
templatesdir = %(configdir)s/templates
qscriptdir = %(configdir)s/qscripts

[executables]
apbs = apbs
drawmembrane = draw_membrane2a
```

The file can be edited in a text editor. For instance, one can add the full path to the `apbs` and `draw_membrane2a` executable binaries.

Any other variables used in run configuration input files can also be added here and will be used as defaults.

Advanced use: You can drop templates for run scripts into *qscriptdir* and have the BornProfiler package pick them up automagically.

1.3 User documentation

This documentation will tell you what you can achieve with **BornProfiler**, sketch out the background, what the typical workflow looks like, and discuss some examples.

1.3.1 Basic operations

Simple example that introduce important concepts and functionality.

Potential surface with membrane

We want to calculate the electrostatic potential around a membrane protein. We need the structure of the protein, assign charges and radii (using the pdb2pqr tool), and create a low-dielectric slab that simulates the membrane. We then run apbs to solve the Poisson-Boltzmann equation and visualize the results.

PQR file

Have a PQR file for your membrane protein of interest available. Getting it from the Orientations of Proteins in Membranes database (OPM) is convenient because you can get a sense for the likely position of the membrane. In the following we assume we use the GLIC channel (a proton gated cation channel) with OPM/PDB id 3p4w as an example. Download the file from OPM and create the PQR file with **`pdb2pqr`**:

```
wget https://storage.googleapis.com/opm-assets/pdb/3p4w.pdb
pdb2pqr --ff CHARMM --whitespace --drop-water 3p4w.pdb 3p4w.pqr
```

Membrane position

The membrane will be simulated as a low-dielectric brick-shaped slab.

Obtain the boundaries of the membrane that OPM suggested: The file contains "DUM" atoms that show the position of the membrane surfaces. The protein is oriented such that the membrane is in the X-Y plane so we extract the minimum and maximum Z component of the DUM atoms using some Unix command line utilities:

```
grep '^HETATM.*DUM'  3p4w.pdb | cut -b 47-54 | sort -n | uniq
```

which gives

```
-16.200
 16.200
```

i.e., the membrane is located between -16.2 Å and +16.2 Å with a thickness of 32.4 Å. The center of the membrane is at `z = 0` and the bottom is at `z = -16.2`.

Pore dimensions

Ion channels have aqueous pathways. They should be represented by a high dielectric environment that is also accessible to ions.

In order to add the pore regions we need to get approximate dimensions of the pore, which is represented as a truncated cone with a center in the X-Y plane and potentially differing radii at the top and bottom of the membrane plane.

You can run **HOLE** to get radii and positions.

For this tutorial we simply load the structure in a visualization tool such a **VMD**, **pymol** or **Chimera** and estimate a diameter of about 20 Å at the center of the protein, which is at `x=0` and `y=0` in the plane of the membrane.

Set up run

Generate a template input file `3p4w.cfg` for the protein:

```
apbs-mem-potential.py --template 3p4w.cfg
```

You only need to keep the sections

- `[environment]`
- `[membrane]`
- `[potential]`

because everthing else is taken from your global configuration file (`~/.bornprofiler.cfg`).

Edit the template in `[environment]` section and the set pqr file.

```
[environment]
pqr = 3p4w.pqr
```

Edit the template in the `[membrane]` section to add data for the membrane.

- `lmem` is the thickness, 32.4 Å; the membrane is set to a dielectric constant of `mdie`.
- `zmem` is the *bottom* of the membrane, `z=-16.2`
- `vmem` is the cytosolic membrane potential (in ???); here we leave it at 0.

- `headgroup_l` is the thickness of the headgroup region with dielectric constant `headgroup_die`. Here we keep it zero for simplicity, but if you have additional data you can set it to a non-zero value. (See, for example Fig 2c in [Stelzl2014]). The total membrane thickness is still `lmem` and the hydrophobic core is then `lmem - 2*headgroup_l`.
- **channel exclusion zone**: a stencil with dielectric constant `cdie` (by default, the solvent dielectric constant) in the shape of a truncated cone can be cut from the membrane. Its axis is parallel to the membrane normal and centered at absolute coordinates `x0_r` and `y0_r`. Alternatively, the center can be given relative to the center of geometry of the protein, with an offset `dx_r` and `dy_r`. The default is to position the exclusion zone at the center of the protein.

```
[membrane]
rtop = 10
rbot = 10
x0_r = None
y0_r = None
dx_r = 0
dy_r = 0
cdie = %(solvent_dielectric)s
headgroup_die = 20
headgroup_l = 0
mdie = 2
vmem = 0
lmem = 32.4
zmem = -16.2
```

The `[potential]` block sets the dimensions of the grid.

Run calculation

Once all information is collected in the cfg file, one runs

```
apbs-mem-potential.py 3p4w.cfg
```

This will create input files for **`apbs`** and run **`drawmembrane2a`** when necessary.

The output consists of *dx* files of the potential (in kT/e). Typically, these files a gzip-compressed to save space. For most external tools, uncompress them with **`gunzip`**.

In particular the following files are of interest:

- **pot_membraneS.dx.gz**: the potential on the grid in kT/e (calculated with membrane included)
- **dielxSm.dx.gz**: the dielectric map with membrane; visualize to verify that there are regions of different dielectric constants. (APBS needs maps that are shifted in X, Y, and Z; for visualization purposes, anyone is sufficient)
- **kappaSm.dx.gz**: map of the exclusion zone (with membrane)
- **pot_bulksolventS.dx**: the potential *without a membrane* (for comparison to see the effect of the membrane); other files without membrane have names similar to the afore mentione ones but *without "m" as the last letter of the name before the .dx.gz*.

Visualization

Uncompress the **pot_membraneS.dx.gz** file

```
gunzip pot_membraneS.dx.gz
```

and load the PQR file `3p4w.pqr` (or the PDB file `3p4w.pdb`) and the DX file `pot_membraneS.dx` in your favorite visualization tool. Contour the density at, for example, –5 kT/e and +5 kT/e.

A simple Born profile

TODO: Outline the problem of ion permeation, discuss simple example and show how this package can solve the problem. Choose something very simple such as nAChR or GLIC.

1.3.2 Background

TODO: Describe the theoretical background and say how individual steps are done in the package.

1.3.3 Workflow

TODO: describe the individual steps

Sample point generation

Path (1D)

- str8path
- HOLE

Volume (3D)

- HOLLOW (custom version)

Parameter settings

Radii and Charges

Use pdb2pqr to generate the input PQR file.

BornProfiler run input file

- ...
- membrane position
- exclusion zone

Queuing system script

Discuss example, highlight what needs to be customized and how.

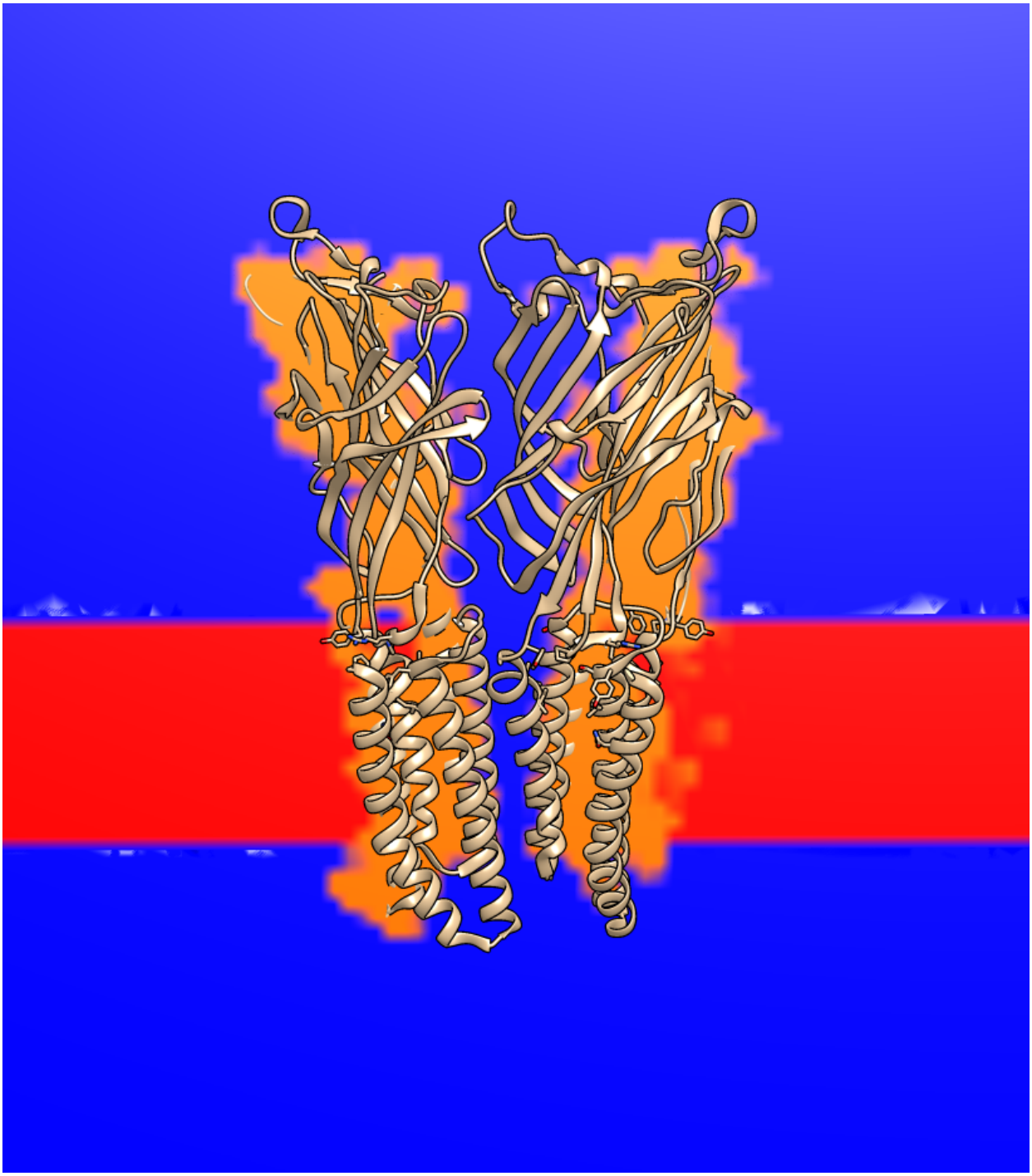

Fig. 1: GLIC channel dielectric map `dielxSm.dx` visualized together with `3p4w.pdb`. Epsilon 2 (membrane) is red, protein (10) is orange, solvent (80) is blue. Visualized and rendered with UCSF Chimera.

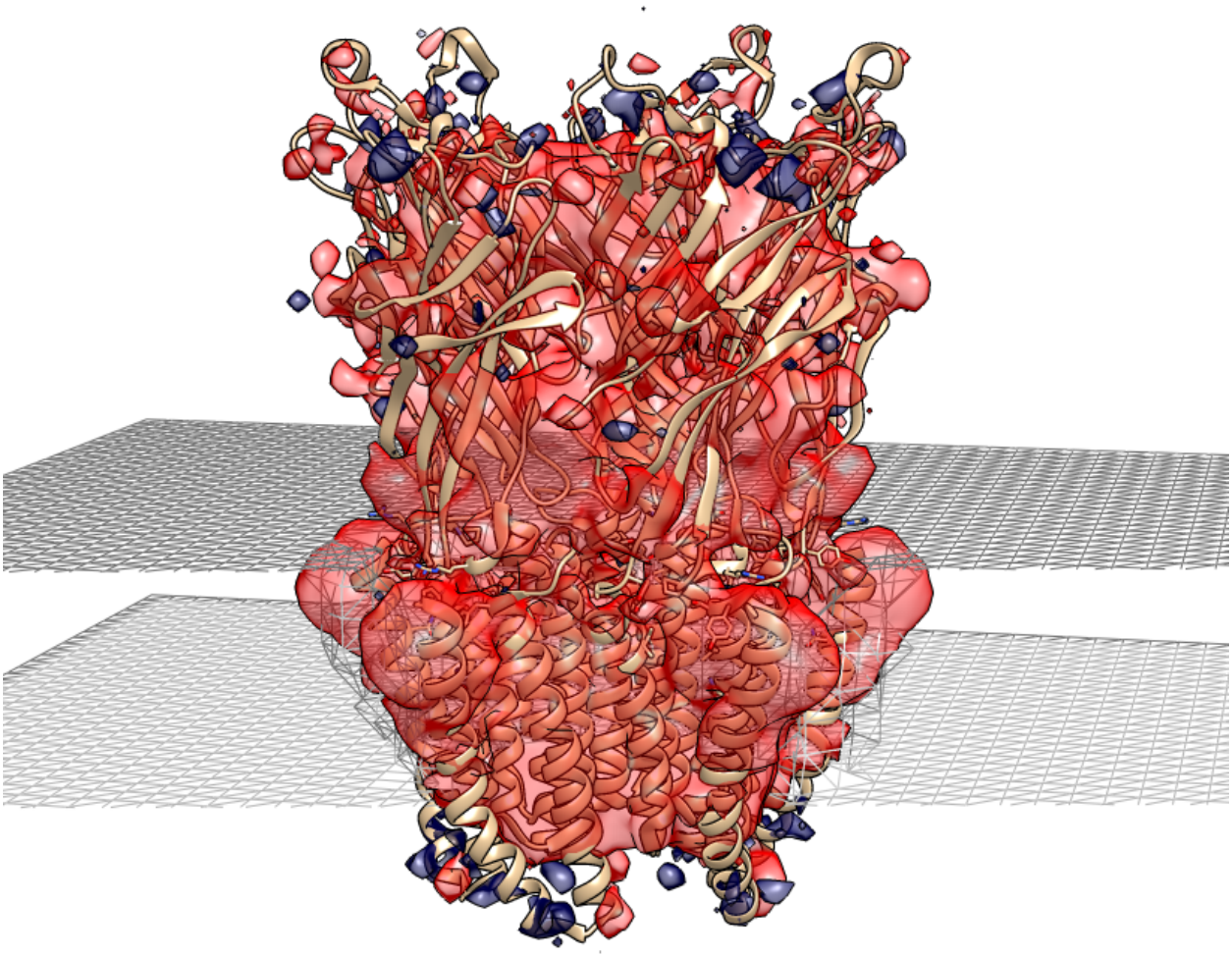

Fig. 2: GLIC channel electrostatic potential `pot_membraneS.dx` visualized with the protein structure `3p4w.pdb`. The potential isocontour surface at –5 kT/e is shown in red, and the one at +5 kT/e in blue. The membrane dielectric region is shown as a gray mesh. Visualized and rendered with UCSF Chimera.

Generate window input files

:: apbs-bornprofile-mplaceion

Run jobs

Manually or typically through a queuing system.

Analyze and visualize data

:: apbs-bornprofile-analyze

Distinguish 1D/3D.

Talk about using Chimera to analyze 3D energy landscapes and add some tips & tricks.

1.3.4 Examples

Two examples are part of the BornProfiler source distribution.

Parsegian

Nicotinic acetylcholine receptor (nAChR)

1.4 Developer documentation

Most users will likely use **BornProfiler** through the provided scripts. However, in order to extend functionality and develop new tools one can also use the `bornprofiler` Python module as a library.

Content:

1.4.1 Core functionality — `bornprofiler.core`

Base classes around which the whole **BornProfiler** package is built,

class bornprofiler.core.**BPbase**

Provide basic infra structure methods for bornprofiler classes.

Defines the file name API.

- simply number files, do not use z or other data in filename;
- assume standard python 0-based indexing and naming
- filenames are generated and typically only take *num*, the window number, as an argument

---

Note: This class cannot be used on its own and it relies on other attributes being set by the parent class.

---

get_taskid(*num*)

Return 1-based Sun Gridengine taskid for an array job

readPQR()

Read PQR file and determines protein centre of geometry

readPoints()
Read positions for Born ions from data file.

Tries to be smart and autodetect standard x-y-z dat file or pdb.

writePQRs(*windows=None*)
Generate input pqr files for all windows and store them in separate directories.

bornprofiler.core.**IONS = {'Ag': {'atomname': 'Ag+', 'charge': 1.0, 'name': 'Ag', 'radiu**
Rashin&Honig ion data as a dict, read from `templates/bornions.dat`.

class bornprofiler.core.**Ion**(*name*, *symbol*, *atomname*, *radius*, *charge*)
Represent parameters for an ion.

class bornprofiler.core.**MPlaceion**(**args*, ***kwargs*)
Generate all input files for a Born profile calculation WITH a membrane

Setup Born profile with membrane.

MPlaceion(paramfile[,basedir])

MPlaceion(pqr,points[,memclass,jobName,ionName,ionicStrength,temperature,script,arrayscript,basedir])

Arguments

parameterfile ini-style file containing all run parameters

pqr PQR file **DEPRECATED**

points data file with sample points **DEPRECATED**

Keywords

memclass a class or type `APBSMem` to customize draw_membrane **DEPRECATED**

jobName name of the run, used as top directory name and as unique identifier [mbornprofile]

ionName name of the Rashin&Honig ion to calculate; any ion in IONS works [Na]

ionicStrength concentration of monovalent NaCl solution in mol/l [0.15]

temperature temperature in K [300.0]

script template for a submission script (contains 'abps %(infile)s > %(outfile)s' or similar; %(jobname)s is also replaced). Can be (a) a local file, (b) a file stored in he user template dir, (c) a bundled template file.

arrayscript template for a queuing system array script; contains the the placeholders %(jobName)s and %(jobArray)s; jobs are stored in the bash array 'job' so there should be a line 'declare -a job'. Window numbers correspond to the task ids (SGE) ar array ids (PBS) in the job array. See `templates/q_array.sge` as an example.

basedir top directory of set up, defaults to [.]

generate(*windows=None*, *run=False*)
Set up all input files for Born calculations with a membrane.

generate([windows[,run]])

The optional parameter *windows* allows one to select a subset of windows instead of all the points in the sample points file; it can be a single number or a list.

Setting *run* to `True` immediately generates setup files, in particular it runs **apbs** and **draw_membrane2a** in order to add the membrane to the system. Because this can take a long time for a large number of windows it is also possible to only generate a bash script for each window and defer the setup (*run* = `False`, the default).

generateMem(*windows=None*, *run=False*)
Generate special diel/kappa/charge files and mem_placeion.in for each window.

The APBS calculation is set up for manual focusing in three stages:

1. **L** is a coarse grid and centered on the protein
2. **M** is a medium grid, centered on the ion, and using focusing (the boundary values come from the **L** calculation)
3. **S** is the finest grid, also centered and focused on the ion

The ion positions ("windows") are simply sequentially numbered, starting at 1, as they appear in the input file. Each window is set up in its own directiroy (called "wNNNN" where NNNN is the 4-digit, zero-padded number).

Warning: At the moment it is not checked that the "inner" focusing region M are always contained in the outer region L; it's on the TODO list.

Keywords

windows [None, number, or list] window number or list of window numbers to generate. If set to None (the default) then all windows are generated. Window numbers start at 0 and end at numPoints-1.

run [bool] True: immediately generate files (can take a while); False defer file generation and just write a script [False]

get_MemBornSetup(*num*)
Return the setup class for window *num* (cached).

The method is responsible for passing all parameters to downstream classes that are used to generate individual windows. It instantiates an appropriately set up class for each window and caches each class. Classes are indexed by *num* (which is the unique window identifier).

process_bornprofile_kwargs()
Hook to manipulate bornprofile_kwargs.

 # set exclusion zone centre to the protein centroid (unless *x0_R* and/or *y0_R* are set in the run input cfg file)

shift exclusion zone centre by *dx_R* and *dy_R* # filter *remove_bornprofile_keywords*

remove_bornprofile_keywords = ('fglen', 'dx_R', 'dy_R')
These keywords are read from the runinput file but should not be passed on through the bornprofile_keywords mechanism. See meth:*process_bornprofile_keywords*.

schedule = {'dime': [(129, 129, 129), (129, 129, 129), (129, 129, 129)], 'glen': [(2
Schedule is run from first to last: L -> M -> S choose dime compatible with nlev=4 (in input file)

writeJob(*windows=None*)
Write the job script.

Can be done selectively for a subset of windows and then the global array script will also only contain this subset of windows.

class bornprofiler.core.**Placeion**(**args*, ***kwargs*)
preparing job for APBS energy profiling by placing ions

generate()
Generate all input files.

readPQR()
Read PQR file and determine bounding box and centre of geometry.

bornprofiler.core.**ngridpoints**(*c*, *nlev=4*)
The allowed number of grid points.

For mg-manual calculations, the arguments are dependent on the choice of *nlev* for dime by the formula

n = c*2**(nlev + 1) + 1

where n is the dime argument, c is a non-zero integer, lev is the nlev value. The most common values for grid dimensions are 65, 97, 129, and 161 (they can be different in each direction); these are all compatible with a nlev value of 4. If you happen to pick a "bad" value for the dimensions (i.e., mismatch with nlev), the APBS code will adjust the specified dime downwards to more appropriate values. This means that "bad" values will typically result in lower resolution/accuracy calculations!

1.4.2 APBS calculations: Membrane simulations — bornprofiler.electrostatics

APBSmem facilitates the setup of electrostatics calculations with a membrane. It implements the workflow from the APBS tutorial 'Helix in a membrane'. The **draw_membrane2a** binary is required to to add a low-dielectric (eps=2) region and sets protein dielectric to eps=10.

Note: Paths to draw_membrane2a and apbs are set in the configuration file ~/.bornprofiler.cfg:

```
apbs = %(APBS)r
draw_membrane2 = %(DRAWMEMBRANE)r
```

APBSnomem simply gives electrostatics info for the protein/solvent system for comparison (and does not need **draw_membrane2a**).

See also:

APBSmem https://apbsmem.sourceforge.io/

class bornprofiler.electrostatics.**APBSmem**(**args*, ***kwargs*)
Represent the apbsmem tools.

Run for S, M, and L grid (change suffix).

Note: see code for kwargs

Set up calculation.

APBSmem(pqr, suffix[, kwargs])

Arguments

- arguments for drawmembrane (see source)
- temperature: temperature [298.15]
- **conc: ionic concentration of monovalent salt with radius 2 A** in mol/l [0.1]
- dime: grid dimensions, as list [(97,97,97)]
- glen: grid length, as list in Angstroem [(250,250,250}]

generate()
Setup solvation calculation.

1. create exclusion maps (runs apbs)
2. create membrane maps (drawmembrane)
3. create apbs run input file

vars = None
"static" variables required for a calculation

class bornprofiler.electrostatics.**APBSnomem**(**args*, ***kwargs*)
Represent the apbsnomem tools.

Run for S, M, and L grid (change suffix).

Note: see code for kwargs

Set up calculation.

APBSnomem(pqr, suffix[, kwargs])

Arguments

- temperature: temperature [298.15]
- **conc: ionic concentration of monovalent salt with radius 2 A** in mol/l [0.1]
- dime: grid dimensions, as list [(97,97,97)]
- glen: grid length, as list in Angstroem [(250,250,250}]

generate()
Setup solvation calculation.

1. create exclusion maps (runs apbs)
2. create apbs run input file

vars = None
"static" variables required for a calculation

class bornprofiler.electrostatics.**BornAPBSmem**(**args*, ***kwargs*)
Class to prepare a single window in a manual focusing run.

Set up calculation.

BornAPBSmem(protein_pqr, ion_pqr, complex_pqr[, kwargs])

Additional arguments:

- apbs_script_name: name on the xxx.in script [mem_placeion.in]
- drawmembrane arguments (see source)
- temperature: temperature [298.15]
- **conc: ionic concentration of monovalent salt with radius 2 A** in mol/l [0.1]
- **dime: grid dimensions, as list with 1 or 3 entries; if o1 entry** then the same dime is used for all focusing stages [(97,97,97)]
- **glen: grid length, as list in Angstroem, must contain three triplets** [(250,250,250),(100,100,100),(50,50,50)]

- **runtype: for standard Born calculations use 'with_protein'; if one only wants to** look at geometrically defined dielectric regions and uncharged proteins (see the `example/Parsegian`) then use 'memonly' ['with_protein']

generate (*run=False*)
Setup solvation calculation.

If *run* = `True` then runs **apbs** and **draw_membrane2** (which can take a few minutes); otherwise just generate scripts (default).

***run* = True**

1. create exclusion maps (runs **apbs** through `run_apbs()`)
2. create membrane maps (**draw_membrane2a** through `run_drawmembrane2()`)
3. create apbs run input file

***run* = False**

1. write a bash script that calls **apbs** and **draw_membrane2** and which can be integrated into a window run script for parallelization.
2. create apbs run input file

suffices = ('L', 'M', 'S')
Suffices of file names are hard coded in templates and should not be changed; see `templates/mdummy.in` and `templates/mplaceion.in`. The order of the suffices corresponds to the sequence in the schedule.

vars = None
"static" variables required for generating a file from a template; The variable names can be found in self.__dict__

bornprofiler.electrostatics.TEMPLATES = [illegible]
cache for template files

1.4.3 Analysis of calculations — `bornprofiler.analysis`

Functions and classes to process the output from the APBS calculations.

class `bornprofiler.analysis.`**AnalyzeElec** (**args*, ***kwargs*)
analyze APBS energy profiling results

plot (*filename=None*, *plotter='matplotlib'*, ***kwargs*)
Plot Born profile.

plot([filename[,plotter[,kwargs ...]]])

Keywords

filename name of image file to save the plot in; with *plotter* = 'Gnuplot' only eps files are supported (I think...)

kwargs other keyword arguments that are passed on to `pylab.plot()`

class `bornprofiler.analysis.`**AnalyzeElec3D** (**args*, ***kwargs*)
analyze APBS energy profiling results

With create=True (default), reads position data from samplepoints file and energies from APBS output files.

With create=False, reads positions and energies from a previous output file.

Grid(*delta*, ***kwargs*)
Package the PMF as a `gridData.Grid` object.

delta should be the original spacing of the points in angstroem.

With a *resample_factor*, the data are interpolated on a new grid with *resample_factor* times as many bins as in the original histogram (See `gridData.Grid.resample_factor()`).

interpolation_order sets the interpolation order for the resampling procedure.

> **Warning:** Interpolating can lead to artifacts in the 3D PMF. If in doubt, **do not resample**.

histogramdd(*delta*, *fillfac=5*)
Histogram PMF on a regular grid.

The spacing *delta* must be the same or larger than used in Hollow; to be on the safe side, just use the value of *grid_spacing* (e.g. 2.0 for 2 A). The histogram grid is chosen large enough to encompass all data points.

If *delta* is bigger than *grid_spacing* or points are not on a regular grid the values are averaged over each bin.

Points without data are filled with fillfac * max(histogram).

Returns histogram, edges (same as `numpy.histogramdd()`)

ranges(*padding=0*)
Returns the range of values in each dimension.

Returns Array *r* of shape (2,3): r[0] contains the smallest and r[1] the largest values.

class bornprofiler.analysis.**Analyzer**(**args*, ***kwargs*)
Base class for analysis

1. read points
2. read energy for each point
3. write data file(s)
4. optional: plot

Developer note: `exporters` contains the logic for exporting to various other formats besides plain column/text such as PDB. For additional exporters, add the entries in the local __init__ *after* calling `super(name, self).__init__(*args,**kwargs)`.

The Born energy W at each point (x,y,z) is stored in `data`, a (4,N) numpy array:

```
X,Y,Z,W = data
```

accumulate()
Read the energy from each datafile and store with the coordinates.

Populates `AnalyzeElec.data`, a (4,N) array where N is the number of windows.

export(*filename=None*, *format='dx'*, ***kwargs*)
Export data to different file format.

The format is deduced from the filename suffix or *format*. *kwargs* are set according to the exporter.

dx histogram on a grid; must provide kwarg *delta* for the grid spacing.

pdb write PDB file with an ion for each sample point and the energy as the B-factor. The kwarg *ion* can be used to set the name/resName in the file (ION is the default).

find_missing_windows()
Return a list of job numbers that have no data corresponding to a point.

read(*filename=None*)
Read datafile *filename* (format: x,y,z,W or z,W).

The data are stored in `data`, a (4,N) array. If only z coordinates are provided then the x and y columns(data[0] and data[1]) are set to 0.

bornprofiler.analysis.**get_files**(*runfile*, *basedir='.'*)
Read *runfile* and return dict with input files and names.

1.4.4 Configuration file handling and setup — bornprofiler.config

The user can set important paths in `~/.bornprofiler.cfg`. The file is automatically created with default values if it does not exist.

Edit the file with a text editor.

In order to restore the default values, simply delete the config file.

Format

The configuration file is in "INI" format: Section titles and variable assignments:

```
[executables]
apbs = apbs
drawmembrane = /path/to/drawmembrane2a

[membrane]
```

Meaning of variables

executables paths to binaries or just the name if they are found via `PATH`

membrane configuration variables for apbs-mem-setup and friends

Accessing the configuration

Important variables are stored in the dict *`configuration`*. Any variable can be accessed via the getter method of the `ConfigParser.SafeConfigParser` instance, `cfg`:

```
from bornprofiler.config import cfg
varname = cfg.get(section, varname)
```

bornprofiler.config.**CONFIGNAME = '/home/docs/.bornprofiler.cfg'**
Default name for the configuration file.

bornprofiler.config.**cfg = <bornprofiler.config.BPConfigParser instance>**
`cfg` is the instance of `BPConfigParser` that makes all global configuration data accessible

bornprofiler.config.**check_APBS**(*name=None*)
Return ABPS version if apbs can be run and has the minimum required version.

Raises error if it cannot be found (OSError ENOENT) or wrong version (EnvironmentError).

bornprofiler.config.**check_drawmembrane**(*name=None*)
Return drawmembrane version or raise EnvironmentError if incompatible version of drawmembrane

bornprofiler.config.**check_setup**()
Check if templates directories are setup and issue a warning and help.

bornprofiler.config.**configuration = {'apbs': 'apbs', 'configfilename': '/home/docs/.bornp**
Dict containing important configuration variables, populated by *get_configuration()* (mainly a shortcut; use *cfg* in most cases)

bornprofiler.config.**get_configuration**(*filename='/home/docs/.bornprofiler.cfg'*)
Reads and parses the configuration file.

Default values are loaded and then replaced with the values from ~/.bornprofiler.cfg if that file exists. The global configuration instance *bornprofiler.config.cfg* is updated.

Normally, the configuration is only loaded when the bornprofiler package is imported but a re-reading of the configuration can be forced anytime by calling *get_configuration()*.

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

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

1. a relative or absolute path,
2. a file in one of the directories listed in *bornprofiler.config.path*,
3. a filename in the package template directory (defined in the template dictionary *bornprofiler.config.templates*) or
4. 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.

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

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

1. a relative or absolute path,
2. a file in one of the directories listed in *bornprofiler.config.path*,
3. a filename in the package template directory (defined in the template dictionary *bornprofiler.config.templates*) or
4. 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.

bornprofiler.config.**path = ['.', '/home/docs/.bornprofiler/qscripts', '/home/docs/.bornpro**
Search path for user queuing scripts and templates. The internal package-supplied templates are always

searched last via *bornprofiler.config.get_templates()*. Modify *bornprofiler.config.path* directly in order to customize the template and qscript searching. By default it has the value `['.', qscriptdir, templatesdir]`. (Note that it is not a good idea to have template files and qscripts with the same name as they are both searched on the same path.)

bornprofiler.config.**resource_basename**(*resource*)
Last component of a resource (which always uses '/' as sep).

bornprofiler.config.**setup**(*filename='/home/docs/.bornprofiler.cfg'*)
Prepare a default BornProfiler global environment.

1) Create the global config file.
2) Create the directories in which the user can store template and config files.

This function can be run repeatedly without harm.

bornprofiler.config.**templates = {'array_ASU_workstations.ge': '/home/docs/checkouts/readtl**
Registry of all template files that come with the package.

1.4.5 Input/Output for the BornProfiler modules — `bornprofiler.io`

The module contains functions and classes to handle common functionality to read and write files.

class bornprofiler.bpio.**PQRReader**(*filename*, ***kwargs*)
Naive implementation of a PQR reader.

class bornprofiler.bpio.**RunParameters**(*filename*, **omissions*, ***defaults*)
All parameters for a BornProfiler or APBSmem run are stored in a INI-style file.

This class accesses these parameters and can create a template with the default values.

The class *guarantees* that all parameters exist; if needed they are populated with default values. Hence it is always possible to access a parameter without having to check if it is there.

Parameters

- zmem : membrane centre (A)
- lmem : membrane thickness (A)
- Vmem : cytosolic potential in kT/e (untested)
- pdie : protein dielectric
- sdie: solvent dielectric
- mdie: membrane dielectric
- Rtop : exclusion cylinder top
- Rbot : exclusion cylinder bottom
- x0_R : exclusion zone centre in X, `None` selects the default
- y0_R : exclusion zone centre in Y, `None` selects the default
- dx_R : shift centre of the exclusion zone in X
- dy_R : shift centre of the exclusion zone in Y
- cdie : dielectric in the channel (e.g. SDIE)
- headgroup_l : thicknes of headgroup region
- headgroup_die : dielectric for headgroup region

- temperature : temperature
- conc : ionic strength in mol/l
- ... and more

Reads and parses the configuration file for a job.

get_apbsmem_kwargs(**args*, ***kwargs*)
Return a dict with kwargs appropriate for `membrane.APBSmem`.

Default values can be supplied in *kwargs*. This method picks unique parameter keys from the relevant sections of the run parameters file (i.e. *bornprofile*, *membrane*, and *environment*).

If args are provided, then either a single value corresponding to the key or a list of such values is returned instead.

get_apbsnomem_kwargs(**args*, ***kwargs*)
Return a dict with kwargs appropriate for `membrane.APBSnomem`.

Default values can be supplied in *kwargs*. This method picks unique parameter keys from the relevant sections of the run parameters file (i.e. *bornprofile* and *environment*).

If args are provided, then either a single value corresponding to the key or a list of such values is returned instead.

get_bornprofile_kwargs(**args*, ***kwargs*)
Return a dict with kwargs appropriate for `bornprofile`.

Default values can be supplied in *kwargs*. This method picks unique parameter keys from the relevant sections of the run parameters file (i.e. *bornprofile*, *membrane*, and *environment*).

If args are provided, then either a single value corresponding to the key or a list of such values is returned instead.

get_bornprofilenomem_kwargs(**args*, ***kwargs*)
Return a dict with kwargs appropriate for `bornprofilenomem`.

Default values can be supplied in *kwargs*. This method picks unique parameter keys from the relevant sections of the run parameters file (i.e. *bornprofile*, *membrane*, and *environment*).

If args are provided, then either a single value corresponding to the key or a list of such values is returned instead.

write(*filename=None*)
Write the current parameters to *filename*.

`bornprofiler.bpio.`**float_or_None**(*s*)
Return *s* as float or None when x == "None".

`bornprofiler.bpio.`**path**(*s*)
Return *s* with user expansion.

`bornprofiler.bpio.`**readPoints**(*filename*)
Read coordinates from either data file or pdb.

Returns (N,3) array.

`bornprofiler.bpio.`**readPointsDat**(*filename*)
Read points from a simple data file.

Example:: # comment x y z x y z ...

bornprofiler.bpio.**readPointsPDB**(*filename*)
Read points form a PDB formatted file.

Takes x,y,z from any ATOM or HETATM record.

bornprofiler.bpio.**read_dat_to_array**(*datfile*)
Reads the dat file into a (4,N) numpy array

bornprofiler.bpio.**read_template**(*filename*)
Return *filename* as one string.

filename can be one of the template files.

1.4.6 Setting up logging — bornprofiler.log

Configure logging for the BornProfiler. Import this module if logging is desired in application code.

Logging to a file and the console.

See http://docs.python.org/library/logging.html?#logging-to-multiple-destinations

The top level logger of the library is named 'bornprofiler'. Note that we are configuring this logger with console output. If the root logger also does this then we will get two output lines to the console. We'll live with this because this is a simple convenience library and most people will not bother with a logger (I think...)

In modules that use loggers get a logger like so:: import logging logger = logging.getLogger('bornprofiler.MODULENAME')

class bornprofiler.log.**NullHandler**(*level=0*)
Silent Handler.

Useful as a default:: h = NullHandler() logging.getLogger("bornprofiler").addHandler(h) del h

Initializes the instance - basically setting the formatter to None and the filter list to empty.

emit(*record*)
Do whatever it takes to actually log the specified logging record.

This version is intended to be implemented by subclasses and so raises a NotImplementedError.

bornprofiler.log.**clear_handlers**(*logger*)
clean out handlers in the library top level logger

(only important for reload/debug cycles...)

bornprofiler.log.**create**(*logger_name='bornprofiler'*, *logfile='bornprofiler.log'*)
Create a top level logger.

- The file logger logs everything (including DEBUG).
- The console logger only logs INFO and above.

Logging to a file and the console.

See http://docs.python.org/library/logging.html?#logging-to-multiple-destinations

The top level logger of the library is named 'bornprofiler'. Note that we are configuring this logger with console output. If the root logger also does this then we will get two output lines to the console. We'll live with this because this is a simple convenience library...

1.4.7 Helper functions and classes — bornprofiler.utilities

The module defines some convenience functions and classes that are used in other modules.

Classes

class bornprofiler.utilities.**AttributeDict**
A dictionary with pythonic access to keys as attributes — useful for interactive work.

Functions

Some additional convenience functions that deal with files and directories:

bornprofiler.utilities.**openany**(*directory*[, *mode='r'*])
Context manager to open a compressed (bzip2, gzip) or plain file (uses *anyopen()*).

bornprofiler.utilities.**anyopen**(*datasource*, *mode='r'*)
Open datasource (gzipped, bzipped, uncompressed) and return a stream.

Arguments

- *datasource*: a file or a stream
- *mode*: 'r' or 'w'

bornprofiler.utilities.**realpath**(**args*)
Join all args and return the real path, rooted at /.

Expands '~', '~user', and environment variables such as $HOME.

Returns `None` if any of the args is `None`.

bornprofiler.utilities.**in_dir**(*directory*[, *create=True*])
Context manager to execute a code block in a directory.

- The *directory* is created if it does not exist (unless *create* = `False` is set)
- At the end or after an exception code always returns to the directory that was the current directory before entering the block.

bornprofiler.utilities.**find_first**(*filename*, *suffices=None*)
Find first *filename* with a suffix from *suffices*.

Arguments

filename base filename; this file name is checked first

suffices list of suffices that are tried in turn on the root of *filename*; can contain the ext separator (`os.path.extsep`) or not

Returns The first match or `None`.

bornprofiler.utilities.**withextsep**(*extensions*)
Return list in which each element is guaranteed to start with `os.path.extsep`.

Functions that improve list processing and which do *not* treat strings as lists:

bornprofiler.utilities.**iterable**(*obj*)
Returns `True` if *obj* can be iterated over and is *not* a string.

bornprofiler.utilities.**asiterable**(*obj*)
Returns obj so that it can be iterated over; a string is *not* treated as iterable

Functions that help handling files:

bornprofiler.utilities.**unlink_f**(*path*)
Unlink path but do not complain if file does not exist.

1.5 References

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Bibliography

[Rashin1985] A.Rashin and B.Honig, J Phys Chem B **89** (1985), 5588

[Pliego2000] J.R. Pliego and J.M. Riveros. Chemical Physics Letters, **332** (5-6): 597–602, 2000. doi: 10.1016/S0009-2614(00)01305-1

[Stelzl2014] Lukas S. Stelzl, Philip W. Fowler, Mark S.P. Sansom, and Oliver Beckstein. *Flexible Gates Generate Occluded Intermediates in the Transport Cycle of LacY*. J Mol Biol **426** (3), 2014, 735–751. doi: 10.1016/j.jmb.2013.10.024. PMCID: PMC3905165.

Python Module Index

b

Index